CHAPTER 4

EVOLUTIONARY COMBINATORIAL OPTIMISATION

It is more easy to destroy
than built,
to backward than to evolve...

classical proverb

Keywords: genetic algorithm, optimisation.

Why evolutionary algorithms and particularly Genetic Algorithm (GA)? This is a fundamental question, asked by many people. The classical answer is 'because it seems to work rather well'. However, there are a number of reasons why we choose to use GAs for design problems. GA are just one of many types of method known in computer science. It is not easy to define exactly which of these methods is best for which problem, except in a very broad sense. However, for a given problem, it is possible to identify methods that consistently produce good results (compared to results produced by other techniques). Indeed, the GAs fall into this category. Rather than spending time, effort and money developing new specialised computational techniques for every new problem, most developers prefer to re-use (inherit) an algorithm proven through extensive trials to be reusable and robust.

Since the 1960s there has been an increasing interest in imitating living being to develop powerful algorithms for optimisation problems. The term *'evolutionary computation'* is now in common use to refer to such techniques. The best known

algorithms in this class include GA. The first objective of this chapter is to make the reader familiar with the genetic algorithm technique. It begins by introducing the origin and the concepts of the GA. Then all the basic aspects (the philosophy) of the GAs are explained, without going into unnecessary details.

1. Causes of variability

When we look to the individuals of the same variety or sub-varieties of our cultivated plants or animals, one of the first points which strikes us is that they generally differ each from other. When we reflect on the vast *diversity* of the plants and animals, we are driven to conclude that this greater variability is simply due to the *non uniform conditions* of life. It seems clear that organic beings must be exposed during several *generations* to the new conditions of life to cause any appreciable amount of variation; and when the organism has once begun to vary, it generally continues to vary for many generations (see the work of Darwin (Burrow, 1979)).

Even when all the individuals exposed to certain conditions (*environment*) are affected in the same way, the change at first appears to be directly due to such conditions but in some cases it can be shown that quite opposite conditions produce similar changes. When among individuals, exposed to the same conditions deviations due to some combinations of circumstances, appears in the parent and in the child, the mere doctrine of *chance* almost compels us to attribute its appearance to *inheritance* (Blumberg, 1997). The laws governing inheritance are quite unknown. No one can exactly explains why the same peculiarity in different individuals of the same species, is sometimes inherited and sometimes not.

2. Natural evolution[1]

Let us consider the steps by which domestic races have been produced, either from one or several species. Some effect may be attributed to the direct action of external conditions of life, and some to habit. One of the remarkable features of our domesticated races is that we see in them *adaptation*, not to the animal's or plant's own good, but to human's use. We cannot suppose that all breeds were suddenly produced as perfect and as useful as we see them now–indeed, we know that this has not been their history. The key is the human power of accumulative selection. Nature gives successive variations (*desired or not*), and the human add them up in certain directions useful to him.

In the case of plants (*and many designs*) a gradual process of improvement is made through the occasional preservation of the best individuals. This improvement may be plainly recognised in the increased size and beauty which can be observed in the varieties of plants when compared with their parents. Evolution has produced some

[1] Natural evolution is a search for the fittest individual in species-space. The success of life on earth demonstrates the computational power of this search process...

marvellous and varied creatures–a highly successful forms of life. The natural evolution has acted through that enormous populations of creatures which has reproduced to generate new offspring that had inherited some features of their parents. The natural selection has played the role of filter to ensure the continuity of our life. It ensures that, on average, more successful creatures are produced each generation than less successful ones.

Only humans have the ability to go beyond instinct and consciously create designs. It is clear that human designs have progressed over time, this progressive refinement can accurately be described as *evolution*. Objects from the moment of their construction, have to perform some functions to survive. In the case of failure, they are discarded and forgotten quickly–artifacts have a limited life-span before they disappear. In a process very similar to reproduction in nature, these artifacts are replaced by new ones that use or 'inherit' features from existing ones. Just as in nature, where the environment is constantly changing, the problem which is intended to be solved is often constantly changing. For instance, at first the only purpose of the house was to shelter from wind and sun, then comfort and safety became important. Today, a house must also be equipped with electronic features and so on.

Since, the publication of Darwin's work (Burrow, 1979), many attempts have been done to understand the adaptive processes of natural systems. John Holland was one of the first who attempted to explain the adaptive processes of natural systems (Holland, 1975). He designed a *genetic algorithm* which is an artificial system based upon these natural systems. In the same time, other attempts were made to use the principles of natural selection within search process. Fogel introduced evolutionary programming (Fogel, 1966), while Rechenberg proposed the evolution strategies (see Schwefel, 1995). An introduction to metaheuristics can be found in (Pirlot, 1993).

It is now believed that for million years, designs have been evolved in nature. Biological designs exceed any human design in terms of complexity and efficiency. As biologists uncover more about the workings of creatures, it is becoming clear that many human designs that were believed to be original have existed in nature long before they were thought of by a human. Indeed, many of recent designs borrow their features somehow directly from nature. There are many examples in our real life–aircraft wings are an inspiration from birds. Designs are more and more *only evolution* of existing ones. The *homage* must be always paid to the first designer–the one who began from scratch.

Humans like other creatures have the ability to create designs. Perhaps the first human tool was only designed by sticking wood with rocks. It is clear that the design of these tools was slowly refined, until finely shaped axes and arrows were being produced. In less than a thousand years, humans progressed from being travelling on horses, to creatures capable of designing space-crafts with the ability to travel to other planets. Computers evolved from slow mechanical adding machines to electronic processing devices of unbelievable speed.
Nowadays, the design has grown and become more sophisticated than of our forefathers. The humans cannot anymore (*or hardly*) follow the rhythm of evolution.

Demands upon the designers to produce new and better designs has reached the top. With the advent of the computer, the human have to *comply with* the fact that computers can *help* them to make good design. The machine just do the hard part of the job. We are far from replacing *definitively* a human by a computer program.

