CHAPTER 3

EVOLUTIONARY COMPUTING

"... one general law, leading to the advancement of all organic beings, namely, multiply, vary, let the strongest live and the weakest die."
(C. Darwin, The Origin of Species, 1859; Wordsworth Editions Limited (1998), p. 186).

"The work done by natural selection is R and D, so biology is fundamentally akin to engineering, a conclusion has been deeply resisted out of misplaced fear for what it might imply. In fact, it sheds light on some of our deepest puzzles. Once we adopt the engineering perspective, the central biological concept of function and the central philosophical concept of meaning can be explained and united. Since our own capacity to respond to and create meaning – our intelligence – is grounded in our status as advanced products of Darwinian processes, the distinction between real and artificial intelligence collapses. There are important differences, however, between the products of human engineering and the products of evolution, because of differences in the processes of evolution into focus, by directing products of our own technology, computers, onto the outstanding questions."
(D. Dennett, Darwin's Dangerous Idea: Evolution and the Meanings of Life, Penguin Books, 1995, p. 185–186)

"... nothing in biology makes sense, except in the light of evolution."
(T. Dobzhansky, The American Biology Teacher, 35, 1973, p. 125–129)

3.1 INTRODUCTION

E*volutionary computing*, also called *evolutionary computation*, is the field of research that draws ideas from evolutionary biology in order to develop search and optimization techniques for solving complex problems. Most *evolutionary algorithms* are rooted on the Darwinian theory of evolution. Darwin proposed that a population of individuals capable of reproducing and subjected to (genetic) variation followed by selection result in new populations of individuals increasingly more fit to their environment. Darwin's proposal was very radical at the time it was formalized, in the late 1850s, because it suggested that a simple algorithmic process of reproduction plus variation and natural selection was sufficient to produce complex life forms.

This simple theory for the origin and diversity of life resulted in the development of one of the most useful natural computing approaches to date, namely, the evolutionary algorithms (EAs). There are several types of EAs, among which the most classical ones are *genetic algorithms*, *evolution strategies*, *evolutionary programming*, and *genetic programming*. This chapter starts by describing what is problem-solving as a search task, and follows with an introduction to *hill-climbing* and the *simulated annealing* algorithm, some traditional search techniques. The motivation behind the simulated annealing algorithm is provided and it is theoretically compared with hill-climbing and some of its variations. This paves the ground for a better understanding of EAs. The focus of the chapter is on the *standard genetic algorithm*, but an overview of the other main evolutionary algorithms is also provided. To appropriately describe the inspiration behind all evolutionary algorithms, with particular emphasis on the genetic algorithm, some background on evolutionary genetics is provided.

3.2 PROBLEM SOLVING AS A SEARCH TASK

Under the evolutionary perspective to be studied in this chapter, a *problem* may be understood as a collection of information from which something (e.g., knowledge) will be extracted or inferred. For instance, consider the cases of a numeric function to be maximized, and the problem of allocating a number of classes to some set of students, known as a timetabling problem. In the first problem, some knowledge (information) about the function to be optimized is available (e.g., the function itself, $f(x) = x^3 + x + 3$), and the objective is to determine the values of x that maximize this function. In the timetabling problem, a lot of information might be available, such as the number of students, classrooms, teachers, and so forth. The objective may be, for example, to make a timetable for all the classes at a college in a semester, given the information available.

The process of *problem solving* corresponds to taking actions (steps), or sequences of actions (steps), that either lead to a desired performance or improve the relative performance of *individuals*. This process of looking for a desired performance or improved performances is called *search*. A search algorithm will take a problem as input and return a solution to it. In this case, one or more individuals, which could be viewed as *agents*, will be used as *candidate solutions* to the problem. Some knowledge about the desired performance of a given individual is not always available, but it might still be possible to evaluate the relative quality of individuals that are being used as means to solve the problem.

The first step in problem solving is the problem formulation, which will depend on the information available. Three main concepts are then involved in problem solving (Michalewicz and Fogel, 2000):

1) *Choice of a representation*: encoding of alternative candidate solutions (individuals) for manipulation. For each problem, the representation of a candidate solution is of paramount importance, and its corresponding interpretation implies the *search space* and its size. The search (or state) space is defined by the initial *state* (*configuration*) and the set of possible states (configurations) of the problem.
2) *Specification of the objective*: description of the purpose to be fulfilled. This is a mathematical statement of the task to be achieved. It is not a function, but rather an expression. For instance, if the objective is to minimize the function $f(x) = x^3 + x + 3$ given above, then the objective can be stated as follows: $\min f(x)$.

3) *Definition of an evaluation function*: a function that returns a specific value indicating the quality of any particular candidate solution (individual), given the representation. In cases when no knowledge about the desired performance of individuals is available, the evaluation function may be used as a means to evaluate the relative quality of individuals, thus allowing for the choice of one or more high quality individuals within a set of candidate solutions.

3.2.1. Defining a Search Problem

Given a search space S, assume the existence of some *constraints* that, if violated, avoid the implementation of a solution. In the search for improved solutions to a problem, we have to be able to move from one solution to the next without violating any of the constraints imposed by the problem formulation, that is, we need operators that generate or select *feasible* solutions. See Appendix B.4.1 for a brief review of optimization problems.

It is now possible to define a search (or optimization) problem (Michalewicz and Fogel, 2000). Given the search space S, together with its feasible part F, $F \subseteq S$, find $x^* \in F$ such that $eval(x^*) \leq eval(x)$, $\forall x \in F$.

In this case, the evaluation function that returns smaller values for x is considered better. Thus, the problem is one of *minimization*. However, we could just as easily use an evaluation function for which larger values are favored, thus turning the search problem into one of *maximization*. To maximize something is equivalent to minimize the negative of the same thing ($\max f(x) = \min -f(x)$). It is important to stress that the search process itself does not know what problem is being solved, as will be further discussed in Section 3.9.1. All it knows is the information provided by the evaluation function, the representation used, and how the possible solutions are sampled. If the evaluation function does not correspond with the objective, we will be searching for the right answer to the wrong problem!

The point x that satisfies the above condition is called a *global solution* or *global optimum*. Finding such a global solution to a problem might be very difficult, but sometimes finding the best solution is easier when it is possible to concentrate on a small portion of the search space. Effective search techniques have to provide a mechanism for balancing two apparently conflicting objectives: *exploiting* the best solutions found so far and at the same time *exploring* the search space. Among the search techniques to be studied, some (e.g., hill-climbing) exploit the best available solution but neglect exploring a large portion of the search space, while others (e.g., simulated annealing and genetic algorithms) combine exploration with exploitation of the search space. It is important to note, however, that it has been proved mathematically that there is no way to choose a single search method that can have an average superior performance in all runs for all problems; a theorem known as 'No Free Lunch' (Wolpert and Macready, 1997).

It is possible, on the other side, to assess and compare the performance of different algorithms in specific problem domains and, therefore, it is possible to look for a technique that provides the best performance, on average, in this domain.

In contrast to the global optimum, a *local optimum* is a potential solution $x \in F$ in respect to a neighborhood N of a point y, if and only if $eval(x) \leq eval(y)$, $\forall\, y \in N(x)$, where $N(x) = \{y \in F : dist(x,y) \leq \varepsilon\}$, *dist* is a function that determines the distance between x and y, and ε is a positive constant. Figure 3.1 illustrates a function with several local optima (minima) solutions, a single global optimum (minimum), and the neighborhood of radius ε around one of the local optima solutions.

The evaluation function defines a response surface, which will later be termed *fitness landscape* (Section 3.4.4), that is much like a topography of hills and valleys. The problem of finding the (best) solution is thus the one of searching for a peak, assuming a maximization problem, in such a fitness landscape. If the goal is that of minimization, then a valley has to be searched for. Sampling new points in this landscape is basically made in the immediate vicinity of current points, thus it is only possible to take local decisions about where to search next in the landscape. If the search is always performed uphill, it might eventually reach a peak, but this might not be the highest peak in the landscape (global optimum). The search might sometimes go downhill in order to find a point that will eventually lead to the global optimum. Therefore, by not knowing the fitness landscape and using only local information, it is not possible to guarantee the achievement of global optima solutions.

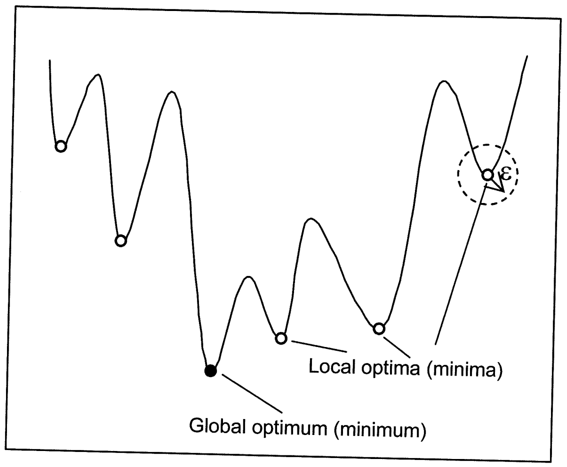


Figure 3.1: Illustration of global and local minima of an arbitrary function. Dark circle: global minimum; White circles: local minima.

3.3 HILL CLIMBING AND SIMULATED ANNEALING

This section introduces two standard search techniques: *hill-climbing* and *simulated annealing*. The simulated annealing algorithm can be seen as a sort of probabilistic hill-climber, and it can also be viewed as a special case of an evolutionary algorithm, which is the main subject of this chapter. Additionally, the study of these two procedures serves as a preparation for the study of evolutionary algorithms, a better understanding of their behavior, and for performance comparisons as well.

3.3.1. Hill Climbing

Hill climbing is a local search method that uses an *iterative improvement* strategy. The strategy is applied to a single point - the current point (or state) - in the search space. At each iteration, a new point **x**' is selected by performing a small displacement or perturbation in the current point **x**, i.e., the new point is selected in the neighborhood of the current point: **x**' $\in N$(**x**). Depending on the representation used for **x**, this can be implemented by simply adding a small random number, Δ**x**, to the current value of **x**: **x**' = **x** + Δ**x**.

If that new point provides a better value for the evaluation function, then the new point becomes the current point. Else, some other displacement is promoted in the current point (a new neighbor is chosen) and tested against its previous value. Termination occurs when one of the following *stopping criteria* is met:

- No further improvement can be achieved.
- A fixed number of iterations have been performed.
- A goal point is attained.

Let **x** be the current point, **g** the goal point (assuming it is known), and `max_it` a maximum number of iterations allowed. Algorithm 3.1 contains the pseudocode of a standard (simple) hill-climbing algorithm.

```
procedure [x] = hill-climbing(max_it,g)
  initialize x
  eval(x)
  t ← 1
  while t < max_it & x != g & no_improvement do,
      x' ← perturb(x)
      eval(x')
      if eval(x') is better than eval(x),
           then x ← x'
      end if
      t ← t + 1
  end while
end procedure
```

Algorithm 3.1: A standard (simple) hill-climbing procedure.

```
procedure [best] = IHC(n_start,max_it,g)
  initialize best
  t1 ← 1
  while t1 < n_start & best != g do,
    initialize x
    eval(x)
    x ← hill-climbing(max_it,g) //Algorithm 1
    t1 ← t1 + 1
    if x is better than best,
        then best ← x
    end if
  end while
end procedure
```

Algorithm 3.2: An iterated hill-climbing procedure.

Hill climbing algorithms have some weaknesses. First, they usually terminate at local optima solutions. Second, there is no information about the distance from the solution found and the global optimum. Third, the optimum found depends on the initial configuration. Finally, it is generally not possible to provide an upper bound for the computational time of the algorithm.

An important aspect of this algorithm, however, is that it performs a *blind search*, i.e., the current point is randomly perturbed and evaluated to check for an improvement of the objective function (see discussion in Section 3.9.1). Note also that hill-climbing methods can only converge to local optima solutions, and these values are dependent upon the starting point. Furthermore, as the global optimum is usually unknown, there is no general procedure to bound the relative error with respect to it. As the algorithm only provides local optima solutions, it is reasonable to start hill-climbing methods from a large variety of points. The hope is that at least one of these initial locations will have a path leading to the global optimum. The initial points might be chosen randomly, or by using a regular grid or pattern, or even by using other types of information. In the case the standard hill-climbing algorithm allows for multiple initializations and maintains a 'memory' of the best solution found so far, we have the so-called *iterated hill-climbing*, as described in Algorithm 3.2 (The input variable `n_start` is the number of different starting points to be used).

As an alternative stopping criterion, it is possible to verify if the algorithm reached a local optima solution by comparing the current value of **x** (or *best*) with some of its previous values. If the value of **x** (*best*) does not change significantly after a number of iterations, then the algorithm can be said to have converged to a local optimum and the process can be halted.

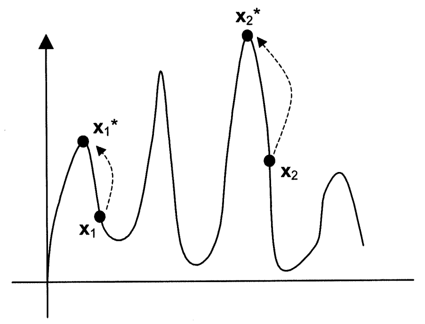


Figure 3.2: An illustrative example of the behavior of the basic hill-climbing algorithm.

There are a number of variations for the hill-climbing algorithm, and the ones discussed here are the simplest cases. Figure 3.2 illustrates the performance of the iterated hill-climbing algorithm assuming only two different initial points, $\mathbf{x}_1$ and $\mathbf{x}_2$. This function is uni-dimensional and contains only four peaks, with the third peak from left to right corresponding to the global optimum (the other peaks are all local optima solutions). If the first start point is at $\mathbf{x}_1$, then the inner loop of the algorithm converges to the local optimum $\mathbf{x}_1$*. In the second start, point $\mathbf{x}_2$, the global optimum will be determined and *best* = $\mathbf{x}_2$*.

Note that in practical applications the number of local optima solutions is very high, making such pure a trial and error strategy almost always unfeasible. Nevertheless, the good exploitation capability of hill-climbing has been used in combination with many other techniques capable of performing a better exploration of the search space.

The standard hill-climbing procedure can be modified in order to accommodate a probabilistic selection of the perturbed point **x**'. The probability that **x**' is selected depends upon the relative merit of **x** and **x**', i.e., the difference between the values returned by the evaluation function for these two points. This modification leads to a new algorithm called *stochastic hill-climbing* procedure, as described in Algorithm 3.3, with the capability of escaping from local optima solutions. The parameter *T* is a control parameter for the decay of the exponential function, and it remains constant along the iterations.

In Algorithm 3.3, by looking at the probability *P* of accepting the new point **x**', *P* = `1/(1+exp[(eval(x)-eval(x'))/T])`, it is possible to note that the greater the value of *T*, the smaller the importance of the relative difference between the evaluation of **x** and **x**'. If *T* is very large, then the search becomes similar to a random search.

```
procedure [x] = stochastic hill-climbing(max_it,g)
  initialize x
  eval(x)
  t ← 1
  while t < max_it & x != g do,
     x' ← perturb(x)
     eval(x')
     if random[0,1) < (1/(1+exp[(eval(x)-eval(x'))/T])),
          then x ← x'
     end if
     t ← t + 1
  end while
end procedure
```

Algorithm 3.3: A stochastic hill-climbing procedure.

3.3.2. Simulated Annealing

The *simulated annealing* (SA) algorithm, proposed by Kirkpatrick et al. (1983), was inspired by the *annealing* process of physical systems. Annealing corresponds to subjecting a material (e.g., glass or metal) to a process of heating and slow cooling in order to toughen and reduce brittleness.

The SA algorithm is based upon that of Metropolis et al. (1953), which was originally proposed as a means of finding the equilibrium configuration of a collection of atoms at a given temperature. The basic ideas of SA were taken from *statistical thermodynamics*, the branch of physics that makes theoretical predictions about the behavior of macroscopic systems (both, liquid and solid) on the basis of laws governing its component atoms.

Although the connection between the Metropolis algorithm and optimization problems had already been noted by Pincus (1970), it was Kirkpatrick et al. (1983) who proposed that it could form the basis of a general-purpose search (optimization) technique to solve combinatorial problems. The authors realized that there is a useful connection between statistical thermodynamics and multivariate combinatorial optimization. A detailed analogy with annealing in solids could provide a framework to develop an optimization algorithm capable of escaping local optima solutions (configurations).

A fundamental question in statistical thermodynamics was raised concerning what happens to a system in the limit of low temperature. Do the atoms remain fluid or solidify? If they solidify, do they form a crystalline structure or a glass? The SA algorithm was instantiated with a procedure used to take a material to its ground state; that is, to a state of lowest energy in which a crystal, instead of a glass, can be grown from a melt. The material is first heated to a high temperature so that it melts and the atoms can move freely. The temperature of the melt is then slowly lowered so that at each temperature the atoms can move enough to begin adopting a more stable orientation. If the melt is cooled down slowly enough, the atoms are able to rest in the most stable orientation, producing a crystal. This heating followed by a slow cooling process is known as *annealing*.

Kirkpatrick et al. (1983) proposed that when it is possible to determine the energy of a system, the process of finding its low temperature is akin to combinatorial optimization. Nevertheless, the concept of temperature of a physical system had no obvious equivalent in a problem being solved (optimized). By contrast, any cost function could play the role of the energy of the system. It remained thus important to propose an equivalent to the temperature parameter when dealing with optimization problems.

Basic Principles of Statistical Thermodynamics

In *statistical thermodynamics* large systems at a given temperature approach spontaneously the equilibrium state, characterized by a mean value of energy, depending on the temperature. By simulating the transition to the equilibrium and decreasing the temperature, it is possible to find smaller and smaller values of the mean energy of the system (Černý, 1985).

Let **x** be the current configuration of the system, $E(\mathbf{x})$ the energy of **x**, and T the temperature. The equilibrium is by no means a static situation. In equilibrium, the system randomly changes its state from one possible configuration to another in such a way that the probability of finding the system in a particular configuration is given by the Boltzmann-Gibbs distribution:

$$P(\mathbf{x}) = K.\exp(-E(\mathbf{x})/T) \qquad \textbf{(3.1)}$$

where K is a constant. If we assume a system with a discrete number of possible states, then the mean energy E_m of the system in equilibrium is given by

$$E_m = \left[\sum_{conf} E_{conf} \exp(-E_{conf}/T) \Big/ \sum_{conf} \exp(-E_{conf}/T) \right] \qquad \textbf{(3.2)}$$

The numerical calculation of E_m might be quite difficult if the number of configurations is high. However, it is possible to employ a Monte Carlo simulation of the random changes of state from one configuration to another such that, in equilibrium, Equation (3.1) holds. One such algorithm is the Metropolis et al. (1953) procedure.

The Simulated Annealing Algorithm

Assuming the general problem of minimizing a function, the simulated annealing algorithm works as follows. Having a current configuration **x**, this is given a small random displacement, resulting in **x**′, where the point **x**′ is chosen in the neighborhood of **x**. The energy $E(\mathbf{x}')$ of this new configuration is computed, and one has to decide whether to accept **x**′ as the new configuration, or to reject it. The resulting change in the energy of the system, $\Delta E = E(\mathbf{x}') - E(\mathbf{x})$, is calculated. If $\Delta E \le 0$, then the displacement is accepted and the configuration **x**′ is used as the starting point for the next iteration step. If $\Delta E > 0$, the probability that the configuration **x**′ is accepted is given by Equation (3.3), which is a particular case of the Boltzmann-Gibbs distribution (Equation (3.1)).

$$P(\Delta E) = \exp(-\Delta E/T) \tag{3.3}$$

To implement the random portion of the procedure, one can simply generate a random number r = `random[0,1)`, sampled over the interval [0,1), and compare it with $P(\Delta E)$. If $r < P(\Delta E)$, the new configuration **x**′ is retained, else it is discarded and **x** is maintained in the next iteration of the algorithm.

By repeating the basic steps described in the paragraph above, one is simulating the thermal motion of atoms in thermal contact with a heat bath at temperature T. By using $P(\Delta E)$ as described by Equation (3.3), the system evolves into a Boltzmann distribution.

To apply this algorithm as a general-purpose optimization procedure, the energy of the system is replaced by an evaluation function and the configurations are defined by a set of variables for this function. The temperature is a *control parameter* in the same units as the evaluation function. As annealing corresponds to first heating and then slowly cooling down the system, the temperature T in the simulation usually starts with a high value and is decreased during the iterative procedure. At each temperature, the simulation must proceed long enough for the system to reach a steady state. The sequence of temperatures and the number of rearrangements of the parameters attempted to reach equilibrium at each temperature are termed *annealing schedule* (Kirkpatrick et al., 1983).

Let **x** be the current configuration of the system, **x**′ be the configuration **x** after a small random displacement, and T the temperature of the system. The standard simulated annealing algorithm for solving minimization problems is described in Algorithm 3.4. The function $g(T, t)$ is responsible for reducing the value of the temperature. Usually a geometrical decrease is employed, e.g., $T \leftarrow \beta.T$, where $\beta < 1$.

```
procedure [x] = simulated_annealing(g)
  initialize T
  initialize x
  eval(x)
  t ← 1
  while not_stopping_criterion do,
      x′ ← perturb(x)
      eval(x′)
      if eval(x′) is less than eval(x),
           then x ← x′
      else if random[0,1) < exp[(eval(x)-eval(x′))/T],
           then x ← x′
      end if
      T ← g(T,t)
      t ← t + 1
    end while
end procedure
```

Algorithm 3.4: The simulated annealing procedure.

```
Step 1:     initialize T
            initialize x
Step 2:     x′ ← perturb(x)
            if eval(x′) is less than eval(x),
                   then x ← x′
            else if random[0,1) < exp[(eval(x)-eval(x′))/T]
                   then x ← x′
            end if
            repeat this step k times
Step 3:     T ← β.T
            if T ≥ T_min
                   then goto Step 2
            else goto Step 1
```

Algorithm 3.5: Typical implementation of the simulated annealing procedure.

The Metropolis algorithm can be viewed as a *generalized iterative improvement* method, in which controlled uphill steps can also be incorporated into the search. Also, this algorithm is very similar to the stochastic hill-climber, with the difference that it allows changes in the parameter T during the run. Most implementations of the simulated annealing algorithm follow the simple sequence of steps (Michalewicz and Fogel, 2000) presented in Algorithm 3.5.

In this typical implementation (Algorithm 3.5) the algorithm is initialized a number of times, whenever the temperature reaches is minimal value (frozen temperature). Also, for each temperature value the algorithm is run a number k of times.

From Statistical Thermodynamics to Computing

In the previous sections the simulated annealing algorithm was described together with its inspiration in physical systems. Table 3.1 summarizes how to interpret the terminology from the physics domain into the one used in the simulated annealing algorithm.

Table 3.1: Interpretation from the physics terminology into the computational domain.

Physics	Simulated Annealing Algorithm
State	(Feasible) solution to the problem, also called point in the search space
Energy	Value returned by the evaluation function
Equilibrium state	Local optimum
Ground state	Global optimum
Temperature	Control parameter
Annealing	Search by reducing T
Boltzmann-Gibbs distribution	Probability of selecting a new point

3.3.3. Example of Application

Consider the uni-dimensional function $g(x)$ presented in Figure 3.3. The variable x is defined over the interval [0,1], $x \in [0,1]$, and assume the global maximum of this function is unknown. It was discussed above that to optimize this function it is necessary to have a well-defined representation, objective function, and evaluation function.

Representation. The most straightforward representation, in this case, is to use the real value of variable x to represent the candidate solutions to the problem. Another representation can be obtained by using a bitstring data structure, as will be described in detail in Section 3.5.2. In order to perturb the current point, i.e., to generate a candidate point in the neighborhood of the current point, a Gaussian random noise of zero mean and small variance, $G(0,\sigma)$, can be added to the current point. If the added noise results in an individual that lies within the domain of x, $x \in [0,1]$, then accept it; else, discard it: $x' = x + G(0,\sigma)$, where σ is a small positive constant. The same could be done with a uniform, instead of Gaussian, distribution.

Objective. The objective for this problem is to find the maximal value of the function $g(x)$; that is, `max` $g(x)$.

Evaluation. The evaluation function used to determine the relative quality of candidate solutions is obtained by simply evaluating the function $g(x)$ for the values of x.

By using real values over the interval [0,1] to represent the variable x, all hill-climbing procedures described (simple, iterated, and stochastic) and the simulated annealing algorithm can be implemented and applied to find the global maximum of $g(x)$. For the simple hill-climbing, different initial configurations can be tried as attempts at finding the global optimum.

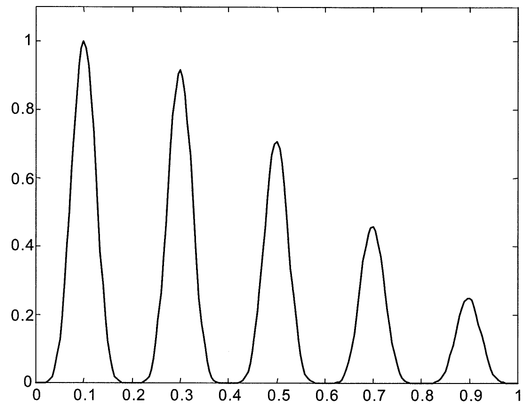


Figure 3.3: Graph of the function $g(x) = 2^{-2((x-0.1)/0.9)^2}(\sin(5\pi x))^6$ to be maximized.

3.4 EVOLUTIONARY BIOLOGY

Evolutionary biology is a science concerned, among other things, with the study of the diversity of life, the differences and similarities among organisms, and the adaptive and non-adaptive characteristics of organisms. Its importance are manifold, from the health sciences to the understanding of how the living organisms adapt to the environment they inhabit. For instance, evolutionary biology helps in the understanding of disease epidemics, population dynamics, and the production of improved cultures. Over roughly the last 60 years, computer scientists and engineers realized that evolutionary biology has various interesting ideas for the development of theoretical models of evolution (some of them being rather abstract models) that can be useful to obtain solutions to complex real-world problems.

The word *evolution* is originated from the Latin *evolvere*, which means to unfold or unroll. Broadly speaking, evolution is a synonym for 'change'. But what type of change? We do not usually employ the word evolution to refer to the changes suffered by an individual during its lifetime. Instead, an evolving system corresponds to the one in which there is a descent of entities over time, one generation after the other, and in which characteristics of the entities differ across generations (Futuyma, 1998). Therefore, *evolution* can be broadly defined as *descent with modification* and often *with diversification*. Many systems can be classified as evolutionary: languages, cellular reproduction in immune systems, cuisines, automobiles, and so on.

Any evolutionary system presents a number of features:

- *Population(s)*: in all evolutionary systems there are populations, or groups, of entities, generally termed *individuals*.
- *Reproduction*: in order for evolution to occur, the individuals of the population(s) must reproduce either sexually or asexually.
- *Variation*: there is variation in one or more characteristics of the individuals of the population(s).
- *Hereditary similarity*: parent and offspring individuals present similar characteristics. Over the course of generations, there may be changes in the proportions of individuals with different characteristics within a population; a process called *descent with modification*.
- *Sorting of variations*: among the sorting processes, it can be emphasized *chance* (random variation in the survival or reproduction of different variants), and *natural selection* (consistent, non-random differences among variants in their rates of survival and reproduction).

Adaptation as a result of variation plus natural selection leads to improvement in the function of an organism and its many component parts. "Biological *or* organic evolution *is change in the properties of populations of organisms, or groups of such populations, over the course of generations*." (Futuyma, 1998; p. 4). Note that according to this definition of evolution, individual organisms do not evolve and the changes of a population of individuals that are assumed to be evolutionary are those resultant from inheritance, via the genetic material, from one generation to the other.

The history of evolutionary biology is marked by a number of hypotheses and theories about how life on earth appeared and evolved. The most influential theory to date is the one proposed by Charles Darwin and formalized in his book *On the Origins of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life* (Darwin, 1859). Historically, Alfred Wallace is also one of the proponents of the theory of evolution by means of natural selection, but it was Darwin's book, with its hundreds of instances and arguments supporting natural selection, the landmark for the theory of evolution.

Among the many preDarwinian hypotheses for the origin and development of beings, the one proposed by Jean Baptist Pierre Antoine de Monet, chevalier de Lamarck, was the most influential. According to Lamarck, every species originated individually by spontaneous generation. A 'nervous fluid' acts within each species, causing it to progress up the chain over time, along a single predetermined path that every species is destined to follow. No extinction has occurred: fossil species are still with us, but have been transformed. According to Lamarck, species also adapt to their environments, the more strongly exercised organs attract more of the nervous fluid, thus getting enlarged; conversely, the less used organs become smaller. These alterations, acquired during an individual's lifetime through its activities, are inherited. Like everyone at that time, Lamarck believed in the so-called *inheritance of acquired characteristics*.

The most famous example of Lamarck's theory is the giraffe: according to Lamarck, giraffes need long necks to reach the foliage above them; because they are constantly stretching upward, the necks grow longer; these longer necks are inherited; and over the course of generations the necks of giraffes get longer and longer. Note that the theory of inheritance of acquired characteristics is not Lamarck's original, but an already established supplement to his theory of 'organic progression' in which spontaneous generation and a chain of beings (progression from inanimate to barely animate forms of life, through plants and invertebrates, up to the higher forms) form the basis. Lamarck's theory may also be viewed as a *transformational theory*, in which change is programmed into every member of the species.

3.4.1. On the Theory of Evolution

Darwin's studies of the natural world showed a striking diversity of observations over the animal and vegetal kingdoms. His examples were very wide ranging, from domestic pigeons, dogs, and horses, to some rare plants. His research that resulted in the book *Origin of Species* took literally decades to be concluded and formalized.

In contrast to the Lamarckian theory, Darwin was certain that the direct effects of the conditions of life were unimportant for the variability of species.

> "Seedlings from the same fruit, and the young of the same litter, sometimes differ considerably from each other, though both the young and the parents … have apparently been exposed to exactly the same conditions of life; and this shows how unimportant the direct effects of the conditions of life are in comparison with the laws of reproduction, and of growth, and of inheritance; for had the action of the conditions been direct, if any of the young had varied, all would probably have varied in the same manner." (Darwin, 1859; p. 10)

Darwin starts his thesis of how species are formed free in nature by suggesting that the most abundant species (those that range widely over the world) are the most diffused and which often produce well-marked varieties of individuals over the generations. He describes some basic rules that promote improvements in organisms: reproduce, change and compete for survival.

Natural selection was the term used by Darwin to explain how new characters arising from variations are preserved. He starts thus paving the ground to his theory that slight differences in organisms accumulated over many successive generations might result in the appearance of completely new and more adapted species to their environment. As defended by himself

> "… as a general rule, I cannot doubt that the continued selection of slight variations … will produce races differing from each other …" (Darwin, 1859; p. 28) and "… I am convinced that the accumulative action of Selection, whether applied methodically and more quickly, or unconsciously and more slowly, but more efficiently, is by far the predominant Power." (Darwin, 1859; p. 35)

In summary, according to Darwin's theory, evolution is a result of a population of individuals that suffer:

- Reproduction with inheritance.
- Variation.
- Natural selection.

These very same processes constitute the core of all evolutionary algorithms. Before going into the details as to how reproduction and variation happen within individuals and species of individuals, some comments about why Darwin's theory was so revolutionary and 'dangerous' at that time (and, to some people, until nowadays) will be made.

3.4.2. Darwin's Dangerous Idea

Darwin's theory of evolution is controversial and has been refuted by many because it presents a sound argument for how a "Nonintelligent Artificer" could produce the wonderful forms and organisms we see in nature. To D. Dennett (1991), *Darwin's dangerous idea* is that evolution, thus life, can be explained as the product of an *algorithmic process*, not of a superior being (God) creating everything that might look wonderful to our eyes. But the reason there is a section on Dennett's book here is not to discuss particular beliefs. Instead, to discourse about some key interpretations of evolution, from a computational perspective, presented by D. Dennett in his book *Darwin's Dangerous Idea: Evo-*

lution and the Meanings of Life. These are not only interesting, but also useful for the understanding of why the theory of evolution is suitable for the comprehension and development of a class of search techniques known as evolutionary algorithms.

Dennett defines an algorithm as a certain sort of formal process that can be counted on (logically) to yield a certain sort of result whenever it is run or instantiated. He emphasizes that *evolution* can be understood and represented in an abstract and common terminology as an *algorithmic process*; it can be lifted out of its home base in biology. Evolutionary algorithms are thus those that embody the major processes involved in the theory of evolution: a population of individuals that *reproduce with inheritance*, and suffer *variation* and *natural selection*.

Dennett also discusses what can be the outcomes of evolution and its probable implications when viewed as an *engineering process*. He stresses the importance of genetic variation and selection, and quotes an interesting passage from M. Eigen (1992).

> "Selection is more like a particularly subtle demon that has operated on the different steps up to life, and operates today at the different levels of life, with a set of highly original tricks. Above all, it is highly active, driven by an internal feedback mechanism that searches in a very discriminating manner for the best route to optimal performance, not because it possesses an inherent drive towards any predestined goal, but simply by virtue of its inherent non-linear mechanism, which gives the appearance of goal-directedness." (Eigen, 1992; quoted by Dennett, 1991, p. 195)

Another important argument is that evolution requires *adaptation* (actually it can also be seen as adaptation plus selection, as discussed in the previous chapter). From an evolutionary perspective, adaptation is the reconstruction or prediction of evolutionary events by assuming that all characters are established by direct natural selection of the most adapted state, i.e. the state that is an 'optimum solution' to a 'problem' posed by the environment. Another definition is that under adaptation, organisms can be viewed as complex adaptive systems whose parts have (adaptive) functions subsidiary to the fitness-promoting function of the whole.

The key issue to be kept in mind here is that evolution can be viewed as an algorithmic process that allows - via reproduction with inheritance, variation and natural selection - the most adapted organisms to survive and be driven to a state of high adaptability (optimality) to their environment. These are the inspiring principles of evolutionary algorithms; the possibility of modeling evolution as a search process capable of producing individuals (candidate solutions to a problem) with increasingly better 'performances' in their environments.

3.4.3. Basic Principles of Genetics

The theory of evolution used in the development of most evolutionary algorithms is based on the three main aspects raised by Darwin as being responsible

for the evolution of species: reproduction with inheritance, variation, and selection. However, the origins of heredity along with variations, which were some of the main ingredients for the natural selection theory, were unknown at that time. This section explores the genetic basis of reproduction and variation in order to provide the reader with the necessary biological background to develop and understand evolutionary algorithms, in particular *genetic algorithms*. The union of genetics with some notions of the selection mechanisms, together with Darwin's hypotheses led to what is currently known as neo-Darwinism.

Gregor Mendel's paper establishing the foundations of *genetics* (a missing bit for a broader understanding of the theory of evolution) was published only in 1865 (Mendel, 1865), but it was publicly ignored until about the 1900. He performed a series of careful breeding experiments with garden peas. In summary, Mendel selected strains of peas that differed in particular *traits* (characteristics). As these differences were clearly distinguishable, their *phenotypes* (measurable attributes, or observable physical or biochemical characteristics of an organism) were identified and scored. For instance, the pea seeds were either smooth or wrinkled, the pod shape was either inflated or constricted, and the seed color was either yellow or green. Then, Mendel methodically performed crosses among the many pea plants, counted the progeny, and interpreted the results. From this kind of data, Mendel concluded that phenotypic traits were controlled by *factors*, later called *Mendelian factors*, and now called *genes*. *Genotype* is the term currently used to describe the genetic makeup of a cell or organism, as distinguished from its physical or biochemical characteristics (the phenotype). Figure 3.4 summarizes the first experiment performed by Mendel.