3. System organisation

There are many search methods known to the computer science (see Appendix 2). The evolutionary search is a one of the recent sub-set. Search algorithms define a design problem in terms of search problem, where the search-space is a space filled with a set of points. Each point in that space defines a solution. The design problem is then transformed into the problem of searching for the best solutions somewhere in the space of valid ones.

3.1. Mapping

It is evident that one has to know what he is looking for before making any search. Loosely speaking, a human deciding to go somewhere has to be aware of his current position, then locate the place where to go. A detective working on a crime, defines it, fixes the goal which is to find a murderer, then follows a method (*which is subjective*) to seek this goal. The whole procedure is composed of three steps which are (1) define the problem, (2) fix the goal and (3) use a method to reach this goal. In general, these three components are more or less well defined except for complex problems. Generally, only the first component is defined in design problems, the goal and the procedure are hardly found. In the following, design and solutions will be confused since the aim of this study is the design using the natural evolution.

In tackling a search problem over some space of possible solutions, it is necessary to construct a *representation* of the possible solutions for manipulation and storage. Thus, before applying a genetic algorithm to any design problem, a certain mapping between human design and the evolutionary method must be made. Because GAs are heavily grounded in natural evolution, the terminology for biology has won over that of computer science. In order to facilitate the discussion some terminology must be introduced, such as gene, allele and locus.

The points in the search space are known as *phenotypes* while their representatives in the solution space are known as *genotypes*. The structures used to represent genotypes are known variously as *genomes* or *chromosomes*. The genotype, as it name implies, specifically refers to an individual's genetic structure. The phenotype refers to the observable appearance of an individual (pheno in Greek means 'to show'). The process of producing a phenotype from a genotype is known as *morphogenesis* (see Figure 4.1).
Generally, the standard chromosome used to represent a solution typically, but not invariably, takes the form of a simple string of values called *genes*–it is also referred to

as a feature or a character detector. More formally, a gene can be identified as an equivalence relation over the search space, and formally constructed from a characterisation of it. The particular values that each gene can take are called *alleles*. As instance, if the 'eye colour' gene can take values 'blue', 'green' and 'brown' then these are its three possible alleles. The position of a gene in its chromosome is its locus. A common example taken from biology is human represented by a chromosome, the skin colour is determined by a gene, the different skin colours are the gene's alleles (white, black, yellow, red) and where the skin colour happens to be in the chromosome is its locus.

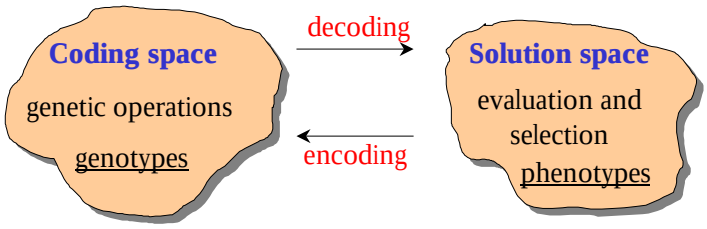

Figure 4.1 . Mapping between solution space and search space.

3.2. How does it works?

We will not dig here deeper into the mechanics of the GAs. The GA is a process that mimics the way biological evolution works. The usual form of the GAs is well described in (Goldberg, 1989). The GAs are stochastic search technique based on the mechanism of natural selection and natural evolution. There are many variations to the standard GA but they are all united by a common thread.

The biological terminology of the natural evolution transfers directly to the mathematical terminology of evolutionary computation. The GAs work in parallel with a certain number of chromosomes. The set of individuals of each generation is called a *population*. The chromosomes evolve through successive iterations (*generations*). Solutions are the individuals, and each individual is characterised by its *fitness*. In order to create new population, new chromosomes called *offspring*, are formed either by merging two chromosomes (*crossover operator*), or by modifying a chromosome (*mutation operator*).

A population of solutions to a given problem is maintained. Evolution plays the role of adaptation of a population to its environment. This adaptation causes the creation of individuals of increasingly higher 'fitness'. The best solutions are favoured for reproduction every generation, the offspring is then generated from these fit parents using crossover and mutation. Thus, evolution drives the population to better and better individuals (Holland, 1975).
In optimisation problems, one seeks to find the optimal solution with respect to certain criteria. In some cases the information about the optimal solution is missing, one is constrained to explore all the search space of the possible solutions to find the optimum. For the most complex problems (NP-complete), this space is so large and

complex that exact methods are seldom used, and one is often satisfied with a not optimal but ‘satisfactory’ solution.

Different search algorithms are appropriate for different type of search spaces. Simple search algorithms apply well to relatively simple search space (databases, arrays). Complex spaces require more advanced search algorithms. The GA is a method for searching for a solution in an arbitrary space and it is considered to be a sophisticated search algorithm for complex and poorly understood search spaces.

The GA differs from conventional search methods in several ways.

- GA works with coding of the parameters and not the parameters themselves.
- GA searches from a population of points not from a single point.
- GA uses objective function information (fitness) not derivatives or other auxiliary knowledge.
- GA uses probabilistic transitions rules and not deterministic ones.

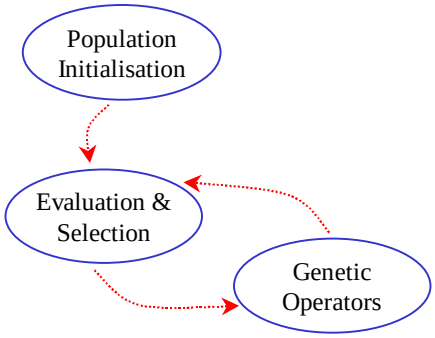

Figure 4.2. Principle of the genetic algorithm.

The standard GA can be summarised by the following steps (see Figure 4.2).

1. The GA can operate on any data type (*representation*). The representation determines the bounds of the search space, and should be complete expression of a solution to the problem. In general and particularly in constrained design problems, it is more desirable that the representation can only encode feasible solutions so that the objective function (*fitness*) measures only *optimality* and not *feasibility*.