The basic structural element of all organisms is the *cell*. Those organisms whose genetic material is located in the *nucleus* (a discrete structure within the cell that is bounded by a nuclear membrane) of the cells are named *eukaryotes*. *Prokaryotes* are the organisms that do not possess a nuclear membrane surrounding their genetic material. The description presented here focuses on eukaryotic organisms.

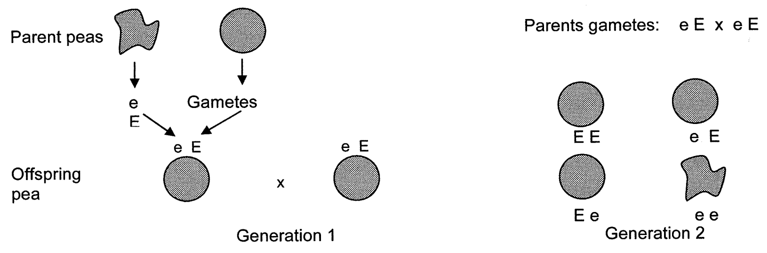


Figure 3.4: First experiment of Mendel. When crossing a normal pea with a wrinkled pea, a normal pea was generated (generation 1). By crossing two daughters from generation 1, three normal peas were generated plus one wrinkled pea. Thus, there is a recessive gene (e) that only manifests itself when there is no dominant gene together. Furthermore, there is a genetic inheritance from parents to offspring; those offspring that carry a factor that expresses a certain characteristic may have offspring with this characteristic.

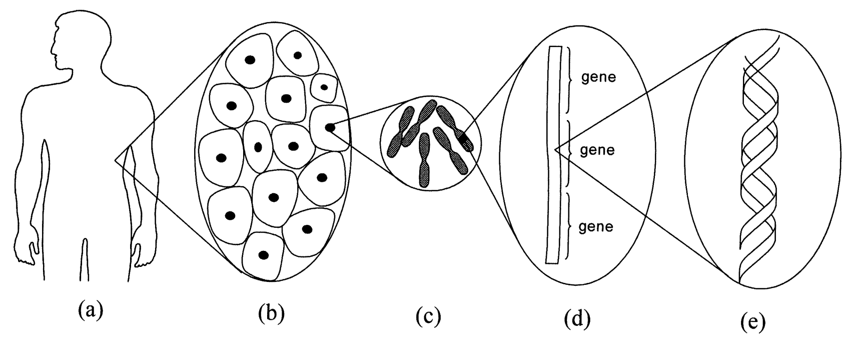


Figure 3.5: Enlargement of an organism to focus the genetic material. (a) Human organism. (b) Cells composing the organism. (c) Each cell nucleus contains chromosomes. (d) Each chromosome is composed of a long DNA segment, and the genes are the functional portions of DNA. (e) The double helix of DNA. (Modified with permission from [Griffiths et al., 1996], © W. H. Freeman and Company.)

In the cell nucleus, the genetic material is complexed with protein and is organized into a number of linear structures called *chromosomes*, which means, 'colored body', and is so named because these threadlike structures are visible under the light microscope only after they are stained with dyes. A *gene* is a segment of a helix molecule called *deoxyribonucleic acid*, or *DNA* for short. Each eukaryotic chromosome has a single molecule of DNA going from one end to the other. Each cell nucleus contains one or two sets of the basic DNA complement, called *genome*. The genome itself is made of one or more chromosomes. The *genes* are the functional regions of DNA. Figure 3.5 depicts a series of enlargements of an organism to focus on the genetic material.

It is now known that the DNA is the basis for all processes and structures of life. The DNA molecule has a structure that contributes to the two most fundamental properties of life: reproduction and development. DNA is a double helix structure with the inherent feature of being capable of replicating itself before the cell multiplication, allowing the chromosomes to duplicate into *chromatids*,

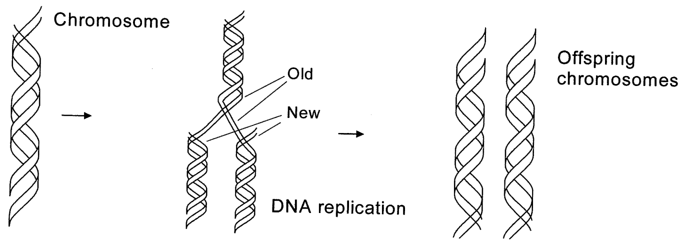


Figure 3.6: When new cells are formed, the DNA replication allows a chromosome to have a pair of offspring chromosomes and be passed onto the offspring cells.

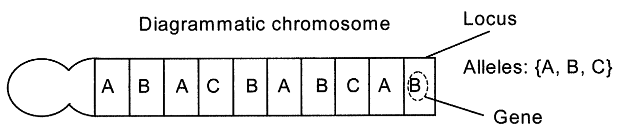


Figure 3.7: Diagrammatic chromosome depicting the locus, genes and three alleles {A, B, C}.

which eventually become *offspring chromosomes* that are transmitted to the *offspring cells*. The DNA replication process is essentially the same for sexual and asexual reproduction and is depicted in Figure 3.6.

It is the DNA replication property that allows the replication of cells and organisms during generations. Therefore, the DNA can be viewed as the 'string' that connects any organism to its descendants (Griffiths et al., 1996).

Genetics is a science named after its main object of study, namely, the genes. Studies in genetics demonstrated that many differences among organisms are results of differences in the genes they carry. Therefore, a gene can also be defined as the genetic factor that controls one trait or one characteristic of the organism. Together with the environmental influences, the genotype determines the phenotype of an organism. The different forms of a gene that determine alternate traits or characteristics are called *alleles*. The specific place on a chromosome where a gene is located is termed *locus*. Figure 3.7 presents a diagrammatic chromosome depicting the locus, genes, and alleles.

The genetic material of eukaryotes is distributed among multiple chromosomes, whose number usually varies according to the characteristics of the species. Many eukaryotes have two copies of each type of chromosome in their nuclei, so their chromosome complement is said to be *diploid*. In diploids, the members of a chromosome pair are called *homologous chromosomes*. Diploid eukaryotes are produced by the fusion of two *gametes* (mature reproductive cell specialized for sexual fusion), one from the female parent and another from the male parent. The fusion produces a diploid cell called *zygote*, which then undergoes embryological development. Each gamete has only one set of chromosomes and is said to be *haploid*.

Eukaryotes can reproduce by asexual or sexual reproduction. In *asexual reproduction*, a new individual develops from either a single cell or from a group of cells in the absence of any sexual process. It is found in multicellular and unicellular organisms. Single-celled eukaryotes grow, double their genetic material, and generate two progeny cells, each of which contains an exact copy (sometimes subjected to a small variation) of the genetic material found in the parent cell. The process of asexual reproduction in haploids is illustrated in Figure 3.8.

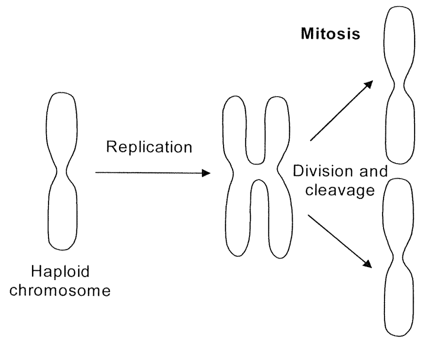


Figure 3.8: Asexual reproduction in haploids. The chromosome replicates itself, the cell nucleus is divided through a process named mitosis, and then the cell is divided into two identical progeny.

Sexual reproduction is the fusion of two *haploid gametes* (sex cells) to produce a single *diploid zygote* cell. An important aspect of sexual reproduction is that it involves *genetic recombination*; that is, it generates gene combinations in the offspring that are distinct from those in the parents. Sexually reproducing organisms have two sorts of cells: *somatic* (body) cells, and *germ* (sex) cells. All somatic cells reproduce by a process called *mitosis* that is a process of nuclear division followed by cell division. Figure 3.9 illustrates the process of sexual reproduction.

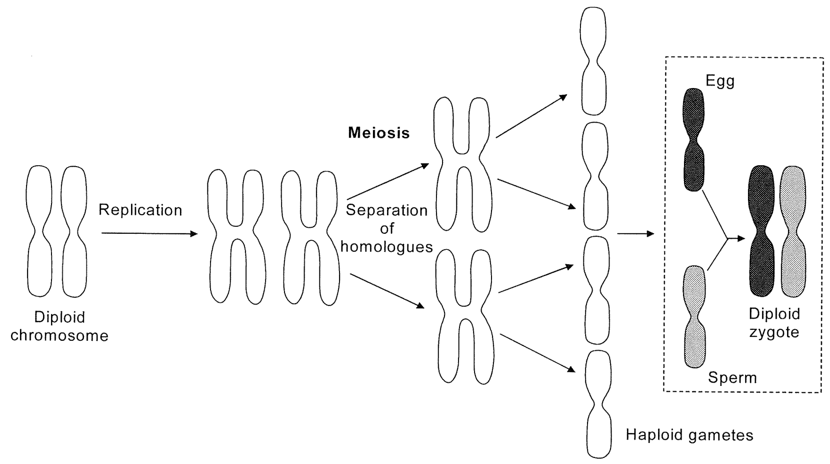


Figure 3.9: Sexual reproduction. A diploid chromosome replicates itself, then the homologues are separated generating haploid gametes. The gametes from each parent are fused to generate a diploid zygote.

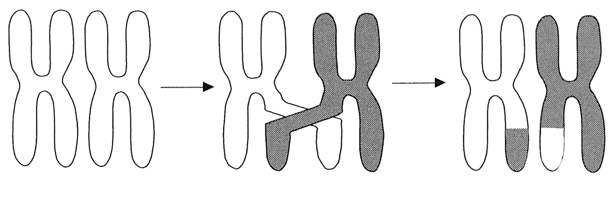


Figure 3.10: Crossing over between two loci in a cell undergoing the first meiotic division. Of the four chromatids, two will have new combinations and two will retain the parental combination of alleles.

In the classical view of the meiosis process in sexual reproduction, homologous chromosomes first undergo the formation of a very tight association of homologues, and then the reciprocal physical exchange of chromosome segments at corresponding positions along pairs of homologous chromosomes, a process termed *crossover* (Russel, 1996). Crossing-over is a mechanism that can give rise to *genetic recombination*, a process by which parents with different genetic characters give birth to progeny so that genes are associated in new combinations. Figure 3.10 depicts the crossing-over process.

The differences among organisms are outcomes of the evolutionary processes of *mutation* (a change or deviation in the genetic material), *recombination* or *crossover* (exchange of genetic material between *chromosomes*; see Figure 3.10), and *selection* (the favoring of particular combinations of genes in a given environment). With the exception of gametes, most cells of the same eukaryotic organism characteristically have the same number of chromosomes. Further, the organization and number of genes on the chromosomes of an organism are the same from cell to cell. These characteristics of chromosome number and gene organization are the same for all members of the same species. *Deviations* are known as *mutations*; these can arise spontaneously or be induced by chemical or radiation mutagens. Several types of mutation exist, for instance point mutation, deletion, translocation, and inversion. Point mutation, deletion and inversion are illustrated in Figure 3.11.

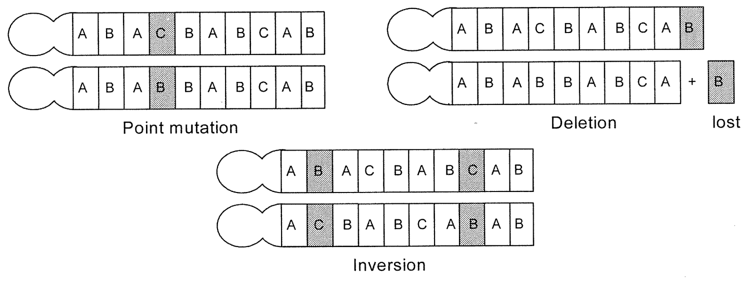


Figure 3.11: Some types of chromosomal mutation, namely, point mutation, deletion, and inversion.

3.4.4. Evolution as an Outcome of Genetic Variation Plus Selection

So far we have seen the two types of reproduction, sexual and asexual, and some of the main mechanisms that alter the genetic makeup of a population of individuals, emphasizing crossover and mutation. It still remains to discuss the process by which these altered individuals survive over the generations.

Populations of individuals change over time. The number of individuals may increase or decrease, depending on food resources, climate, weather, availability of breeding areas, predators, and so forth. At the genetic level, a population may change due to a number of factors, such as mutation and selection. These processes not only alter allele frequencies, but also result in changes in the adaptation and diversity of populations, thus leading to the evolution of a species (Gardner et al., 1991).

The viability and fertility of an individual are associated with *fitness*, a term that is used to describe the overall ability of an organism to survive and reproduce. In many populations, survival and reproductive ability are variable traits. Some individuals die before they have a chance to reproduce, whereas others leave many progeny. In a population of stable size, the average number of offspring produced by an individual is one.

Variation in fitness is partially explained by the underlying genetic differences of individuals. The crossing-over of parental genetic material and mutation can increase or decrease fitness, depending on their effects on the survival and reproductive capabilities of the individuals. Thus, genetic recombination and mutation can create phenotypes with different fitness values. Among these, the most fit will leave the largest number of offspring. This differential contribution of progeny implies that alleles associated with superior fitness will increase in frequency in the population. When this happens, the population is said to be undergoing *selection*.

As Darwin made a series of observations of domestic animals and plants, and also those existing free in nature, he used the term natural selection to describe the latter in contrast to men's selection capabilities of domestic breeds. To our purposes, the more general term *selection* is assumed in all cases, bearing in mind that selection under nature has been originally termed natural selection, and selection made by men has been sometimes termed artificial selection.

Under the evolutionary biology perspective, *adaptation* is the process by which traits evolve making organisms more suited to their immediate environment; these traits increase the organisms' chances of survival and reproduction. Adaptation is thus responsible for the many extraordinary traits seen in nature, such as eyes that allow us to see, and the sonar in bats that allow their guidance through the darkness. Note however, that, more accurately speaking, adaptation is a result of the action of both, variation and selection. Variation by itself does not result in adaptation; there must be a way (i.e., selection) of promoting the maintenance of those advantageous variations.

S. Wright (1968-1978) introduced the concept of *adaptive landscapes* or *fitness landscapes*, largely used in evolutionary biology. In his model, each population of a *species* (reproductively isolated group) is symbolized by a point on a *topographic map*, or *landscape*. The contours of the map represent different levels of adaptation to the environment (fitness). Populations at high levels (peaks) are more adapted to the environment, and populations at low levels (valleys) are less adapted. At any one time, the position of a population will depend on its genetic makeup. Populations with alleles that improve fitness will be at a higher peak than populations without these alleles. Consequently, as the genetic makeup of a population changes, so will its position on the adaptive landscape. Figure 3.12 depicts a landscape representing the different levels of adaptation of the populations in relation to the environment.

The adaptive (fitness) landscape corresponds to the response surface discussed in Section 3.2.1 in the context of problem solving via search in a search space. Note that, under the evolutionary perspective, the search performed is for individuals with increased survival and reproductive capabilities (fitness) in a given environment (fitness landscape).

A *niche* can thus be defined as the region consisting of the set of possible environments in which a species can persist; members of one species occupy the same ecological niche. In natural ecosystems, there are many different ways in which animals may survive (grazing, hunting, on water, etc.), and each survival strategy is called an ecological niche. However, it is generally recognized that the niche of a single species may vary widely over its geographical range. The other fundamental concept of niche was proposed by Elton (1927) "The niche of an animal means its place in the biotic environment, its relations to food and enemies;" where the term *biotic* refers to life, living organisms. Thus, niche in this case is being used to describe the role of an animal in its community (Krebs, 1994).

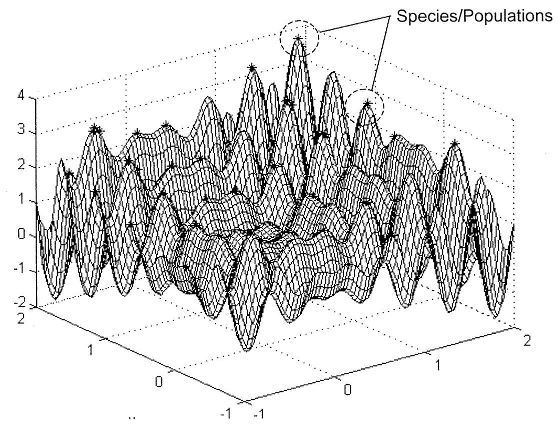


Figure 3.12: An example of an adaptive landscape. The topographic map (landscape or surface) corresponds to the different levels of adaptation of the populations (points in the landscape) to the environment. The populations or individuals at each peak are assumed to be reproductively isolated, i.e., they only breed with individuals in the same peak, thus forming species inhabiting distinct niches.

It has been discussed that evolution is an outcome of genetic variation plus selection. In order for continuing evolution to occur, there must be mechanisms that increase or create genetic variation and mechanisms that decrease it. We have also seen that recombination and mutation cause differences among (variations in) organisms. Other two important mechanisms of evolution are the so-called *genetic drift* (chance fluctuations that result in changes in allele frequencies) and *gene flow* (spread of genes among populations via *migration*). It is known that selection and genetic drift decrease variation, while mutation, recombination, and gene flow increase genetic variation (Colby, 1997).