2. The initial population, is created during an initialisation phase and it is often generated at random. In general some knowledge is used by the GA to start the search from promising regions of the search space.

3. Every member of the population is then evaluated and given a fitness value according to how well it fulfils the objective or fitness function. If there is no clear way to compare the quality of different solutions, then there can be no clear way for the GA to allocate more offspring to the fitter solutions.

4. The GA favours individuals with a higher overall fitness when picking 'parents' from the population. The fitness function allows the evaluation of potential solutions of the problem. These scores are then used to determine which individuals will participate to the creation of the new population.

5. Based on the fitness values, the GA selects candidate solutions and combines the best traits of the parents to produce superior children. This combination operator is called *crossover*. This operator determines how the space can be searched. Thus, it defines simple transition rules to construct new candidate solutions from old ones.

6. In a process very similar to the natural evolution, only *few* occasional invention '*mutation*' are discovered. Similarly a small part of the population is mutated. Single existing individuals are modified to produce a single new one. It is more likely to produce harmful or even destructive changes than beneficial ones. A GA with high mutation rate may be degenerated to a random search.

7. The natural *selection* ensures that weakest creatures die, or at least do not reproduce as successfully as the stronger creatures. In the same way, within a GA a population of solutions to the problem is maintained, with the 'fittest' solutions being favoured for reproduction. New generations are formed by selecting some parents and offspring and rejecting the less fittest so as to keep the population size constant.

8. A generation is a population at a particular iteration of the loop. This *iterative* process (selection, recombination, and mutation) continues until either the specified number of generations (i.e. loops) is passed, or until an acceptable solution has emerged.

The GA can be summarised as: the selection of individuals with simple underlying structure (strings) from a population with a preference for the fitter. Following by a recombination of these structures to yield a new generation of individuals with higher average of fitness...

In the next sections, we will show the existing similarities between the natural evolution, and the evolutionary computation–namely the genetic algorithms.

4. How does the GA imitate the natural evolution?

The GA attempts to apply the successful self-organising principles found in the evolution of natural systems to artificial systems. It mimics the natural selection and biological evolution to achieve its goals. A GA for a particular problem must be designed using the following steps.

1. A *representation* for potential solutions to the problem.
2. A way to create an *initial* population of potential solutions.

3. A *sampling mechanism* that plays the role of the environment, thus, rating the solutions in terms of their fitness.
4. Genetic *operators* that alter the composition of children.
5. Tune the various *parameters* the GA uses (population size, probabilities of applying the genetic operators, etc.).

A poorly designed GA may perform nearly as badly as a random search. In contrast, a carefully designed GA has the potential to perform well or better than any other search methods. Thus, the GAs are a paradigm that must be adapted to the task on hand (Falkenauer, 1999). These five components of GA will be detailed in the next sections.

For a given design problem, and for a GA as search method, the following elements have to be considered. First, specify the coding of the allowable solution (genotype). Second, the evolutionary approach must be chosen–the genetic operators. Third, the genotypes are mapped (decoding process) to solutions (phenotypes). Finally, these phenotypes are analysed and evaluated, this provides the search method with a fitness values for each solution. We will show in the next section, that these elements are problem dependent. Thus, the fact of studying each of them in depth, is far from being loosing time.

4.1. Representation

The first step in designing a GA for a particular problem is to devise a suitable representation. One can loosely speak about the problem description in the GA's *language*. In some cases this representation is simple while in others it is more involved. As instance, it is quite natural to represent an n-dimensional vector as a string of *n* values (genes), while it is difficult to represent a graph without introducing extra information, such as a labelling of the nodes.

4.1.1. Encoding

For many real-world industrial applications, the simple GA is difficult to apply directly because the binary string is not a natural coding. The binary alphabet is largely overstate even if in some problems it is not only unwarranted, more it is detrimental. Although GAs have been used in engineering problems, there are reasons why designers are reluctant to embrace the GAs. Indeed, most GAs were built to solve arbitrary functions. In order to avoid falling in term of the proverb *'what starts bad, will finishes bad...'*, the encoding must be adapted to the particular search problem on hand. Using a good *chromosomic* representation is the first step to narrow the gap between theory and practice in the context of engineering optimisation.

The nature has its own *indication* about the environment, it gets a minimal amount of response in terms of information. Many search methods–to be correct their

inventors–purport to be an universal problems solver... Many practitioners still have an understandable desire to own a *'magic bullet'* algorithm that can solve any problem (Culberson, 1998). It seems to be a persistent *blind* belief that it should be possible to have an algorithm that outperforms all others. Some adaptive algorithms claim that they perform well without special information from the environment. That means that they do well using any representation–no prior knowledge is necessary. The nature's behaviour is too complex to be imitated by black boxes. Thus, as long as we lack a valid representation of nature, it is timeless to try to imitate it.

The nature is *unknown* and changes with time, and by the way there is no certainty on the exactitude of the data involved. This makes place to the fuzzy logic[2] paradigm (Zadeh, 1965). Special care must also be taken when we deal with *online*[3] problems, the time response–the *speed* of the method–must be involved while choosing the suitable encoding (Brancke, 1999).

It is sure that a not well designed GA (*at least in the encoding point of view*) can be useless and should be abandoned in favour of other methods. In constrained search problems, the feasible region of the search space is so restricted that is renders the search very difficult. Once the search is no longer blind, that is, knowledge was integrated in the search system, then the GA can be enormously speeded up towards good achievements.

Another issue that affects the encoding choice is how is fast it is to access the data. Apart from the obvious benefits of speeding-up in execution time, which is the main reason evolutionary methods are used, fast access to the data can heavily facilitate the *code-writing* of the local improvement algorithms. The paralleling of the GAs may also changes the way the solutions are encoded (Culberson, 1998).

4.1.2. Feasibility

In real-world design problems, a set of *constraints* define the *feasible* region of the search space (solutions to the problem). Almost all problems involve arbitrary constraints that define feasible solutions. For instance, a solution to the 'bin packing' problem is infeasible when one or many bin capacity is exceeded. Often the set of feasible solutions is extremely small compared to the total solution space.
It is known that guessing feasible solutions within the large domain is a difficult task. For some problems, the GA may tests many proposed solutions without finding a feasible one. Sometimes, the infeasible region is so large that local search methods (exploitation) are the best way of search, exploration approaches will often reach the infeasible region. The GA has first to check whether the candidate solution is

---

[2] The fuzzy logic is a form of knowledge representation suitable for notions that cannot be defined precisely, but which depend upon their context. It enables computerised devices to reason more like humans.

[3] Static problems are those whose statement is subject to no change during the optimisation. Conversely, dynamic (on-line) problems are those whose statement can be modified during optimisation.

feasible. If it is not the case, it is a waste of time to evaluate its fitness especially in case of costly computing objective function. If a feasible candidate solution is found, a local search routine can be used to improve the solution quality.

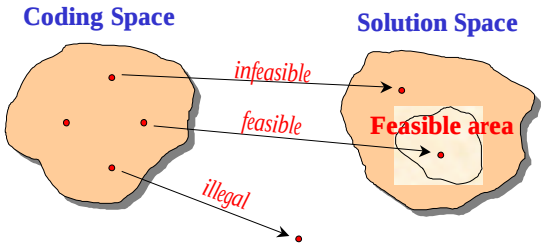

Figure 4.3. Feasibility of solutions.

Any random generation of feasible solutions in the case of constrained search spaces is a hopeless task. For some problems, feasible solutions cannot be reached even after a million of random attempts. If there are many infeasible solutions, one have to seek for a heuristic to produce feasible ones.

Three cases may occur while mapping between phenotype and genotype (see Figure 4.3):

1. *Feasible* area represents a set of normal solutions, which can be decoded form chromosomes. The valid solutions lie in the feasible region of the search space.
2. *Infeasible* areas arises especially in constrained problems. Many techniques are used to render infeasible solutions to feasible ones. In assembly line design, a solution where a station its process time exceeds the cycle time is declared as infeasible.
3. *Illegal* chromosomes can originates from the nature of the encoding as well as from the reproduction technique used. Illegal chromosomes cannot be decoded to a solution, then such genotype cannot be evaluated. A chromosome (solution) where a task is assigned to two different station is said to be illegal. Evolutionary methods creating illegal genotypes must be discarded no matter what price. The time wasted to repair illegal solutions may better be spent to create valid ones. Anyhow, when the illegality of genotypes occurs, immigration policy from illegal space to solution space must be found.

Each time an illegal solution is detected, there are three options available:

a. Prevent it ever from existing in the population. Every time the GA creates an illegal solution, that solution is discarded, the GA takes substantially longer to evolve solutions, since illegal ones are often discarded. The system simply gets trapped in an endless loop of creating and discarding illegal child solutions.

b. Correct the genotype. If illegal solutions are allowed to exist in a population, this is a kind of ‘genetic engineering’. The population of candidate solutions becomes

very static, with little evolution occurring. Indeed, almost all new solutions generated by crossover and mutation will be illegal designs. If these are corrected every time, the evolution being undone so the population stagnates.

c. A best compromise between discarding illegal genotypes and spending time correcting them, seems to correct only the phenotypes. This allows evolution to continue unconstrained, does not reduce the speed noticeably and avoids the population to stagnate, whilst still ensuring that only legal phenotypes exist. Hence, the system allows encoded illegal solutions to exist as genotypes, but corrects them when they are mapped to the phenotypes (Bentley, 1996).

In summary, the GAs may employ four basic strategies to deal with infeasible solutions: rejection, repair, modifying the genetic operator, and assigning penalties (Gen, 1997). The rejection strategy simply discards all infeasible individuals. The repairing strategy attempts to create feasible solutions out of an infeasible ones. For some problems, genetic operators can be modified so that they create only feasible solutions. Finally, penalty functions can be used when infeasible solutions can be recombined to form feasible solutions. The strategy to be chosen is highly dependent on the problem at hand.

4.1.3. Chromosomes and solutions spaces

Perhaps the most significant element of life is the process of generating a phenotype from *instructions* held within a genotype. Life uses efficient methods for mapping genotypes to phenotypes. Nature *evaluates* phenotypes and not genotypes. The GAs differ from traditional algorithms in the way they work. They manipulate a coding of the solutions and not the solutions themselves (Goldberg, 1989).

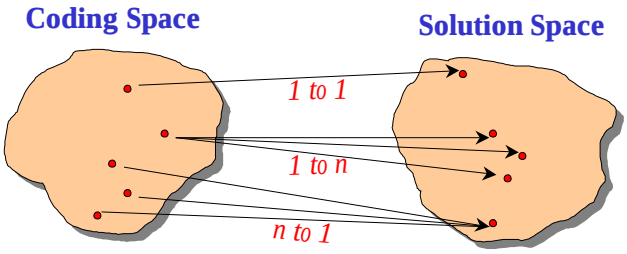

Figure 4.4. Mapping from the encoding to solutions.

In the GA, the representation needs to be unique, i.e. any solution can be defined in only one way. It must define only one solution at time and it must be able to express a wide range of solutions to the problem. Ideally, it should be accurate, i.e. all the objects defining the problem are represented without approximation.

If a genome can represent infeasible solutions, care must be taken into the objective function to give partial credit to the genome for its good genetic material while

sufficiently penalising it for being infeasible. Many approaches tend to attribute to each genotype a fitness and an unfitness terms. Infeasible solutions, tend to have higher unfitness scores (Chu, 1997).

It is clear that the *'1-to-1'* mapping is the best one among these cases. Each solution is represented by exactly one chromosome and each chromosome decodes in exactly one solution of the original problem. The *'1-to-n'* mapping is far the most undesired one. Such an encoding is redundant and its redundancy is a major blow to the efficiency of a GA. The *'n-to-1'* mapping, means that there are several solutions which are represented by only one chromosome. Such an encoding suffers from the lack of details, because some information is hidden from the GA (see Figure 4.4).