Natural selection sifts through the genetic variations in the population, preserving the beneficial ones and eliminating the harmful ones. As it does this, selection tends to drive the population uphill in the adaptive or fitness landscape. By contrast, the random genetic drift will move the population in an unpredictable fashion. The effect of all these mechanisms (mutation, recombination, genetic drift, gene flow, plus selection) will bring the population to a state of 'genetic' equilibrium, corresponding to a point near or at a peak on the adaptive landscape. Actually, the population will hover around a peak because of fluctuations caused by genetic drift. Note also, that, under nature, the environment is constantly changing, hence the population is also adapting to the new landscape resultant from the new environment, in a never-ending process of variation and selection.

3.4.5. A Classic Example of Evolution

A classic example of evolution comes from species that live in disturbed habitats. In the particular example of the evolution of melanic (dark) forms of moths, human activity has altered the environment and there has been a corresponding change in the species that inhabit this environment. The peppered moth, *Biston betularia*, is found in wooded areas in Great Britain, where it exists in two color forms, light and dark; light being the typical phenotype of this species. The difference between the two forms is believed to involve a single gene. Since 1850, the frequency of the dark form has increased in certain areas in England, in particular in industrialized parts of the country. Around heavily industrialized cities, such as Manchester and Birmingham, the frequency of the dark form has increased drastically from 1 to 90% in less than 100 years. In other areas of England, where there is little industrial activity, the dark form has remained very rare (Gardner et al., 1991).

The rapid spread of the dark form in industrialized areas has been attributed to natural selection. Both, light and dark forms are active at night. During the day, the moths remain still, resting on tree trunks and other objects in the woodlands. Since birds may find the moths and eat them during their resting period, camouflage is their only defense against predation. On white or gray tree bark, the light moths are protectively colored, especially if the bark is overgrown with lichens. However, in industrialized areas most of the lichens have been killed by pollution and the tree bark is oftenest darkened by soot. Such conditions offer little or no cover for the light moths, but make ideal resting spots for the dark

ones. Predatory birds have difficulties in seeing the dark moths on the darkened barks, similarly to what happens with light moths on light barks. The spread of the dark moths in industrialized areas thus appears to be a result of its selective advantage on a sooty background. Because the dark moth survived more in polluted woods, the gene responsible for the dark color increased in frequency over industrialized regions.

3.4.6. A Summary of Evolutionary Biology

The three basic principles of Darwin's theory of evolution, together with the genetics introduced by Mendel and some ideas about the natural selection process, form what is currently known as neo-Darwinism; for Darwin had no knowledge of genetics, a missing bit of his theory.

We have seen that evolution is a result of a reproducing population(s) of individuals, which suffer genetic variation followed by selection. The variation affects the genetic makeup of individuals (genotype), which will present a selective advantage over the others if their phenotypes confer them a better adaptability to the environment they inhabit. This degree of adaptability to the environment (capability of surviving and reproducing) is broadly termed fitness.

The genetic entities that suffer variation are located within the cell nucleus and are named chromosomes, whose basic functional units are the genes. In both types of reproduction, asexual and sexual, there may occur a deviation (mutation) in one or more alleles of a chromosome, allowing the appearance of a new character in the phenotype (physical and chemical characteristics) of the offspring individual. In sexual reproduction, in addition to mutation, there is the recombination (crossing-over) of parental genetic material, resulting in offspring that present features in common with both parents.

The survival and reproductive advantage of an individual organism, its fitness, endows it with a selective advantage over the others, less fit individuals. Those individuals whose genetic makeup result in a phenotype more adapted to the environment have higher probabilities of surviving and propagating their genotypes. It can thus be seen that there is a competition for survival among the many offspring generated. As argued by Darwin

> "Every being, which during its natural lifetime produces several eggs or seeds, must suffer destruction during some period of its life, and during some season or occasional year, otherwise, on the principle of geometrical increase, its numbers would quickly become so inordinately great that no country could support the product. Hence, as more individuals are produced than can possibly survive, there must in every case be a struggle for existence, either one individual with another of the same species, or with the individuals of distinct species, or with the physical conditions of life." (Darwin, 1859; p. 50)

An adaptive landscape (or surface) is a topographic map used to represent the degree of adaptation of individuals in a given environment. Individuals that only reproduce with each other are part of the same species, which occupies one or more biological niche. As the fittest individuals of the population have higher chances of surviving and reproducing, the outcome of evolution is a population increasingly more fit to its environment.

Viewed in this manner, evolution is clearly a search and optimization problem solving process (Mayr, 1988). Selection drives phenotypes as close as possible to the optimum of a fitness landscape, given initial conditions and environmental constraints. Evolution thus, allows the discovery of functional solutions to particular problems posed by the environment in which some organisms live (e.g., a more appropriate color for moths living in a sooty area). This is the perspective assumed when using evolutionary biology as a source of inspiration for the development of evolutionary computation. In such a case, an evaluation function will define a *fitness landscape* for the problem, and finding the best solution to the problem (most adapted population or individual in a given environment) will correspond to the search for a peak (assuming a maximization problem) of this landscape by subjecting individuals to genetic variation (e.g., crossover and mutation) operators.

3.5 EVOLUTIONARY COMPUTING

Evolution can be viewed as a search process capable of locating solutions to problems offered by an environment. Therefore, it is quite natural to look for an algorithmic description of evolution that can be used for problem solving. Such an algorithmic view has been discussed even in philosophy (Section 3.4.2). Those iterative (search and optimization) algorithms developed with the inspiration of the biological process of evolution are termed *evolutionary algorithms* (EAs). They are aimed basically at problem solving and can be applied to a wide range of domains, from planning to control. *Evolutionary computation* (EC) is the name used to describe the field of research that embraces all evolutionary algorithms.

The basic idea of the field of evolutionary computation, which came onto the scene about the 1950s to 1960s, has been to make use of the powerful process of natural evolution as a problem-solving paradigm, usually by simulating it on a computer. The original three mainstreams of EC are *genetic algorithms* (GAs), *evolution strategies* (ES), and *evolutionary programming* (EP) (Bäck et al., 2000a,b). Another mainstream of evolutionary computation that has been receiving increasingly more attention is *genetic programming* (Koza, 1992; Koza, 1994a). Despite some differences among these approaches, all of them present the basic features of an evolutionary process as proposed by the Darwinian theory of evolution.

3.5.1. Standard Evolutionary Algorithm

A standard evolutionary algorithm can be proposed as follows:

- *A population of individuals that reproduce with inheritance*. Each individual represents or encodes a point in a search space of potential solutions to a problem. These individuals are allowed to reproduce (sexually or asexually), generating offspring that inherit some features (traits) from their parents. These inherited traits cause the offspring to present resemblance with their progenitors.
- *Genetic variation*. Offspring are prone to genetic variation through mutation, which alters their genetic makeup. Mutation allows the appearance of new traits in the offspring and, thus, the exploration of new regions of the search space.
- *Natural selection*. The evaluation of individuals in their environment results in a measure of adaptability, quality, or fitness value to be assigned to them. A comparison of individual fitnesses will lead to a competition for survival and reproduction in the environment, and there will be a selective advantage for those individuals of higher fitness.

The standard evolutionary algorithm is a generic, iterative, and probabilistic algorithm that maintains a population **P** of N individuals, $\mathbf{P} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, at each iteration t (for simplicity of notation the iteration index t is suppressed). Each individual corresponds to (represents or encodes) a potential solution to a problem that has to be solved. An individual is represented using a *data structure*. The individuals $\mathbf{x}_i$, $i = 1, \dots ,N$, are evaluated to give their measures of adaptability to the environment, or *fitness*. Then, a new population, at iteration $t + 1$, is generated by *selecting* some (usually the most fit) individuals from the current population and *reproducing* them, sexually or asexually. If employing sexual reproduction, a *genetic recombination* (*crossover*) operator may be used. Genetic variations through *mutation* may also affect some individuals of the population, and the process iterates. The completion of all these steps: reproduction, genetic variation, and selection, constitutes what is called a *generation*. An initialization procedure is used to generate the initial population of individuals. Two parameters pc and pm correspond to the genetic recombination and variation probabilities, as will be further discussed.

Algorithm 3.6 depicts the basic structure of a standard evolutionary algorithm. Most evolutionary algorithms can be implemented using this standard algorithm, with some differences lying on the representation, selection, reproduction, variation operators, and in the order these processes are applied. The stopping criterion is usually a maximum number of generations, or the achievement of a prespecified objective.

Note that all evolutionary algorithms involve the basic concepts common to every algorithmic approach to problem solving discussed in Section 3.1: 1) representation (data structures), 2) definition of an objective, and 3) specification of an evaluation function (fitness function). Although the objective depends on the problem to be solved, the representation and the evaluation function may depend on the designers' experience or expertise.

```
procedure [P] = standard_EA(pc,pm)
  initialize P
  f ← eval(P)
  P ← select(P,f)
  t ← 1
  while not_stopping_criterion do,
      P ← reproduce(P,f,pc)
      P ← variate(P,pm)
      f ← eval(P)
      P ← select(P,f)
      t ← t + 1
  end while
end procedure
```

Algorithm 3.6: A standard evolutionary algorithm.

3.5.2. Genetic Algorithms

All evolutionary algorithms embody the basic processes of the Darwinian theory of evolution, as described in Algorithm 3.6. However, the *genetic algorithms* (GAs) are those that use a vocabulary borrowed from natural genetics. This method has been offered by a number of researchers (e.g., Fraser, 1959; Friedberg, 1958; Anderson, 1953; Bremermann, 1962), but its most popular version was presented by J. Holland (1975). This is the one to be introduced here.

The data structures representing the *individuals* (*genotypes*) of the population are often called *chromosomes*; these are one-chromosome individuals, i.e., *haploid* chromosomes. In standard genetic algorithms the individuals are represented as strings of binary digits {0,1}, or *bitstrings*. In accordance with its biological source of inspiration (i.e. genetics) each unit of a chromosome is a *gene*, located in a certain place in the chromosome called *locus*. The different forms a gene can assume are the *alleles*. Figure 3.13 illustrates a bitstring of length $l = 10$ corresponding to a chromosome in a genetic algorithm. Note the similarity of this representation with the one used for the diagrammatic chromosome of Figure 3.7.

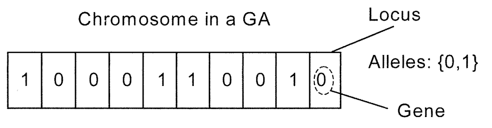


Figure 3.13: Bitstring of length $l = 10$ representing a chromosome in a standard genetic algorithm. Each position (locus) in the chromosome can assume one of the two possible alleles, 0 or 1.

As discussed in Section 3.1, the problem to be solved is defined and captured in an *objective function* that allows to evaluate the *fitness* (procedure `eval` of Algorithm 3.6) of any potential solution. Each genotype, in this case a single chromosome, represents a potential solution to a problem. The meaning of a particular chromosome, its *phenotype*, is defined according to the problem under study. A genetic algorithm run on a population of chromosomes (bitstrings) corresponds to a search through a space of potential solutions.

As each chromosome $\mathbf{x}_i$, i, … , N, often corresponds to the encoded value of a candidate solution, it often has to be decoded into a form appropriate for evaluation and is then assigned a fitness value according to the objective. Each chromosome is assigned a probability of reproduction, p_i, i, … , N, so that its likelihood of being selected is proportional to its fitness relative to the other chromosomes in the population; the higher the fitness, the higher the probability of reproduction, and vice-versa. If the fitness of each chromosome is a strictly positive number to be maximized, selection (procedure `select` of Algorithm 3.6) is traditionally performed via an algorithm called *fitness proportional selection* or *Roulette Wheel selection* (Fogel, 2000a).

The assigned probabilities of reproduction result in the generation of a population of chromosomes probabilistically selected from the current population. The selected chromosomes will generate offspring via the use of specific *genetic operators*, such as *crossover*, and *bit mutation* might be used to introduce genetic variation in the individuals. Crossover is the genetic recombination operator embodied in the procedure `reproduce` of Algorithm 3.6. Mutation is the genetic variation mechanism of procedure `variate` of Algorithm 3.6. The iterative procedure stops when a fixed number of iterations (generations) has been performed, or a suitable solution has been found.

Roulette Wheel Selection

In the *fitness proportional* or *Roulette Wheel* (RW) selection method, the probability of selecting a chromosome (individual of the population) is directly proportional to its fitness value. Each chromosome $\mathbf{x}_i$, $i = 1, \ldots, N$, of the population is assigned to a part of the wheel whose size is proportional to its fitness value. The wheel is then tossed as many times as parents (N) are needed to create the next generation, and each winning individual is selected and copied into the parent population. Note that this method allows an individual to be selected more than once and the 'death' of some individuals as well.

Figure 3.14 depicts the RW applied to a population composed of four individuals. To play the roulette might correspond to obtaining a value from a random number generator with uniform distribution in the interval [0,1] (see Section 3.5.3 for another form of implementing the RW selection). The value obtained is going to define the chromosome to be chosen, as depicted in Figure 3.14. The higher the fitness of an individual, the larger the portion of the roulette to be assigned to this individual, thus the higher its probability of being selected.

Ind.	Chromosome	Fitness	Degrees
1	0001100101010	16	240
2	0101001010101	4	60
3	1011110100101	2	30
4	1010010101001	2	30

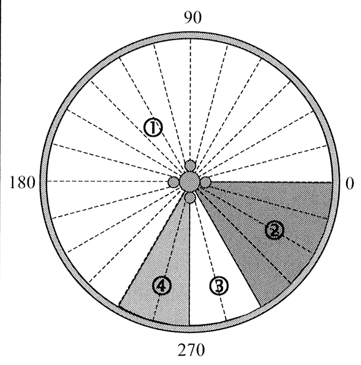


Figure 3.14: Roulette Wheel selection. (reproduced with permission from [de Castro and Timmis, 2002]. © L. N. de Castro and J. Timmis, 2002).

Crossover

In biological systems, crossing-over is a process that yields the recombination of alleles via the exchange of segments between pairs of chromosomes (see Figure 3.10). As discussed in Section 3.4.3, it occurs only in sexually reproducing species. This process can be abstracted as a general operator to the level of the data structures discussed, i.e., bitstrings. Crossing-over proceeds basically in three steps (Holland, 1975):

- Two strings $\mathbf{x} = x_1x_2 \ldots x_l$ and $\mathbf{y} = y_1y_2 \ldots y_l$ are selected from the current population **P**.
- A number r indicating the crossover point is randomly selected from $\{1, 2, \ldots, l-1\}$.
- Two new strings are formed from **x** and **y** by exchanging the set of attributes to the right of position r, yielding $\mathbf{x}' = x_1 \ldots x_i y_{i+1} \ldots y_l$ and $\mathbf{y}' = y_1 \ldots y_i x_{i+1} \ldots x_l$.

The two new chromosomes (strings), **x**' and **y**', are the offspring of **x** and **y**. This single-point crossover is illustrated in Figure 3.15 for two strings of length $l = 8$.

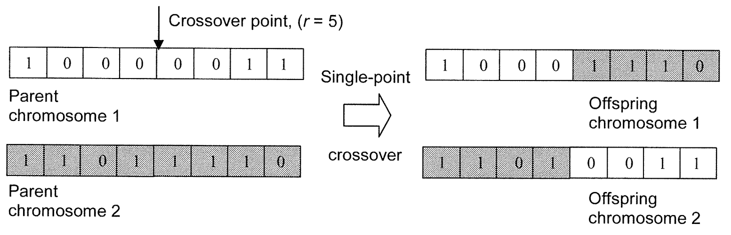


Figure 3.15: Single-point crossover for a pair of chromosomes of length $l = 8$.

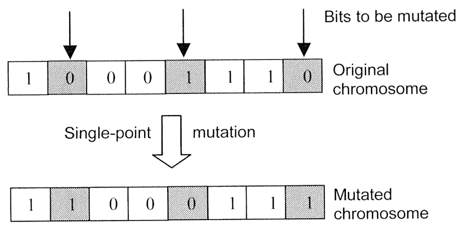


Figure 3.16: Point mutations for a chromosome of length $l = 8$ (three points are selected for being mutated).

Mutation

In genetics (Section 3.4.3), point mutation is a process wherein one allele of a gene is randomly replaced by (or modified to) another to yield a new chromosome (structure). Generally, there is a small probability of mutation at each gene in the structure (in the GA context, a bitstring). It means that, each bitstring in the population P is operated upon as follows (Holland, 1975):

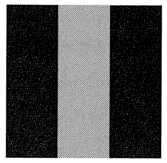
- The numbers $r,\ldots,u$ indicating the positions to undergo mutation are determined by a random process where each position has a small probability pm of undergoing mutation, independently of the other positions.
- A new string $\mathbf{x}' = x_1 \ldots x_r \ldots x_u \ldots x_l$ is generated where $x_r \ldots x_u$ are drawn at random from the set of alleles for each gene. In the case of bitstrings, if a position has an allele '0', then it becomes '1', else if it is originally '1', then it becomes '0'.

The parameter pm is the mutation probability at each position. Figure 3.16 illustrates the process of point or bit mutation.

3.5.3. Examples of Application

Algorithm 3.6 shows the standard evolutionary algorithm, which is the same as a standard genetic algorithm, and follows the Darwinian theory of evolution. In the standard GA, selection - `select` - is performed via Roulette Wheel, reproduction - `reproduce` - is performed by crossing-over pairs of the selected individuals, and genetic variation - `variate` - is performed via mutation. Fitness evaluation - `eval` - depends on the problem under study.