A real representation rather than a binary encoding is often used when applying GA to an optimisation problem with a real number domain. Many *practitioners* have found that using a representation natural to the problem domain is a good first choice.

4.2. Initialisation of the population

Most evolutionary design system permit the user to intervene during the initialisation process by specifying a set of desiderata. By initialising the first population with entirely random solutions, the GA receives a complete freedom to evolve any solutions that will fulfil the problem specification. The GA can create novel and potentially unconventional solution. Alternatively, the GA could be seeded with sub-solutions to the problem by using knowledge on the problem. The system thus evolves from both random and sub-solutions.

The traditional method of generating an *initial* population is to sample the search space at random. Another way is to use known heuristics. In constrained problems, the population can be created by only *mutating* the first created solution. Such hybrid methods provide significant enhancements to GAs. Some care must be taken to ensure that the used method provide sufficient diversity in the population.

When initialising the GAs with *random* values, the evolution makes extremely rapid progress at first. Indeed, most solutions are extremely different and belong to the different areas of the search space. There is some job for the GAs to locate the best regions. Over time, the population begins to converge, with the separate individuals resembling each other more and more (Davis, 1991). The GA narrows its search in the solution-space and reduces the changes made by evolution until eventually the population converges to a single solution. The mutation aims to create the diversity inside the population.

The GAs like other meta-heuristics claim to be general problem solvers. Though GA have been applied successfully to a wide range of different combinatorial optimisation problems, the time-consuming tuning (calibration) of the GA constitutes a major drawback. Ideally the GA-modeller should only design a coding scheme, specify a set of potential operators together with a range of their application

rates. After the completion of this creative part the modeller have to submit the *'tedious and tiring'* job of calibration to some kind of assistant, who operates using special procedural knowledge on how to produce high-quality solutions (Derigs, 1999). In our opinion, a successful design method of a GA to a specific problem must be viewed as two iterative stages: *generate, followed by the test process*. Any separation of the two phases yields unsuccessful GA.

4.3. Sampling mechanism

> *… This preservation of favourable variations and the rejection of injurious variations, I call natural selection… Charles Darwin (Burrow, 1979)*

The naturalist Charles Darwin defined the *natural selection* or *survival of the fittest* as 'the preservation of favourable individual differences and variations, and the destruction of those that are injurious... ' In nature, the individuals have to *adapt* to their environment in order to *survive* in a process called *evolution*, in which those features that make an individual more suited to compete are preserved when it reproduces, and those features that make it weaker are eliminated. When selection is the only mechanism at work, the best individual is eventually selected enough times to completely take over the population. The selection mechanism determines which individuals will have all or some of their *genetic material* passed to the next generation. The most commonly used selection schemes are reviewed in the following.

Roulette wheel selection (RWS)
This technique gets its name from the fact that the method works like a roulette wheel in which each slot on the wheel is paired with an individual of the population. The size of each slot is proportional to the corresponding individual fitness. The maximisation problems fit directly into the paradigm *larger slot implies larger fitness*.

Each individual is given a 'slice of a wheel' and a random number is generated, like rolling a ball on a roulette wheel, to select an individual. Individuals with a higher fitness have a better chance of being chosen, it is possible for an individual to be selected more than once, or not at all. Thus, the RWS is not the best way to implement selection point of view efficiency and fairness.

The proportional selection causes premature convergence by giving high selection probabilities to individuals with high fitness. Conversely, towards the end of the search, proportional selection may stagnate by not giving a selection probability to the best individuals much better than the population average. Therefore, it is common in the GAs to employ a scaling function to scale down the contribution of the best fit individuals early in the search and scale up their contribution later in the search (Goldberg, 1989).

Elitist models
The first variation called the *elitist* model enforces preserving the best solution. The *expected value model* reduces the stochastic errors of the selection mechanism. This is

done by introducing a count for each solution *s*, initially set to the $f(s)/\bar{f}$ value ($f(s)$ is the fitness value of solution *s* and $\bar{f}$ is the average fitness of the population) and decreased by *0.5* or *1* each time the solution is selected for reproduction with crossover or mutation respectively. Thus, when a chromosome count falls below zero, the solution is not available for selection any longer. The *elitist expected value* model is a combination of the 'elitist' and 'the expected value' models. In the *crowding factor model*, a newly generated solution replaces the old one and the doomed one is selected from those which resemble the new one (De Jong, 1975).

Stochastic methods

Techniques like *deterministic sampling*, *remainder stochastic sampling*, *stochastic tournament*, etc. have demonstrated their superiority on simple selection methods. Baker (Baker, 1987) provided a comprehensive theoretical study of these methods, and presented a method called *stochastic universal sampling* (SUS). This method uses a single wheel spin. This wheel, is spun with a number of equally spaced markers equal to the population size. The RWS generates a random number to select an individual for the next generation. Therefore, the SUS generates one random number and performs all the necessary selections to fill the next population by selecting individuals at a fixed interval from each other on the 'wheel'. This method gives each individual the proper number of trials, eliminating selection noise. However, the SUS does not control the *selection pressure*[4] which is a primary concern when maintaining diversity in GAs.

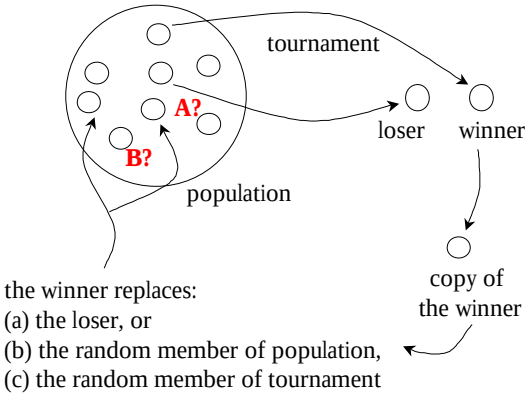

Figure 4.5. Tournament selection method.