To illustrate how to put all these procedures together in practical problems, consider the following examples. The first example - pattern recognition - is aimed at illustrating the basic steps and implementation of a standard genetic algorithm. In this particular case, the GA capability of learning a known pattern will be tested. The second example - numeric function optimization - the GA potential of determining the (unknown) global optimal (maximum) to a numeric function will be assessed. The focus of this latter example is in the encoding (representation) scheme used; the basic selection and genetic operators are the same as those used in the first example.



(a)

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$4 \times 3 = 12$

(b)

Figure 3.17: Character to be recognized using a standard genetic algorithm. (a) Pictorial view. (b) Matrix representation.

A Step by Step Example: Pattern Recognition (Learning)

As a first example of how the standard GA works, consider the problem of evolving a population of individuals toward recognizing a single character; the number '1' depicted in Figure 3.17(a). The resolution of this picture is *rows* = 4 and *columns* = 3. This character is drawn by using two colors, light and dark gray, and can be represented by a matrix of '0s' and '1s', where each '0' corresponds to the dark gray, while each '1' corresponds to the light gray (Figure 3.17(b)).

In the genetic algorithm a small population **P** with only $N = 8$ individual chromosomes will be chosen to be evolved toward recognizing the character '1'. This population **P** will correspond to a matrix with $N = 8$ rows (the individuals) of $l = 12$ columns. To generate a single individual we simply place one row of the character after the other downward. For example, the target individual of Figure 3.17 is represented by the vector $\mathbf{x}_1 = [010010010010]$. (Compare Figure 3.17(b) with $\mathbf{x}_1$.)

The standard evolutionary algorithm presented in Algorithm 3.6 also corresponds to the standard genetic algorithm. What differs the standard GA from the other evolutionary algorithms (*evolution strategies* and *evolutionary programming*) is basically the data representation, the operators, the selection method, and the order in which these operators are applied; the basic idea remains. The standard GA has the following main features:

- Binary representation of the data structures.
- Fixed population size.
- Single-point crossover.
- Point mutation.
- Roulette Wheel selection.

Initialization

Figure 3.18 depicts a randomly initialized population **P** to be evolved by Algorithm 3.6. The matrix **P** corresponding to the individuals of Figure 3.18 is:

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & \underline{1} \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \overset{8}{\leftarrow} \text{individual}$$

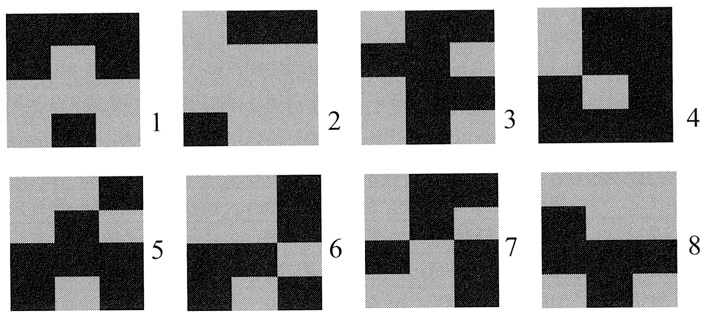


Figure 3.18: Randomly initialized population P for the standard GA.

Fitness Evaluation

After initializing the population of candidate solutions, the next step in the algorithm (Algorithm 3.6) is to evaluate the quality of each individual in relation to the problem to be solved, i.e., to calculate the fitness of each individual. As the goal is to generate individuals that are as similar as possible to the target individual (Figure 3.17), a straightforward way of determining fitness is by counting the number of similar bits between each individual of the population and the target individual. The number of different bits between two bitstrings is termed *Hamming distance*. For instance, the vector **h** of Hamming distances between the individuals of **P** and the target individual is $\mathbf{h} = [6,7,9,5,5,4,6,7]$.

The fitness of the population can be measured by subtracting the length $l = 12$ of each individual by its respective Hamming distance to the target individual. Therefore, the vector of fitness becomes $\mathbf{f} = [f_1, \ldots, f_8] = [6,5,3,7,7,8,6,5]$.

The ideal individual is the one whose fitness is $f = 12$. Therefore, the aim of the search to be performed by the GA is to maximize the fitness of each individual, until (at least) one individual of **P** has fitness $f = 12$.

Roulette Wheel Selection

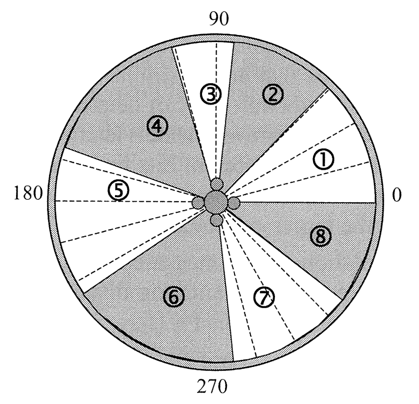
To perform selection via Roulette Wheel, each individual will be assigned a portion of the Roulette proportional to its fitness; the higher the fitness, the higher the slice of the roulette, and vice-versa. Roulette Wheel can be implemented as follows:

- Sum up the fitness of all individuals, $f_T = \Sigma_i f_i$, $i = 1, \dots, N$.
- Multiply the fitness of each individual by 360 and divide it by the total fitness: $f_i' = (360.\, f_i)/f_T$, $i = 1, \dots, N$, determining the portion of the wheel to be assigned for each individual.
- Sort a random number in the interval (0,360] and compare it with the interval belonging to each individual. Select the individual whose interval contains the sorted number and place it in the new population **P**.

The whole process is illustrated in Figure 3.19, including the size of each portion and its interval (Portion). (Note that the Roulette Wheel can also be implemented within the interval [0,1] instead of (0,360], as discussed previously.)

Individual	Chromosome	Fitness	Degree	Portion of the Roulette
1	000010111101	6	46	(0,46]
2	100111111011	5	38	(46,84]
3	100001100101	3	23	(84,107]
4	100100010000	7	54	(107,161]
5	110101000010	7	54	(161,215]
6	110110001010	8	61	(215,276]
7	100101010110	6	46	(276,322]
8	111011000101	5	38	(322,360]

(a)



(b)

Figure 3.19: Roulette Wheel selection. (a) Descriptive table. (b) Roulette Wheel. (*Note: the degrees were rounded for didactic purposes.*)

Assuming that **s** = [230,46,175,325,275,300,74,108] were the sorted random numbers, the population **P** will be composed of the individuals **P** = [$\mathbf{x}_6$, $\mathbf{x}_1$, $\mathbf{x}_5$, $\mathbf{x}_8$, $\mathbf{x}_6$, $\mathbf{x}_7$, $\mathbf{x}_2$, $\mathbf{x}_4$]^T, where ^{T} corresponds to the matrix transposition operator (Appendix B.1.4).

It can be noticed that the less fit individual of the population ($\mathbf{x}_3$) went extinct, while the most fit individual ($\mathbf{x}_6$) was selected twice. This can be interpreted as individual $\mathbf{x}_6$ having had a higher probability of leaving progenies, while $\mathbf{x}_3$ having left no progeny at all.

Reproduction

In the standard GA, reproduction is accomplished by selecting pairs of individuals and performing the crossing-over of their genetic material with a crossover probability *pc*. Basically, two approaches can be taken. In the first one, both offspring replace the parents, and in the second one a single offspring replace a single parent and the process is repeated *N* times. Let us assume the first case in which both offspring will replace both parents. Thus, for the matrix **P**, *N*/2 random numbers *r* over the interval [0,1] will be chosen and if $r \leq pc$, then the crossing-over is performed according to the procedure described in Section 0; else, the original parents are repeated in the next generation.

To illustrate this procedure, let **r** = [0.5,0.7,0.3,0.9] be the four values randomly chosen for *r*, and *pc* = 0.6. There are four pairs of parents in the population **P**: $\mathbf{x}_6$ and $\mathbf{x}_1$; $\mathbf{x}_5$ and $\mathbf{x}_8$; $\mathbf{x}_6$ and $\mathbf{x}_7$; $\mathbf{x}_2$ and $\mathbf{x}_4$. In this case, the first and third pairs were selected for crossing over. Thus, $\mathbf{x}_6$ will be crossed over with $\mathbf{x}_1$, and $\mathbf{x}_6$ will again be crossed over with $\mathbf{x}_7$. What remains to be done is to define the crossover point between the strings in both cases. Assume that the randomly chosen crossover points, *cp*, from the interval [1,*l*–1] are *cp* = 5 and *cp* = 9, respectively, for each pair. Figure 3.20 illustrates the crossing-over between $\mathbf{x}_6$ and $\mathbf{x}_1$, and $\mathbf{x}_6$ and $\mathbf{x}_7$, for *cp* = 5 and *cp* = 9, respectively.

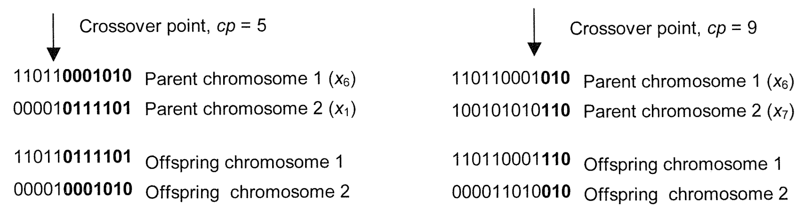

Crossover point, cp = 5
110110001010 Parent chromosome 1 (x6)
000010111101 Parent chromosome 2 (x1)
110110111101 Offspring chromosome 1
000010001010 Offspring chromosome 2
Crossover point, cp = 9
110110001010 Parent chromosome 1 (x6)
100101010110 Parent chromosome 2 (x7)
110110001110 Offspring chromosome 1
000011010010 Offspring chromosome 2

Figure 3.20: Crossing-over between x_6 and x_1, and x_6 and x_7.

The new population **P** is thus:

$$
\mathbf{P} = \begin{bmatrix}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}.
$$

Mutation (Genetic Variation)

Assume a mutation probability $pm = 0.02$. For each position in matrix **P**, a random number r is generated and compared with pm. If $r \leq pm$, then the corresponding bit is selected for being mutated. For the bits selected, as depicted in **P'** below (underlined and boldface numbers), the population **P** before and after mutation is:

$$
\mathbf{P} = \begin{bmatrix}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & \underline{\mathbf{0}} & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & \underline{\mathbf{1}} & 1 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix};
$$

$$
\mathbf{P} = \begin{bmatrix}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & \underline{\mathbf{1}} & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & \underline{\mathbf{0}} & 1 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

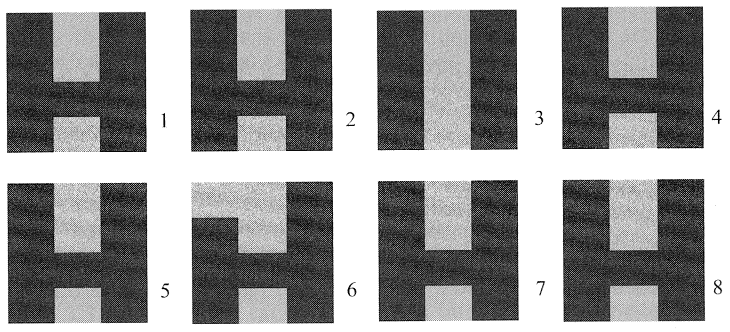
Evaluation

The last step in Algorithm 3.6 before selection is applied again is to evaluate the fitness of the population at the end of each generation. This will result in the following vector of fitnesses: $\mathbf{f} = [5,9,7,5,7,10,6,6]$, and $f_T = 55$.

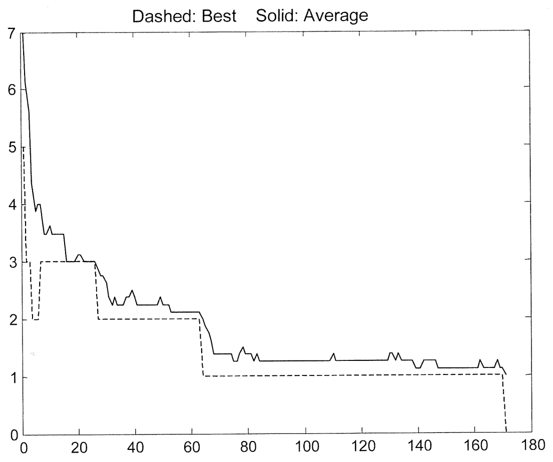
If the total fitness is divided by the number of individuals in the population, the average fitness can be obtained $f_{av} = f_T/N$. The average fitness at the previous

generation is $f_{av} = 47/8 = 5.875$, while the average fitness at the present generation is $f_{av} = 55/8 = 6.875$. If one looks at the Hamming distance instead of the fitness, it can be noticed that the average Hamming distance is being reduced towards zero. It can thus be noticed that by simply performing selection, crossover, and mutation, the population of individuals becomes increasingly more capable of recognizing the desired pattern. Note that in this particular example, minimizing the $\mathbf{h}$ is equivalent to maximizing the evaluation function chosen ($f_i = l - h_i$).

Figure 3.21(a) depicts the final population evolved by the GA after 171 generations. Figure 3.21(b) depicts the average Hamming distance, and the Hamming distance of the best individual of the population (individual $\mathbf{x}_3$) at the end of the evolutionary search process.



(a)



(b)

Figure 3.21: Final population and evolution of the population over the generations. (a) Final population. (b) Evolution of the population. Dashed line: Hamming distance of the best individual in relation to the target one. Solid line: Average Hamming distance of the whole population.

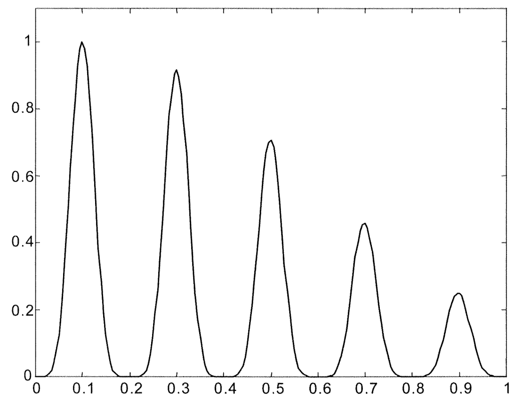


Figure 3.22: Graph of the function $g(x) = 2^{-2((x-0.1)/0.9)^2}(\sin(5\pi x))^6$ to be maximized with a standard GA.

Numerical Function Optimization

The second problem to be explored is the one of maximizing a simple unidimensional function presented in Section 3.3.3. The function is defined as

$$g(x) = 2^{-2((x-0.1)/0.9)^2}(\sin(5\pi x))^6,$$

and is depicted in Figure 3.22 (Goldberg, 1989). The problem is to find **x** from the range [0..1] which maximizes $g(\mathbf{x})$, i.e., to find $\mathbf{x}^*$ such that

$$g(\mathbf{x}^*) \geq g(\mathbf{x}),\ \forall\ \mathbf{x} \in [0..1].$$

The first aspect that has to be accounted for is the representation. How to represent the real value x using a binary string? Although it is not necessary to use a binary representation for x, it is our intent to employ the standard GA and its basic operators, as described in the previous example. Actually, some authors (e.g., Michalewicz, 1996), argue that a float-point representation is usually the most suitable one for numeric function optimization problems. Evolution strategies (Section 3.6.1) have also been very successful in solving numeric function optimization problems.

The length l of the bitstring depends on the numeric precision required. Assuming a bitstring of length l, the mapping from a binary string $\mathbf{x} = \langle x_l,\ldots, x_2, x_1\rangle$ into a real number z is completed in two steps (Michalewicz, 1996):

- Convert the binary string $\mathbf{x} = \langle x_l,\ldots, x_2, x_1\rangle$ from base 2 to base 10: $(\langle x_l,\ldots,x_2,x_1\rangle)_2 = \left(\sum_{i=0}^{l-1} x_i.2^i\right)_{10} = z'$; and

- Find the corresponding real value for z: $z = z_{\min} + z'.\frac{z_{\max} - z_{\min}}{2^l - 1}$; where $z_{\max}$ is the upper bound of the interval in which the variable is defined, $z_{\min}$ is the lower bound of this interval, and l is the string length.

To evaluate each individual of the population one has to decode the binary chromosomes **x** into their real values and then evaluate the function for these values. The remaining steps of the algorithm (selection, recombination, and mutation) are the same as the ones described for the previous example.

3.5.4. Hill-Climbing, Simulated Annealing, and Genetic Algorithms

All the algorithms discussed so far, hill-climbing (HC), simulated annealing (SAN), and genetic algorithms (GAs), have a number of versions. The focus of this text was on the standard versions of all of them. The major difference among these three approaches is that the evolutionary algorithms are population-based search techniques, while HC and SAN work with a single individual.

In the case of the standard hill-climbing search (Algorithm 3.1), its success or failure in determining the global optimum of a given function (problem) depends on its starting point. It is clear that if the starting point is not on a hill that will lead to the global optimum, the algorithm will never be able to reach this peak in a single run. Additional runs, with different starting conditions, will have to be performed. This is automatically performed in the iterated hill-climbing procedure (Algorithm 3.2). The stochastic hill-climbing algorithm (Algorithm 3.3) incorporates random variation into the search for optimal solutions, just as the simulated annealing (Algorithm 3.4) and genetic algorithm (Algorithm 3.6). In these algorithms, no two trials could be expected to take exactly the same course (sequence of steps), i.e., to do the same search in the search space.

The stochastic hill-climber, simulated annealing, and genetic algorithms are capable of escaping local optima solutions due to their stochastic nature. The simulated annealing algorithm allows the acceptance of a new point, given a Boltzmann probability, which might result in the exploration of a different region of the search space. The broader exploration of the search space in evolutionary algorithms is accomplished by the use of multiple individuals and genetic operators, which allow the creation of individuals with new features and also the sharing of features with their parents.