Tournament selection

It uses the idea of ranking in an efficient way. At each iteration the method selects a number *k* (called tournament size) of individuals and selects the best one from this set into the next generation. This process is repeated *P* times (where *P* is the population size). It is clear, that large values of *k* increase the selective pressure of this procedure. The main features of the method are presented at Figure 4.5.

[4] The selection pressure is done either by selecting better solutions more often for reproduction, and/or by preferentially replacing less good solutions from the population.

Ranking selection

The solutions are selected proportionally to their *rank* rather than to their evaluation (Pareto optimality). The population is sorted from the best to the worst one, and each individual is copied as many times as it can, and then proportionate selection is performed. Such methods are more suited for multiple objective problems. A number of studies have been made to understand how to use such selection models in evolutionary techniques (Fonseca, 1995). Background to our work in this domain is described in Chapter 5.

The selection pressure refers to the probability of the best individuals to be selected. A strong selection pressure, gives rise to an 'aggressive search', in which there is strong *exploitation* of information gathered during search, but relatively little emphasis on *exploration* of up-to-here untested parts of the search space. The trade-off between exploitation and exploration is generally viewed as one of the key features in an effective search. The appropriate level of the selection pressure is a matter of much debate. It is widely accepted that a higher selection pressure lead to *fast convergence*, but also increases the likelihood of premature convergence (local optima). On the other hand, very low selection pressure increases the run-time and can even causes the *failure to improve* solutions (Gen, 1997).

All these selection schemes are normally preservative, that all the solutions are sometimes picked, no matter how poor their evaluation, and can operate in either a *steady-state* update scheme, in which a single or pair of solutions are produced at each step, or *generationally*, that is case some large portion of the population (most commonly the whole of it) is updated en masse.

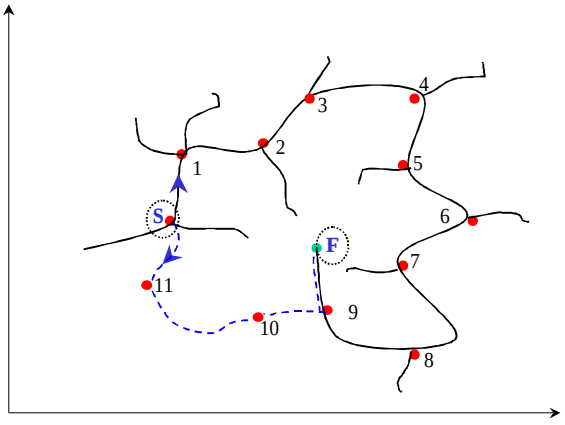

Figure 4.6. Selection and convergence speed.

Since the search space is most of the time large and complex, the selection decides about the evolution of the search mechanism. Figure 4.6 represents a two-dimensional research space. Suppose the first best solution found at the first generation is **S** (start solution). Suppose one wants to reach the best-ever solution which is represented by **F**. It is clear that depending on the chosen solution (1 or 11),

the *search trajectory* will be passing through the points {2, 3, 4, 5, 6, 7, 8, 9, F} in the case of 1 or {10, 9, F} in the case of 11. The convergence speed depends highly on the selection method.

Since the selection methods use the fitness to decide about the evolution, and since fitness is far from being well defined in some problems, this phenomenon still needs deeper research each time an evolutionary method has to be designed. In the next sections the most important features of GA the genetic operators will be introduced.

4.4. Genetic operators

The selection mechanism does not introduce any new solutions for consideration from the search space. It just copies some solutions to form an intermediate population. The second step of the evolution cycle is recombination (see Figure 4.2), it takes the responsibility of introducing new individuals into the population. This is done by the genetic operators: crossover, mutation and inversion.

4.4.1. Crossover

The critical attribute of the GAs is that they contain some sort of *reproduction* procedure. The simple version of this operator inherits individuals as they are. The most popular one is the crossover where two individuals are selected and are mated (crossed-over) in order to produce offspring. Information is extracted from the parents and is used to create offspring. The aim of crossover is to produce new solutions in regions of search space where successful ones have already been found.

The *schema theorem* of Holland (Holland, 1975) shows that the parts of solutions (schemata) that are observed to perform well (i.e. they are parts of good solutions) will be sampled with an increasing frequency[5]. The theorem shows that under a proportional selection, an above-average schema will receive an *exponentially* increasing number of copies over generations. The crossover permits: 1) to re-sample with an increasing frequency the same schemata that have been observed above-average, as well as 2) to avoid sampling the same points of the search space.

Crossover makes new instances of schemata already in the population and generates new ones that have not occurred yet. The crossover affects many schemata simultaneously. Holland called this parallel effect: *intrinsic parallelism*. Crossover disrupts shorter schemata less often than longer ones. This means that short fit schemata proliferate in the population and can be used as *building blocks* toward a solution.

There are a many variations of the crossover operator, the *'P-point crossover'* and the *'uniform crossover'* are probably the most common (Goldberg, 1989). In the *P*-point

[5] This is only valid for binary encoding (see (Holland, 1975) and (Falkenauer, 1998)).

crossover, each parent is divided at *P* locations into *P+1* contiguous sections, numbered *1* through *P+1*. Two offspring are created by exchanging every odd section between the two parents. The o*ne-point* and *two-point* crossover are probably the most commonly used type of *P*-point crossover. The uniform crossover can be thought of *P*-point crossover, where *P+1* is the number of genes in each parent. Therefore each gene is a section, and every section is probabilistically interchanged between the two parents. Other recombination operators take many shapes and forms, including crossover operators that use more than two parents to generate a single offspring.

The crossover gives the GA an advantage over other meta-heuristics. Without crossover, given the fact that the GA lacks the additional instruments of the *simulated annealing* (SA) (Van Laarhoven, 1987) or the *tabu search* (TS) (Glover, 1997) like temperature, tabu list, etc. the GA would have little chance to perform better than the other meta-heuristics.