In the genetic algorithms, which are population-based approaches, there is the concept of competition (via a selective mechanism that privileges better – more fit – individuals) between candidate solutions to a given problem. It is also interesting to note that while hill-climbing and simulated annealing algorithms generate new candidate points in the neighborhood of the current point, evolutionary algorithms allow the examination of points in the neighborhood of two (or even more) candidate solutions via the use of genetic operators such as crossover.

To conclude this section, let us cite a passage quoted by Michalewicz (1996), which serves as a metaphor for comparing hill-climbing, simulated annealing, and genetic algorithms:

"Notice that in all [hill-climbing] methods discussed so far, the kangaroos can hope at best to find the top of a mountain close to where he starts. There's no guarantee that this mountain will be Everest, or even a very high mountain. Various methods are used to try to find the actual global optimum.

In simulated annealing, the kangaroo is drunk and hops around randomly for a long time. However, he gradually sobers up and tends to hop up hill.

In genetic algorithms, there are lots of kangaroos that are parachuted into the Himalayas (if the pilot didn't get lost) at random places. These kangaroos do not know that they are supposed to be looking for the top of Mt. Everest. However, every few years, you shoot the kangaroos at low altitudes and hope the ones that are left will be fruitful and multiply." (Sarle, 1993; quoted by Michalewicz, 1996; p. 30)

3.6 THE OTHER MAIN EVOLUTIONARY ALGORITHMS

This section provides a brief overview of the other three main types of evolutionary algorithms: evolution strategies (ES), evolutionary programming (EP) and genetic programming (GP).

3.6.1. Evolution Strategies

The *evolution strategies* (ES) are a class of evolutionary algorithms, developed by Rechenberg (1965), Schwefel (1965), and Bienert, employed mainly to solve parameter optimization problems (Schwefel, 1995; Beyer, 2001). They were initially used to tackle problems on fluid mechanics and later generalized to solve function optimization problems, focusing the case of real-valued functions. The first ESs operated with a single individual in the population subjected only to mutation.

The data structure of an ES corresponds to real-valued vectors of candidate solutions in the search space. Let us consider the most general case of an ES. An individual $\mathbf{v} = (\mathbf{x},\boldsymbol{\sigma},\boldsymbol{\theta})$ may be composed of the attribute vector **x,** and the sets of the strategy parameters $\boldsymbol{\sigma}$ and $\boldsymbol{\theta}$. The adaptation procedure is usually implemented performing first (the recombination and) the mutation - according to some probability density function - of the parameter vectors $\boldsymbol{\sigma}$ and $\boldsymbol{\theta}$, resulting in $\boldsymbol{\sigma}'$ and $\boldsymbol{\theta}'$, and then using the updated vectors of the strategy parameters to (recombine and) mutate the attribute vector **x**, resulting in **x**'. The pseudocode to implement an ES is the same as the one described in Algorithm 3.6, with the difference that the strategy parameters suffer genetic variation before the attribute vector.

Note that this new type of representation incorporates the strategy parameters into the encoding of the individuals of the population. This is a very powerful idea, because it allows the evolutionary search process to adapt the strategy parameters together with the attribute vector. This process is known as *self-adaptation*. Despite self-adaptation, recombination operators are not always used. Recombination was placed within parenthesis in the paragraph above because most ESs only use mutation; no crossover between individuals is performed. Assuming this simpler case where there is no recombination, it remains to the designer only to define how the mutation and selection operators are performed.

Selection

There are basically two types of selection mechanisms in the standard ES: $(\mu + \lambda)$ selection and (μ, λ) selection. In the $(\mu + \lambda)$-ES, μ parents generate λ offspring and the whole population $(\mu + \lambda)$ is then reduced to μ individuals. Thus, selection operates in the set formed by parents and offspring. In this case, the parents survive until the offspring become better adapted to the environment (with higher fitness) than their parents. In the (μ, λ)-ES, μ parents generate λ offspring $(\lambda \geq \mu)$ from which μ will be selected. Thus, selection only operates on the offspring set.

Both $(\mu + \lambda)$-ES and (μ, λ)-ES have the same general structure. Consider thus the most general case of a (μ, λ)-ES. An individual $\mathbf{v} = (\mathbf{x}, \boldsymbol{\sigma}, \boldsymbol{\theta})$ is composed of three elements: $\mathbf{x} \in \Re^{l}$, $\boldsymbol{\sigma} \in \Re^{l\sigma}$, and $\boldsymbol{\theta} \in (0, 2\pi]^{l\theta}$, where l is the dimension of **x**, $l\sigma \in \{1, \ldots, l\}$, and $l\theta \in \{0, (2l - l\sigma)(l\sigma - 1)/2\}$ (Bäck et al., 2000b). The vector **x** is the attribute vector, $\boldsymbol{\sigma}$ is the vector of standard deviations, and $\boldsymbol{\theta}$ is the vector of rotating angles (see further explanation).

Crossover

There are a number of recombination operators that can be used in evolution strategies, either in their usual form, producing one new individual from two randomly selected parents, or in their global form, allowing attributes to be taken for one new individual potentially from all individuals available in the population. Furthermore, recombination is performed on strategy parameters as well as on the attribute vector. Usually, recombination types are different for the attribute vector and the strategy parameters. In the following, assume **b** and **b**' to stand for the actual parts of **v**. In a generalized manner, the recombination operators can take the following forms:

$$\mathbf{b}_i' = \begin{cases} \mathbf{b}_{S,i} & (1) \\ \mathbf{b}_{S,i} \text{ or } \mathbf{b}_{T,i} & (2) \\ \mathbf{b}_{S,i} + u(\mathbf{b}_{T,i} - \mathbf{b}_{S,i}) & (3) \\ \mathbf{b}_{Sj,i} \text{ or } \mathbf{b}_{Tj,i} & (4) \\ \mathbf{b}_{Sj,i} + u(\mathbf{b}_{Tj,i} - \mathbf{b}_{Sj,i}) & (5) \end{cases} \quad \textbf{(3.4)}$$

where S and T denote two arbitrary parents, u is a uniform random variable over [0,1], and i and j index the components of a vector and the vector itself, respectively. In a top-down manner we have (1) no recombination, (2) discrete recombination (or uniform crossover), (3) intermediate recombination, (4) global discrete recombination, and (4) global intermediate recombination.

Mutation

The mutation operator in ESs works by adding a realization of an l-dimensional random variable with uniform distribution $X \sim \mathbf{N}(\mathbf{0},\mathbf{C})$, $\mathbf{0}$ as the expectation vector and co-variance matrix given by

$$\mathbf{C} = (c_{ij}) = \begin{cases} \text{cov}(X_i, X_j) & i \neq j \\ \text{var}(X_i) & i = j \end{cases} \tag{3.5}$$

C is symmetric and positive definite, and has the probability density function given by

$$f_X(x_1,\ldots,x_n) = \sqrt{\frac{\det(\mathbf{C})}{(2\pi)^l}} \exp\left(\frac{1}{2}\mathbf{x}^T\mathbf{C}^{-1}\mathbf{x}\right) \tag{3.6}$$

where the matrix **C** is the co-variance matrix described by the mutated values $\boldsymbol{\sigma}'$ and $\boldsymbol{\theta}'$ of the strategy parameters. Depending on the number of parameters incorporated in the representation of an individual, the following types of self-adaptation procedures are possible.

A Single Standard Deviation for all Attributes of **x**

$l\sigma = 1$, $l\theta = 0$, $X \sim \sigma^t\, \mathbf{N}(\mathbf{0},\mathbf{I})$.

In this case, the standard deviation of all attributes of **x** are identical, and all attributes are mutated as follows:

$$\begin{aligned} \sigma^{t+1} &= \sigma^t \exp(\tau_0.N(0,1)) \\ x_i^{t+1} &= x_i^t + \sigma^{t+1}.N_i(0,1) \end{aligned} \tag{3.7}$$

where $\tau_0 \propto l^{-1/2}$ and $N_i(0,1)$ indicates that the random variable is sampled independently for each i. The lines of equal probability density of the normal distribution are hyperspheres in an l-dimensional space.

Individual Standard Deviations for each Attribute of **x**

$l\sigma = l$, $l\theta = 0$, $X \sim \mathbf{N}(\mathbf{0}, \sigma^t.\mathbf{I})$.

Each attribute of **x** has its own standard deviation σ_i that determines its mutation as follows:

$$\begin{aligned} \sigma_i^{t+1} &= \sigma_i^t \exp(\tau'.N(0,1) + \tau.N_i(0,1)) \\ x_i^{t+1} &= x_i^t + \sigma_i^{t+1}.N_i(0,1) \end{aligned} \tag{3.8}$$

where $\tau' \propto (2.l)^{-1/2}$ and $\tau \propto (2.l^{1/2})^{-1/2}$.

The lines of equal probability density of the normal distribution are hyperellipsoids.

Individual Standard Deviations for each Attribute of **x** *and Correlated Mutation*

$l\sigma = l$, $l\theta = l.(l-1)/2$, $X \sim \mathbf{N}(\mathbf{0},\mathbf{C})$.

The vectors $\boldsymbol{\sigma}$ and $\boldsymbol{\theta}$ represent the complete covariance matrix of an l-dimensional normal distribution, where the co-variances c_{ij} ($i \in \{1, \ldots, l-1\}$, and $j \in \{i+1, \ldots, l\}$) are represented by a vector of rotation angles $\boldsymbol{\theta}_k$ that describe the coordinate rotations necessary to transform an uncorrelated mutation vector into a correlated mutation vector. Thus, the mutation is performed as follows:

$$\begin{aligned} \sigma_i^{t+1} &= \sigma_i^t \exp(\tau'.N(0,1) + \tau.N_i(0,1)) \\ \theta_i^{t+1} &= \theta_i^t + \beta.N_j(0,1) \\ \mathbf{x}^{t+1} &= \mathbf{x}^t + \mathbf{N}(\mathbf{0},\mathbf{C}(\boldsymbol{\sigma}^{t+1},\boldsymbol{\theta}^{t+1})) \end{aligned} \tag{3.9}$$

where $\mathbf{N}(\mathbf{0}, \mathbf{C}(\boldsymbol{\sigma}^{t+1}, \boldsymbol{\theta}^{t+1}))$ corresponds to the vector of correlated mutations and $\beta = 0.0873$ (5^o).

This way, the mutation corresponds to hyper-ellipsoids capable of rotating arbitrarily, and θ_k characterizes the rotation angles in relation to the coordinate axes.

The last step to implement the (μ, λ)-ES or ($\mu + \lambda$)-ES with correlated mutation is to determine $\mathbf{C}(\boldsymbol{\sigma}^{t+1}, \boldsymbol{\theta}^{t+1})$. Rudolph (1992) demonstrated that a symmetric matrix is positive definite if, and only if, it can be decomposed as $\mathbf{C} = (\mathbf{ST})^T\mathbf{ST}$, where **S** is a diagonal matrix with positive values corresponding to the standard deviations ($s_{ii} = \sigma_i$) and

$$\mathbf{T} = \prod_{i=1}^{l-1}\prod_{j=i+1}^{l} \mathbf{R}_{ij}(\theta_{ij}) \tag{3.10}$$

Matrix **T** is orthogonal and is built by a product of $l(l-1)/2$ rotation matrices $\mathbf{R}_{ij}$ with angles $\theta_k \in (0,2\pi]$. An elementary rotation matrix $\mathbf{R}_{ij}(\theta)$ is the identity matrix where four specific entries replaced by $r_{ii} = r_{jj} = cos\theta$ and $r_{ij} = -r_{ji} = -sen\theta$.

3.6.2. Evolutionary Programming

Fogel and collaborators (Fogel et al., 1966) introduced *evolutionary programming* (EP) as an evolutionary technique to develop an alternative form of *artificial intelligence*. An intelligent behavior was believed to require two capabilities: 1) the prediction of a certain environment, and 2) an appropriate response according to this prediction. The environment was considered as being described by a finite sequence of symbols taken from a finite alphabet. The evolutionary task was then defined as the evolution of an algorithm capable of operating in the sequence of symbols generated by the environment in order to produce an output symbol that maximizes the performance of the algorithm in relation to the next input symbol, given a well-defined cost function. *Finite state machines* (FSM) were considered suitable to perform such task.

Although the first evolutionary programming techniques were proposed to evolve finite state machines, they were later extended by D. Fogel (1992) to operate with real-valued vectors subject to Gaussian mutation, in a form similar to the one used in evolution strategies.

In a later development of evolutionary programming, called meta-evolutionary programming (meta-EP), an individual of the population is represented by $\mathbf{v} = (\mathbf{x},\mathbf{var})$, where $\mathbf{x} \in \Re^l$ is the attribute vector, $\mathbf{var} \in \Re^{l\mathrm{var}}$ is the variance vector ($var = \sigma^2$) of the mutation variables, l is the dimension of $\mathbf{x}$, and $l\mathrm{var} \in \{1, \dots , l\}$. The variance is also going to suffer mutation in a self-adaptive manner, similarly to the evolution strategies. In another approach, Fogel (1992, pp. 287-289) introduced the *Rmeta-PE*, which incorporates the vector of the correlation coefficients into the set of parameters that will suffer mutation. It was implemented in a form essentially identical to the evolution strategies, but few experiments were performed with this version of EP.

Selection

The selection mechanism used in evolutionary programming is essentially similar to the *tournament selection* sometimes used in genetic algorithms. After generating μ offspring from μ parents and mutating them (see next section), a stochastic tournament will select μ individuals from the set formed by the parents plus offspring. It thus corresponds to a stochastic $(\mu + \lambda)$ selection in evolution strategies.

For each individual $\mathbf{v}_i \in \mathbf{P} \cup \mathbf{P}'$, where $\mathbf{P}$ denotes the whole population and $\mathbf{P}'$ denotes the offspring population, q individuals are randomly selected from the set $\mathbf{P} \cup \mathbf{P}'$ and compared with $\mathbf{v}_i$ in relation to their fitness values. Given $\mathbf{v}_i$, we count how many individuals have a fitness value less than $\mathbf{v}_i$, resulting in a score w_i ($i \in \{1, \dots , 2\mu\}$), and the μ individuals with greater score w_i are selected to form the population for the next generation:

$$w_i = \sum_{j=1}^{q} \begin{cases} 1 & \text{if } fit(\mathrm{v}_i) \leq fit(\mathrm{v}_{\chi_j}) \\ 0 & \text{otherwise} \end{cases} \qquad \textbf{(3.11)}$$

where $\chi_j \in \{1, \dots , 2\mu\}$ is an integer and uniform random variable sampled for each comparison.

Mutation

In the standard EP, a Gaussian mutation with independent standard deviation for every element x_i of $\mathbf{x}$ is obtained as the square root of a linear transformation applied to the fitness value of $\mathbf{x}$ (Bäck and Schwefel, 1993):

$$\begin{aligned} x_i^{t+1} &= x_i^t + \sigma_i.N_i(0,1) \\ \sigma_i &= \sqrt{\beta_i.fit(\mathbf{x}) + \gamma_i} \end{aligned} \qquad \textbf{(3.12)}$$

where the parameters β_i and γ_i must be adjusted according to the problem under study.

To solve the parameter adjustment problems of EP, the meta-EP self-adapts l variances by individual, similarly to what is performed with the ES:

$$\begin{aligned} x_i^{t+1} &= x_i^t + \sqrt{var_i}\,.N_i(0,1) \\ var_i^{t+1} &= var_i^t + \sqrt{\alpha.var_i}\,.N_i(0,1) \end{aligned} \qquad \textbf{(3.13)}$$

where α is a parameter that guarantees that var_i is nonnegative, and $N_i(0,1)$ indicates that the random variable is sampled independently for each i. Fogel (1992) proposed the simple rule $\forall var_i \leq 0,\ var_i \leftarrow \xi$, where ξ is a value close to zero, but not zero.

Although this idea is similar to the one used in the ES, the stochastic process behind the meta-EP is fundamentally different from the one behind ES (Bäck, 1994). The procedure to implement EP is the same as the one used to implement ES, but with $\lambda = \mu$, and is just a slight variation of the standard evolutionary algorithm procedure (Algorithm 3.6).

3.6.3. Genetic Programming

Genetic programming (GP), proposed by Cramer (1985) and further developed and formalized by Koza (1992, 1994a), constitutes a type of evolutionary algorithm developed as an extension of genetic algorithms. In GP, the data structures that suffer adaptation are representations of computer programs, and thus the fitness evaluation involves the execution of the programs. Therefore, GP involves a search based upon the evolution of the space of possible computer programs such that, when run, they will produce a suitable output, which is usually related to solving a given problem.

The main question that motivated the proposal of genetic programming was "How can computers learn to solve problems without being explicitly programmed to do so?" In its original form, a computer program is basically the application of a sequence of functions to arguments: *functional paradigm*. Implementing GP is conceptually straightforward when associated with programming languages that allow the manipulation of a computer program in the form of data structures, such as the LISP programming language. However, it is possible to implement GP using virtually any programming language that can manipulate computer programs as data structures and that can then compile, link and execute the new data structures (programs) that arise. As LISP was the original language used to study and apply GP and is easily understandable, it will be briefly reviewed in the following.

In LISP there are only two types of entities: *atoms* (constants and variables) and *lists* (ordered set of items delimited by parentheses). A symbolic expression, or *S*-expression, is defined as an atom or a list, and it constitutes the only syntactic form in LISP; that is, all programs in LISP are *S*-expressions. In order to evaluate a list, its first element is taken as a function to be applied to the other elements on the list. This implies that the other elements from the list must be evaluated before they can be used as arguments for the function, represented by the first element of the list. For example, the *S*-expression (– 4 1)

requires function '–' to be applied to two arguments represented by the atoms '4' and '1'. The value returned by evaluating this expression is '3'.