4.4.2. Mutation

As pointed above, both the crossover and the mutation applied to chromosomes. Mutation introduces random changes to chromosomes. It is a mechanism that has only a small chance of occurring. This not means that the mutation operator is useless or 'bad' in any sense. Indeed, in absence of *diversity*, the crossover would again fall into the trap of non-representative sampling, because the progeny would be identical to the parents. The mutation also creates new points containing some of the old schemata.

The standard mutation operator *perturbs* offspring composition by changing a small number of alleles. Unlike crossover, mutation is a unary operator, it only acts on a single individual at a time. Some GAs use only the mutation operator–they do not perform any recombination. These GAs are roughly equivalent to running many SA algorithms in parallel, and are therefore *mutation-based* methods.

Mutation-based algorithms repeatedly modify a single individual to produce new solutions. In almost all mutation-based methods, tournament between the original and mutated solution is performed. The loser of each competition is discarded, and a new solution is then created by modifying the winner (local improvement). The winner and the new solution then compete in a new tournament, and the process is repeated as in the SA[6]. Other mutation-based algorithms, including the evolution strategy (Bäck, 1996) utilise the deterministic competitions that are always won by the competitor of higher fitness.

---

[6] In the simulated annealing (Kirkpatrick, 1983), the tournament is stochastic–i.e. the winner of each tournament is not necessarily the highest fitness individual. This allows the 'stochastic backtracking', where the algorithm can extract itself from dead-ends. The probability of accepting a lower fitness individual over a higher fitness individual (the 'acceptance probability') can be made to decrease over time.

Although mutation is important it is *secondary* to crossover. Many people have the erroneous belief that mutation plays the central role in natural evolution. The mutation is more likely to produce harmful or even destructive changes than beneficial ones. An environment with high mutation levels would quickly kill off most if not all the organisms. Without mutation the search is limited to solutions that can be formed by recombining the initial population. Mutation maintains *diversity* in the population and can be an answer to the question why sometimes the children differ from their parents.

4.4.3. Inversion

The inversion operator was first described in the original formulation of GA (Holland, 1975) but, over time, it seems to be disappeared from common usage. Inversion changes the genome linkage that is it moves together genes that were far apart.

The inversion is used to mitigate a *drawback* of the crossover operator. Since the crossing sites are picked at random (typically uniform), longer schemata are disrupted more often than shorter ones. Whenever one of the crossing sites falls between the genes which define the schema, a child will inherit only a part of the schema (unless the other part of the schema is inherited from the other parent). The main drawback is that with the fixed position of loci on the chromosome, it is always the same schemata which are more or less subject to *disruption*. That introduces an unwanted bias: we usually do not know which schemata should be inherited with preference, so fixing the position of loci in advance could mean that the important schemata end up to be the most disrupted. Inversion allows for shortening of long schemata, by rearranging the positions of loci on the chromosome. In the standard inversion operator, two sites are selected at random, and the order of the loci between the sites is reversed.

The inversion operator, when used, permutes schemata without changing their fitness. Similarly to crossover, one inversion affects many schemata simultaneously. This is another example of intrinsic parallelism. Thus, together crossover, mutation and inversion allow the GAs to discover over time fit, short and low order schemata.

5. Landscapes and fitness

Individuals of the population are chromosomes that represent the possible solutions of the problem. GAs work on a population of candidate solutions to the *objective function*. Applying a fitness function to each one of these chromosomes permits to measure the quality of the solution. Generally, the *fitness* of an individual reflects its performance. A fitness *landscape* is a set of points in n-dimension space (*hyper-surface*) obtained by applying the fitness function to every point in the search space.

Biological evolution does not necessarily drives the population to generate individuals of higher ability, unless this *ability* is strictly defined as a *propagative* success. Better individuals are simply those that produce the most reproducing offspring under a variety of environmental conditions. For an evolutionary algorithm to efficiently optimise a function, the fitness must be clearly defined, and higher fitness individuals must be explicitly promoted. As a result, if any other ability besides propagative success is desired, the GA must directly encourages the formation of individuals with the desired ability.

In a constrained optimisation problem, the input parameters are subject to a set of constraints that define the feasible region of the search space. The goal of global optimisation is to find the globally minimum or maximum solution within the set of all possible ones.

6. Population

The main difference between the GA and the classical search methods is the fact that they do not search from one single point, but from a population of points. Choosing an appropriate *population size* is a rather an intriguing problem. While a number of attempts have been made to provide theoretical estimates of appropriate size, no generally satisfactory approach has yet been found.

A number of population modes updating are used in GAs, the main approaches are the *steady-state* update and *generational* update. A generational update scheme is a population maintenance mechanism in which N children are produced from a population of size N to form the population at the next time-step (generation), and this new population of children completely replaces the parent population. In contrast, the steady-state approach, a single child is produced at each time-step, and this child replaces a single member of the old population.

Like in natural evolution, the maintenance of *diversity* contributes to the formation of better and better solutions. The most straightforward way to maintain population diversity is to *increase the population size*. On large problems, however, restrictions on the computer resources such as time and memory makes it infeasible to run GAs with the population size needed to maintain the required diversity. Another way is to *use a higher mutation rate*. However, high mutation rates are highly disruptive and makes the search behaves more like random search, which is a very poor search technique on complex problems. The population can be viewed as a very basic form of *learning*; the algorithms are storing explicit representations of *'good'* parts of the target space and non-represented regions are implicitly avoided and therefore considered *'bad'*.

7. Simple... but it works!

When a search is guaranteed to visit all solutions in a search space, it is called *'ergodic search'*. However, any algorithm can be modified to simply enumerate the entire

search space and therefore be ergodic; therefore ergodicity by itself is a rather uninteresting property. What is more important is efficiency–how quickly an algorithm can find good or even optimal solutions. The frequent question asked by many people is why the evolutionary computation are used to design problems–particularly the GAs–and *why it work*? The answer is not simple.

GAs are an attractive class of techniques for several reasons. They simultaneously search a population of points, rather than a single point. The transition rules applied by the GA allows the search to occasionally 'back-up' to get past local optima, and find better solutions. In addition, GAs only use the pay-off information of the objective function to determine which regions to explore, so no other information about the search space is needed. Finally, as the GAs model the inherently parallel processes of natural selection and evolution, they can easily be implemented on parallel computers. This allows the GAs to take advantage of the scalable processing power of the parallel computers to speed-up execution and solve complex problems. GAs have two primary tasks: (1) *explore* sufficiently a search space in order to locate promising regions and (2) *exploit* promising regions of the search space in order to focus the search towards the global optimum. The GAs tend to deal with these dual objectives by balancing exploration and exploitation.

Several applications of GAs to real-world combinatorial optimisation problems lead us to believe that GAs are good optimisation methods for some problems. Indeed, the reason why we prefer the GA to other heuristics is the concept of combining (*crossover*) parts of good solutions to produce new ones. But there is of course a *catch*. Indeed, the crossover cannot function properly in an arbitrary setup. The act of combination performed by the crossover must take into account the problem being solved.

Falkenauer (Falkenauer, 1999) showed that the *encoding* and the *genetic operators* must be adapted to the particular optimisation task on hand. In order to comply with that necessity in practice, we pledge to abandon functions as targets of GA optimisation, in profit of optimisation problems.

The GAs have been found to be some of the most flexible, efficient and robust of all search algorithms known to computer science. Because of these properties, these methods are now becoming widely used to solve a broad range of real-world problems.

8. References

(Baker, 1987) Baker J., 'Reducing bias and inefficiency in the selection algorithm', Proceedings of the second international conference on genetic algorithms, San Mateo, July 1987.

(Bäck, 1996) Bäck T., 'Evolutionary algorithms in theory and practice: evolution strategies, evolution programming, genetic algorithms', Oxford University Press, New York, 1996.

(Bentley, 1996) Bentley P. J., 'Generic evolutionary design of solid objects using a genetic algorithm', Ph.D. Thesis, Division of Computing and Control Systems, School of Engineering, University of Huddersfield, 1996.

(Blumberg, 1997) Blumberg R.B. 'MendelWeb', Netscape, University of Washington at Seattle, http://www.netspace.org/MendelWeb/Mendel.html, 1997.

(Brancke, 1999) Brancke J. 'Evolutionary algorithms for dynamic optimisation problems: a survey', Universität Karlsruhe (TH), Internal Report, 1999.

(Burrow, 1979) Burrow J.W., 'Darwin the origin of species', Pelican classics, Middlesex, England, 1979.

(Chu, 1997), Chu P.C.H., 'A genetic algorithm approach to combinatorial optimisation problems', Ph.D. Thesis, The Management School, Imperial College of Science, Technology and Medicine, London, England, 1997.

(Culberson, 1998) Culberson J.C., 'On the futility of blind search: an algorithmic view of "no free lunch"', Evolutionary Computation, Vol. 6(2), 109-127, 1998.

(Davis, 1991) Davis L., 'Handbook of genetic algorithms', Van Nostrand Reinhold, New York, 1991.

(De Jong, 1975) De Jong A. K., 'An analysis of the behavior of a class of genetic adaptive systems', Ph.D. thesis, University of Michigan, 1975.

(Derigs, 1999) Derigs U., Kabath M., Zils M., 'Adaptive genetic algorithms: a methodology for dynamic auto-configuration of genetic search algorithms', in Stafan Voss, Silvano Martello, Ibrahim H. Osman and Catherine Roucairol, Metha-Heuristics Advances and Trends in Local Paradigms for Optimisation, Kluwer Academic Publishers, 1999.

(Falkenauer, 1998) Falkenauer E., 'Genetic algorithms and grouping problems', John Wiley & Sons Inc., Chichester, First edition, 1998.

(Falkenauer, 1999) Falkenauer E., 'Applying evolutionary algorithms to real-world problems', In L.D. Davis, K. De Jong, M. Vose and L.D. Whitley (Eds.) Evolutionary Algorithms, IMA Volumes in Mathematics and its Applications, Vol. 111, Springer Verlag, 1999.

(Fogel, 1966) Fogel L.J., Owens A.J. and Walsh M.J., 'Artificial intelligence through simulated evolution', New York: Wiley, 1966.

(Fonseca, 1995) Fonseca C.M. and Fleming P.J., 'An overview of evolutionary algorithms in multiobjective optimisation', Evolutionary Computation, Vol. 3(1), pp. 1-16. 1995.

(Gen, 1997) Gen M. and Cheng R., 'Genetic algorithms & engineering design', John Wiley & Sons Inc., Canada, First Edition, 1997.

(Goldberg, 1989) Goldberg D. E., 'Genetic algorithms in search, optimisation and machine learning', Addison-Wesley Publishing Company. Inc, 1989.

(Glover, 1997) Glover F. and Laguna M., 'Tabu search', Kluwer Academic Publishers, Boston, 1997.

(Holland, 1975) Holland J.H., 'Adaptation in natural and artificial systems', University of Michigan Press, Ann Arbor, 1975.

(Kirkpatrick, 1983) Kirkpatrick S., Gelatt C.D. Jr. and Vecchi M.P., 'Optimisation by simulated annealing', Science, Vol. 220, pp. 671-680, 1983.

(Pirlot, 1993) Pirlot M., 'General local search in combinatorial optimization: a tutorial', Belgian Journal of Operations Research, Statistics and Computer Science , Vol. 32(1-2), pp. 7-67, 1993.

General local search methods, by M. Pirlot, in European Journal of Operations Research, Volume 92, Number (3), pp. 493-511, Aug. 9 1996.

(Schwefel, 1995) Schwefel, H.-P., "Evolution and optimum seeking", New York, NY: John Wiley, 1995.

(Van Laarhoven, 1987) Van Laarhoven P. J. and Aarts E. H., Simulated annealing: theory and applications", D. Reidel, Dordrecht, 1987.

(Zadeh, 1965) Zadeh Lotfi, "Fuzzy sets", Information and Control, Vol. 8, pp. 338-353, 1965.