A list may also contain other lists as arguments, and these are evaluated in a recursive, depth-first way; for instance, the *S*-expression (– (× 2 4) 6) returns '2' when evaluated. Therefore, the programs in LISP can be seen as compositions of functions. It is also important to note that these functions must not be restricted to simple arithmetic operators. It is possible to implement all types of constructs that can be used in a computer program (sequence, selection, and repetition). An important feature of LISP is that all LISP programs have a single *S*-expression that corresponds to the parse tree of the program. Figure 3.23 illustrates the parse tree representations of both *S*-expressions exemplified above.

As with all other evolutionary algorithms, in GP a problem is defined by its representation and the specification of an objective function that will allow the definition of a fitness function. The representation in GP consists of choosing an appropriate *function set F* and an appropriate *terminal set T*. The functions chosen are those that are *a priori* believed to be useful and sufficient for the problem at hand, and the terminals are usually variables or constants. The computer programs in the specified language are then the individuals of the population that can be represented in a data structure such as a tree. These individuals have to be executed so that the phenotype of the individual is obtained. While defining the representation in GP, two important aspects must be taken into account:

- *Syntactic closure*: it is necessary to examine the values returned by all functions and terminals such that all of them will accept as argument any value and data type that may be returned by a function from *F* or a terminal from *T*.
- *Sufficiency*: defining the search space corresponds to defining *F* and *T*. It is intuitive thus that the search space has to be large enough to contain the desired solution. Besides, the functions and terminals chosen have to be adequate to the problem domain.

Several types of functions and terminals can be used, such as arithmetic functions (e.g., +, ×, –, /), logic functions (e.g., AND, OR, NAND), and standard programming functions. The terminals may be variables, constants, or functions that do not receive arguments.

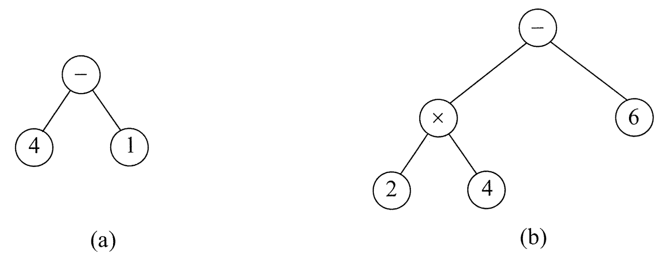


Figure 3.23: Parse tree representation of *S*-expressions in LISP. (a) (– 4 1). (b) (– (× 2 4) 6).

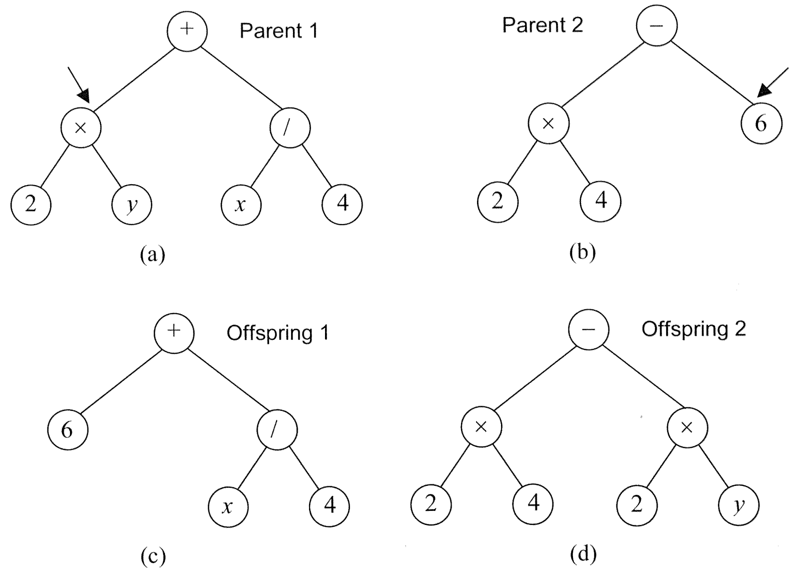


Figure 3.24: Crossover of *S*-expressions in GP. The crossover points are indicated by the arrows. (a) (+ (× 2 *y*) (/ *x* 4)). (b) (– (× 2 4) 6). (c) (+ 6 (/ *x* 4)). (d) (– (× 2 4) (× 2 *y*)).

Crossover

One interesting feature of genetic programming is that the individuals of the population may have different sizes. The recombination operator (crossover) is used to generate individuals (computer programs) for the next generation from parent programs chosen according to their fitness value. The offspring programs are composed of sub-trees from the parent programs, and may have different sizes and shapes from those of their parents. Similarly to the genetic algorithms, the application of crossover in GP requires two *S*-expressions as inputs. However, recombination cannot be random in this case, for it has to respect the syntax of the programs. This means that different crossover points may be chosen for each of the parent *S*-expressions. The crossover point isolates a sub-tree from each parent *S*-expression that will be exchanged to produce the offspring. Crossover in GP is illustrated in Figure 3.24. In this example, $F = \{+, –, ×, /\}$ and $T = \{1,2,3,4,5,6,x,y\}$.

Mutation

Mutation in GP may be useful but is not always used. Mutation may be important for the maintenance and introduction of diversity and also to maintain a dynamic variability of the programs being evolved. Mutation in GP is illustrated in Figure 3.25.

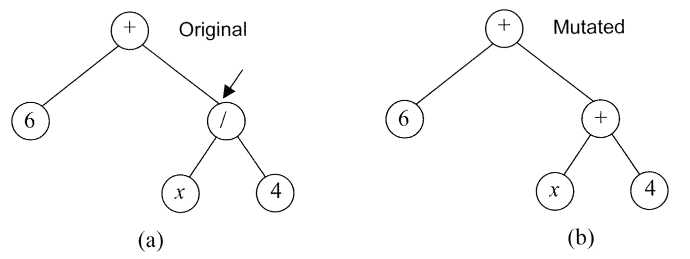


Figure 3.25: Mutation of an *S*-expression in GP. The mutation point is indicated by the arrow. (a) (+ 6 (/ x 4)). (b) (+ 6 (+ x 4)).

3.6.4. Selected Applications from the Literature: A Brief Description

This section illustrates the application of these three types of evolutionary algorithms (ES, EP, and GP) to three different problems. In the first example, a simple evolution strategy is applied to an engineering design problem. The second example illustrates the application of evolutionary programming to optimize the parameters of a voltage circuit, and the third example is a pattern classification task realized using a genetic programming technique. All descriptions are brief and aimed at illustrating the application potential of the approaches and the use of the three design principles: representation, objective, and function evaluation. Further details can be founded in the cited literature.

ES: Engineering Design

Engineering design is the process of developing a plan (e.g., a model, a system, a drawing) for a functional engineeing object (e.g., a building, a computer, a toaster). It often requires substantial research, knowledge, modelling and interactive adjustment and redesign until a useful and efficient plan is proposed. In (Hingston et al., 2002), the authors applied a simple evolution strategy to determine the geometry and operating settings for a crusher in a comminution circuit for ore processing. The task was to find combinations of design variables (including geometric shapes and machine settings) in order to maximize the capacity of a comminution circuit while minimizing the size of the crushed material.

The authors used a crusher with the shape of an upside down cone in which material is introduced from above and is crushed as it flows downward through the machine, as illustrated in Figure 3.26(a). Crushing is performed by an inner crushing surface, called mantle, mounted on the device and driven in an eccentric motion swivelling around the axis of the machine. The material is poured onto the top of the crushing chamber and is crushed between the mantle and the bowl liner when it passes through them. The area through which the material passes is called closed-side setting, and can be made wider or narrower depending on the desired size of the crushed material.

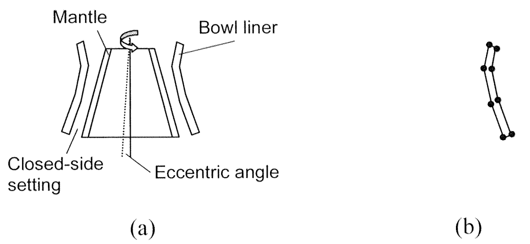


Figure 3.26: (a) Simplified diagram of the crusher used in Hingston et al., 2002. (b) Shape representation of each liner.

Evolutionary algorithms are particularly suitable for solving such types of problems because they are too complex to be solved analytically; a well-defined evaluation function for the designs can be proposed; there is little information *a priori* to guide an engineer in the determination of near-optimal designs; the search space is too large to be searched exhaustively; and once a candidate design has been proposed, it is possible to use a simulation tool to evaluate it. These are some of the features that make specific problems suitable for natural computing approaches, as discussed in Section 1.5.

The authors proposed the following representation, objective, and fitness function.

Representation: The representation of the individuals of the population corresponds to real-valued vectors in specific domains representing the closed-side settings, eccentric angle and rotational speed. This is a straightforward representation, for all parameters assume real values. The geometric shapes of the bowl liners were represented as a series of line segments using a variable-length list of points, each represented by its x-y coordinates on the plane (Figure 3.26(b)). Mutation was performed using individual standard deviations for each attribute of the chromosome representation (Equation (3.7)) and with varying strategy parameters following Equation (3.8). The selection method was the $(\mu + \lambda)$-ES, where $\mu = \lambda = 1$. Note that not only the device parameters are being optimized, but also the shape (geometry) of the bowl liners.

Objective: The objective for this problem is to maximize a function g, which corresponds to maximizing the capacity of the comminution circuit while minimizing the size of the product.

Evaluation: The fitness function used is directly proportional to the crushing capacity C of the circuit in terms of tons per hour, and to the size measure of the crushed product P:

$g(C,P) = 0.05 \times C + 0.95 \times P$,

As one of the goals is to minimize the size of the crushed product, P refers to a size s in mm such that 80% of the product is smaller than s mm. Therefore, the fitness function $g(C,P)$ to be maximized takes into account the capacity of the circuit and the percentage of material crushed within a given specification.

The constants multiplying each factor were chosen by the authors so as to equalize their variability, quantifying the importance of each component. Note that, in this case, after a plant has been designed by the evolutionary algorithm, a simulation tool is used to evaluate and compare alternative designs (candidate solutions), thus automating the process.

EP: Parameter Optimization

The process of engineering design involves mainly two levels: topology design and parameter optimization. In the previous example we discussed the use of an evolution strategy to perform both, topology design (bowl liner shape) and parameter design (combination of appropriate eccentric angle, closed-side setting, and rotation speed, which will influence C and P).

In (Nam et al., 2001), the authors applied an evolutionary programming approach to the problem of optimizing a set of parameters for a voltage reference circuit. Similarly to the engineering design problem mentioned above, in order to design an electronic circuit, several parameters of each part of the circuit have to be determined, but, in the present example, the topology of the circuit was already defined. The authors argued that such problems are highly multi-modal, making it difficult to apply more traditional techniques, such as gradient-based search and justifying the use of alternative approaches like EAs. Besides, the parameters of the circuit to be optimized are real-valued vectors, one reason why the authors chose to use evolutionary programming, though an evolution strategy could have been used instead or a genetic algorithm with float-point encoding.

The circuit whose parameters they wanted to optimize was an on-chip voltage reference circuit; that is, a circuit from semiconductor technology that generates a reference voltage independent of power fluctuation and temperature variation. Given the power voltage V_T and the temperature T, it is possible to write the reference voltage V_{ref} equation for the circuit as a function of several parameters, such as resistors, transistors, etc. The goal is to find appropriate parameter values that maintain the reference voltage stable when the temperature and power voltage vary within given ranges. A simplified equation for the reference voltage can be written as a function of only V_T and T:

$$V_{ref} = K_1 V_T + K_2 T \tag{3.14}$$

where K_1 and K_2 are two constants, V_T is the power voltage and T is the temperature.

Representation: The representation used was real-valued vectors and each candidate solution corresponded to a combination of parameter values. The authors used a ($\mu + \lambda$) selection scheme and a simplified mutation scheme. Equation (3.15) below describes how the authors evolved the attribute values of the vector to be optimized **x** and the strategy parameters as well.

$$\begin{aligned} x_i^{t+1} &= x_i^{t} + \sigma_i . N_i(0,1) \\ \sigma_i^{t} &= \beta_i . f(\mathbf{x}^{t}) \end{aligned} \tag{3.15}$$

where x_i^t is the i-th parameter of one chromosome at iteration t, σ_i its associated variance, $N_i(0,1)$ is a normal distribution of zero mean and standard deviation 1, β_i is a positive constant, and $f(\mathbf{x})$ is the fitness of chromosome **x**. To determine $fit(\mathbf{x})$, a cost function was proposed taking into account a set of parameters to be optimized.

Objective: The objective is to minimize a function $f(\mathbf{x})$ that takes into account a reference voltage, temperature compensation and an active voltage constraint.

Evaluation: Based on this objective, the proposed fitness function was:

$$f(\mathbf{x}) = K_1|V_{25}(\mathbf{x}) - 2.5| + K_2(|V_0(\mathbf{x}) - V_{25}(\mathbf{x})| + |V_{25}(\mathbf{x}) - V_{100}(\mathbf{x})|) +$$
$$+ K_3|V_{\text{act0}}(\mathbf{x}) - 0.8|,$$

where V_0, V_{25}, and V_{100} represent reference voltages at 0°C, 25°C, and 100°C, respectively, and V_{act0} is an active bias voltage.

In this approach a simulation tool is also used to evaluate the performance of the evolved parameters.

GP: Pattern Classification

Adaptation problems often present themselves as problems of classification or function approximation. In classification problems, the objective is to discern a pattern from others and to develop a procedure capable of successfully performing the classification. The procedure is usually developed using a set of input data with known classification and new, previously unseen, data is used to evaluate the suitability of the adaptation (classification) process. The *generalization capability* of the classifier is assessed on these novel data. Learning relationships that successfully discriminate among examples associated with problem solving choices is a typical application of natural computing.

One possible application of genetic programming is in the design of classifiers (Koza, 1992, 1994b). Consider the input data set illustrated in Figure 3.27.

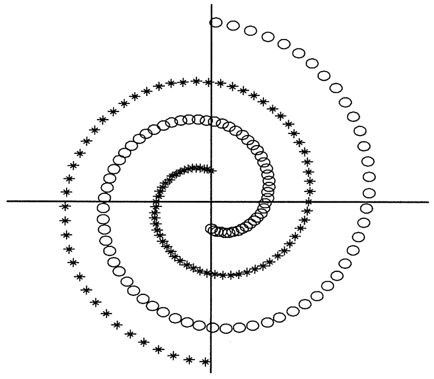


Figure 3.27: Input data set used to test the performance of many classifiers.

This is a typical data set used to test some classification techniques, including neural networks (Chapter 4), whose difficulty lies in the fact that data are presented using Cartesian coordinates making them nonlinearly separable.

This problem can be solved using a genetic programming approach (Koza, 1992). In this case, the GP will have to provide as output a computer program (*S*-expression) that determines to which spiral a given point belongs. Assume that the points belonging to one of the spirals correspond to class +1 and the points belonging to the other spiral correspond to class −1.

Representation: It is important to remember that the representation in GP corresponds to a function set *F* and a terminal set *T* believed to be sufficient for solving the problem. Due to the nature of the spirals problem, the terminal set *T* consists of the *x* and *y* coordinates of all given points (190 in the picture shown). As some numerical constants $c \in \Re$ may be needed to process all these data, some constants, $c \in [-1,+1]$, can be added to the terminal set. Thus, the terminal set is $T = \{\mathbf{X},\mathbf{Y},\Re\}$, where **X** and **Y** are the sets of coordinates for the *x* and *y* variables of the data set and $\Re$ is the set of real numbers belonging to the interval [−1,+1]. To determine the function set for this problem, we must have in mind that the resultant computer program has to determine to which of the spirals a given point belongs. Thus, the programs to be evolved may include the four arithmetic operations, a conditional comparative function for decision making, and the trigonometric *sine* and *cosine* functions. The function set for this problem is then $F = \{+,-,\times,/,\text{IFLTE},sin,cos\}$, in which these functions take 2, 2, 2, 2, 4, 1, and 1 arguments, respectively. Function IFLTE (If-Less-Than-Or-Equal-To) is a four argument conditional comparative operator that executes its third argument if its first argument is less than its second argument and, otherwise, executes the fourth (else) argument. The conditional IFLTE is used so as to allow the classification of points into one of the classes, and functions *sine* and *cosine* are included so as to increase nonlinearity. In order to have a binary classification in the output, a wrapper must be employed mapping any positive value returned by the evolved program when executed to class +1, and to class −1 otherwise.

Objective: The objective is to maximize the number of points correctly classified or, equivalently, minimize the number of points misclassified.

Evaluation: Assuming the objective is to maximize the correct classification rate of the *S*-expression, fitness can be calculated by taking the total number of correctly classified points.

3.7 FROM EVOLUTIONARY BIOLOGY TO COMPUTING

This chapter described the standard evolutionary algorithms with particular emphasis on the genetic algorithms. Table 3.2 suggests an interpretation of the biological terminology into the evolutionary algorithms.

Table 3.2: Interpretation of the biological terminology into the computational world for genetic algorithms.

Biology (Genetics)	Evolutionary Algorithms
Chromosome	Data structure
Genotype	Encoding of a potential candidate solution to a problem (equivalent to a chromosome)
Phenotype	Decoded value of one or more chromosomes
Gene	Element occupying a given position in a data structure
Locus	Position occupied by a gene in a data structure
Alleles	Variations of an element (gene) that can occupy a locus
Crossover	Exchange of portions between data structures
Mutation	Replacement of a given gene by a different one
Fitness	Value that indicates the quality of an individual in relation to the problem being solved
Selection	Process that allows the survival and reproduction of the fittest individuals in detriment of the less fit ones

3.8 SCOPE OF EVOLUTIONARY COMPUTING

Given a problem, how do we know if an evolutionary algorithm can solve it effectively? Brief discussions about it have been made throughout this chapter and Chapter 1, but there is no rigorous answer to this question, though some key intuitive aspects can be highlighted (Mitchell, 1998):

- If the search space is large, neither perfectly smooth nor unimodal, is unknown, or if the fitness function is noisy, then an EA will have a good chance of being a competitive approach.
- If the search space is smooth or unimodal, then *gradient* or *hill-climbing* methods perform much better than evolutionary algorithms.
- If the search space is well understood (such as in the *traveling salesman problem* – TSP), heuristics can be introduced in specific methods, including the EAs, such that they present good performances.

Beasley (2000) divided the possible application areas of evolutionary algorithms into five broad categories: planning (e.g., routing, scheduling and packing); design (e.g., signal processing); simulation, identification, control (general plant control); and classification (e.g., machine learning, pattern recognition and classification). More specifically, evolutionary algorithms have been applied in fields like art and music composition (Bentley, 1999; Bentley and Corne, 2001), electronics (Zebulum et al., 2001), language (O'Neill and Ryan, 2003), robotics (Nolfi and Floreano, 2000), engineering (Dasgupta, and Michalewicz, 1997), data mining and knowledge discovery (Freitas and Rozenberg, 2002), industry (Karr and Freeman, 1998), signal processing (Fogel, 2000b), and many others.

This is a typical data set used to test some classification techniques, including neural networks (Chapter 4), whose difficulty lies in the fact that data are presented using Cartesian coordinates making them nonlinearly separable.

This problem can be solved using a genetic programming approach (Koza, 1992). In this case, the GP will have to provide as output a computer program (*S*-expression) that determines to which spiral a given point belongs. Assume that the points belonging to one of the spirals correspond to class +1 and the points belonging to the other spiral correspond to class –1.

Representation: It is important to remember that the representation in GP corresponds to a function set F and a terminal set T believed to be sufficient for solving the problem. Due to the nature of the spirals problem, the terminal set T consists of the x and y coordinates of all given points (190 in the picture shown). As some numerical constants $c \in \Re$ may be needed to process all these data, some constants, $c \in [-1,+1]$, can be added to the terminal set. Thus, the terminal set is $T = \{\mathbf{X},\mathbf{Y},\Re\}$, where **X** and **Y** are the sets of coordinates for the x and y variables of the data set and $\Re$ is the set of real numbers belonging to the interval $[-1,+1]$. To determine the function set for this problem, we must have in mind that the resultant computer program has to determine to which of the spirals a given point belongs. Thus, the programs to be evolved may include the four arithmetic operations, a conditional comparative function for decision making, and the trigonometric *sine* and *cosine* functions. The function set for this problem is then $F = \{+,-,\times,/,\text{IFLTE},sin,cos\}$, in which these functions take 2, 2, 2, 2, 4, 1, and 1 arguments, respectively. Function IFLTE (If-Less-Than-Or-Equal-To) is a four argument conditional comparative operator that executes its third argument if its first argument is less than its second argument and, otherwise, executes the fourth (else) argument. The conditional IFLTE is used so as to allow the classification of points into one of the classes, and functions *sine* and *cosine* are included so as to increase nonlinearity. In order to have a binary classification in the output, a wrapper must be employed mapping any positive value returned by the evolved program when executed to class +1, and to class –1 otherwise.

Objective: The objective is to maximize the number of points correctly classified or, equivalently, minimize the number of points misclassified.

Evaluation: Assuming the objective is to maximize the correct classification rate of the *S*-expression, fitness can be calculated by taking the total number of correctly classified points.

3.7 FROM EVOLUTIONARY BIOLOGY TO COMPUTING

This chapter described the standard evolutionary algorithms with particular emphasis on the genetic algorithms. Table 3.2 suggests an interpretation of the biological terminology into the evolutionary algorithms.

Table 3.2: Interpretation of the biological terminology into the computational world for genetic algorithms.

Biology (Genetics)	Evolutionary Algorithms
Chromosome	Data structure
Genotype	Encoding of a potential candidate solution to a problem (equivalent to a chromosome)
Phenotype	Decoded value of one or more chromosomes
Gene	Element occupying a given position in a data structure
Locus	Position occupied by a gene in a data structure
Alleles	Variations of an element (gene) that can occupy a locus
Crossover	Exchange of portions between data structures
Mutation	Replacement of a given gene by a different one
Fitness	Value that indicates the quality of an individual in relation to the problem being solved
Selection	Process that allows the survival and reproduction of the fittest individuals in detriment of the less fit ones

3.8 SCOPE OF EVOLUTIONARY COMPUTING

Given a problem, how do we know if an evolutionary algorithm can solve it effectively? Brief discussions about it have been made throughout this chapter and Chapter 1, but there is no rigorous answer to this question, though some key intuitive aspects can be highlighted (Mitchell, 1998):

- If the search space is large, neither perfectly smooth nor unimodal, is unknown, or if the fitness function is noisy, then an EA will have a good chance of being a competitive approach.
- If the search space is smooth or unimodal, then *gradient* or *hill-climbing* methods perform much better than evolutionary algorithms.
- If the search space is well understood (such as in the *traveling salesman problem* – TSP), heuristics can be introduced in specific methods, including the EAs, such that they present good performances.

Beasley (2000) divided the possible application areas of evolutionary algorithms into five broad categories: planning (e.g., routing, scheduling and packing); design (e.g., signal processing); simulation, identification, control (general plant control); and classification (e.g., machine learning, pattern recognition and classification). More specifically, evolutionary algorithms have been applied in fields like art and music composition (Bentley, 1999; Bentley and Corne, 2001), electronics (Zebulum et al., 2001), language (O'Neill and Ryan, 2003), robotics (Nolfi and Floreano, 2000), engineering (Dasgupta, and Michalewicz, 1997), data mining and knowledge discovery (Freitas and Rozenberg, 2002), industry (Karr and Freeman, 1998), signal processing (Fogel, 2000b), and many others.

3.9 SUMMARY

This text introduced some basic concepts for problem solving, focusing optimization problems. Several versions of the hill-climbing procedure were presented for comparison with simulated annealing and evolutionary algorithms. The emphasis was always on how the standard algorithms were developed, their respective sources of inspiration, and how to interpret the natural motivation into the computational domain. It was discussed that simulated annealing was developed using ideas gleaned from statistical thermodynamics, while evolutionary algorithms were inspired by the neo-Darwinian theory of evolution. All strategies reviewed are conceptually very simple and constitute general-purpose search techniques.

Basic principles of genetics were reviewed so as to prepare the reader for an appropriate understanding of evolutionary algorithms, in particular genetic algorithms and their functioning. Some discussion about the philosophy of the theory of evolution and what types of outcomes can be expected from EAs was also provided. The focus of the chapter was on a description of evolutionary algorithms - those inspired by evolutionary biology - as a problem-solving approach. Any algorithm based upon a population of individuals (data structures) that reproduce, and suffer variation followed by selection, can be characterized as an EA. The standard evolutionary algorithm was presented in Algorithm 3.6, and a set of elementary genetic operators (crossover and mutation) was described for the standard genetic algorithm and the other traditional evolutionary algorithms.

Particular attention was also given to a perspective of evolution as an algorithmic process applied in problem solving. Under this perspective, the problem to be solved plays the role of the environment, and each individual of the population is associated with a candidate solution to the problem. Thus, an individual will be more adapted to the environment (will have a higher fitness value) whenever it corresponds to a 'better' solution to the problem. At each evolutionary generation, improved candidate solutions should be produced, though there is no guarantee that an optimal solution can be found. The evolutionary algorithm thus constitutes a useful iterative and parallel search procedure, suitable for solving search and optimization problems that involve a vast search space.

3.9.1. The Blind Watchmaker

To conclude this chapter on evolutionary biology and algorithms, it is important to say a few words about the type of search being performed by evolution and thus evolutionary algorithms.

Until Darwin's proposal, almost everyone embraced the idea that living systems were designed by some God-like entity. Even scientists were convinced by the so-called *watchmaker* argument; that is, the argument from design, proposed by theologian William Paley in his 1802 book *Natural Theology*. Paley noted that watches are very complex and precise objects. If you found a watch on the ground, you could not possibly believe that such a complex object had been created by random chance. Instead, you would naturally conclude that the watch

must have had a maker; that there must have existed sometime, somewhere, an *artificer*, who designed and built it for the purpose which we find it actually to answer. For Paley, the same logic argument applies to living systems. Therefore, living systems, like watches, must have a maker, concluded Paley.

The point to be raised here is the one discussed by Richard Dawkins in his 1986 awarded book *The Blind Watchmaker*, namely, that evolution by means of natural selection might be seen as a *blind watchmaker* (see Chapter 8).

"In the case of living machinery, the 'designer' is unconscious natural selection, the blind watchmaker." (Dawkins, 1986, p. 45)

The philosophical discussions presented so far have already focused on the algorithmic (designer's) potentiality of evolution. What has to be emphasized now is that it is a blind (i.e., *unsupervised*) process.

While studying the example given in Section 3.5.3, one might be inclined to think that evolutionary algorithms are employed when the solution to a problem is known. However, this is <u>not</u> usually the case and deserves some comments. While in the first example, provided for the GA, the solution was indeed known, in the second example no assumption about the optimum of the problem had to be made so that the GA could be applied. The first example demonstrated that GAs are capable of generating an increasingly more fit population of individuals, thus being capable of evolving (adapting to the environment). The second example showed that GAs are capable of finding optimal solutions to problems. And both examples showed that though GAs might sometimes use information about the solution of a problem, they are never told <u>how</u> to find the solution, i.e., they perform a *blind search*. By simply initializing a population of chromosomes (represented using some type of data structure), creating copies of them (reproduction) and applying random genetic operators (performing variation and selection), it is possible to determine (unknown) solutions to complex problems. The same holds true for the other evolutionary algorithms presented: evolution strategies, evolutionary programming and genetic programming. All of them follow the same standard evolutionary procedure (Algorithm 3.6).

3.10 EXERCISES

3.10.1. Questions

1. This text introduced three versions of hill-climbing. The last version, namely stochastic hill-climbing, has many features in common with the simulated annealing algorithm. Do you find it necessary to know the inspiration taken from statistical thermodynamics in order to understand simulated annealing? Justify your answer.
2. Based upon the discussion presented concerning the evolution of species, provide an answer for the following classical question. What came first, the egg or the chicken? What arguments would you use to support your answer?

3. It is known that the eyes evolved more than once, in different times, and for different species. What do you think would be a good 'fitness measure' for a working 'eye'? Take the particular case of human eyes for example.
4. If evolution is considered to be a never-ending process, do humans present any sign of evolution from the last few thousand years? Justify your answer.
5. In addition to the classic example of the evolution of the moths in Great Britain (Section 3.4.5), can you provide any other example of an organism whose evolution can be (has been) clearly accompanied over the last years (or centuries, or thousands of years)?
6. Name at least three difficulties for the theory of evolution. Can you think of some arguments that could be used to refute these difficulties?
7. How would you explain the many similarities among living organisms? For instance, consider the case of whales and fishes. They belong to different species, but are similar in structure and habitat. Why would that be so?
8. If biology (e.g., evolution) can be seen as engineering, suggest another biological phenomenon or process that can be abstracted to an algorithmic level. Write down a procedure to implement it and propose two practical applications for this procedure.

3.10.2. Computational Exercises

1. Implement the various hill-climbing procedures and the simulated annealing algorithm to solve the problem exemplified in Section 3.3.3. Use a real-valued representation scheme for the candidate solutions (variable x).

   By comparing the performance of the algorithms, what can you conclude?

   For the simple hill-climbing, try different initial configurations as attempts at finding the global optimum. Was this algorithm successful?

   Discuss the sensitivity of all the algorithms in relation to their input parameters.
2. Implement and apply the hill-climbing, simulated annealing, and genetic algorithms to maximize function $g(x)$ used in the previous exercise assuming a bitstring representation.

   Tip: the perturbation to be introduced in the candidate solutions for the hill-climbing and simulated annealing algorithms may be implemented similarly to the point mutation in genetic algorithms. Note that in this case, no concern is required about the domain of x, because the binary representation already accounts for it.

   Discuss the performance of the algorithms and assess their sensitivity in relation to the input parameters.
3. Implement a standard genetic algorithm for the example of Section 3.5.3.

4. The algorithms presented in Section 3.3 were all devised assuming maximization problems. How would you modify the procedures in case minimization was desired? Rewrite the codes if necessary.
5. Apply an evolution strategy (ES) with no correlated mutation to the function maximization problem of Exercise 1.
6. Repeat the previous exercise with an evolutionary programming (EP) technique. Compare the results with those obtained in Exercises 1 and 5.
7. Determine, using genetic programming (GP), the computer program (S-expression) that produces exactly the outputs presented in Table 3.3 for each value of x. The following hypotheses are given:
   - Use only functions with two arguments (binary trees).
   - Largest depth allowed for each tree: 4.
   - Function set: $F = \{+, *\}$.
   - Terminal set: $T = \{0, 1, 2, 3, 4, 5, x\}$.

Table 3.3: Input data for the GP.

x	Program output
-10	153
-9	120
-8	91
-7	66
-6	45
-5	28
-4	15
-3	6
-2	1
-1	0
0	3
1	10
2	21
3	36
4	55
5	78
6	105
7	136
8	171
9	210
10	253

8. Solve the SPIR pattern recognition task of Section 3.6.4 using genetic programming. Assume the same hypotheses given in the example.

3.10.3. Thought Exercises

1. Evolutionary algorithms were introduced as general-purpose search algorithms developed with inspiration in evolutionary biology and applied for problem solving. Most of their applications are of an optimization type, but they can also be applied for design, arts, and so forth. For instance, one can use an EA to design tires for racing cars. In such a case, the tires would have to be encoded using some suitable representation, the evaluation function would have to take into account the endurance, shape, adherence to the ground in normal and wet conditions, etc., and the evaluation function would have to allow the distinction between candidate solutions (individual chromosomes) to your problem.

   Suggest a novel application for an evolutionary algorithm. This can be from your domain of expertise or any field of your interest. Provide a suitable representation, objective, and evaluation function. If you are using a binary representation, then the crossover and mutation operators described may be suitable for evolving a population of candidate solutions. However, if the representation chosen is not binary, suggest new crossover and mutation operators suitable for the proposed representation.
2. How would you implement a GA for a two variable numeric function? What would change: the representation, evaluation function, or genetic operators?
3. How would you hybridize a local search procedure, such as the standard hill-climbing, with a genetic algorithm? Provide a detailed discussion, including a discussion about the benefits of this hybridization.

3.10.4. Projects and Challenges

1. The *traveling salesman problem* (TSP) is a popular problem in combinatorial optimization with applications in various domains, from fast-food delivery to the design of VLSI circuits. In its simplest form, the traveling salesman must visit every city in a given territory exactly once, and then return to the starting city. Given the cost of travel between all cities (e.g., distance or cost in money), the question posed by the TSP is related to what should be the itinerary for the minimal cost of the whole tour.

   Implement an evolutionary algorithm to solve the pictorial TSP illustrated in Figure 3.28. The cities were placed on a regular grid to facilitate analysis and are labeled by a number on their top left corner. Three main aspects deserve careful examination: representation, evaluation function, and genetic operators.

   First, one has to define a representation for the chromosomes. Is a binary representation, such as the one used in standard genetic algorithms, suitable? If not, explain why, and suggest a new representation.

   Second, the definition of an evaluation function requires the knowledge of the objective. In this case, assume that the objective is simply to minimize

   the distance traveled by the salesman. The coordinates of each city on the x-y plane can be extracted from Figure 3.28.

   Lastly, if the representation you chose is not binary, what should be the crossover and mutation operators employed? Can you see any relationship between these operators and the biological crossover and mutation?

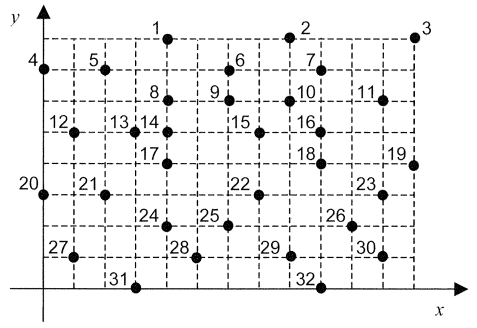


Figure 3.28: Simple TSP with 32 cities. The cities are placed on a regular grid on the x-y plane, in which each point (•) represents a city, the number next to each city corresponds to its index, and each square on the grid corresponds to one unit of distance (uod - e.g., Km).

2. In the evolutionary algorithm developed in the previous exercise, add a standard hill-climbing procedure as a means of performing local search in the individuals of the population.
3. Maximize function $g(x) = 2^{-2((x-0.1)/0.9)^2}(\sin(5\pi x))^6$, $x \in [0,1]$, using an evolution strategy with correlated mutation.

3.11 REFERENCES

[1] Anderson, R. L. (1953), "Recent Advances in Finding Best Operating Conditions", *Jour. of Am. Stat. Assoc.*, **48**, pp. 789–798.

[2] Bäck, T. and Schwefel, H.-P. (1993), "An Overview of Evolutionary Algorithms for Parameter Optimization", *Evolutionary Computation*, **1**(1), pp. 1–23.

[3] Bäck, T. (1996), *Evolutionary Algorithms in Theory and Practice*, Oxford University Press.

[4] Bäck, T., Fogel, D. B. and Michalewicz, Z. (eds.) (2000a), *Evolutionary Computation 1: Basic Algorithms and Operators*, Institut of Physics Publishing.

[5] Bäck, T., Fogel, D. B. and Michalewicz, Z. (eds.) (2000b), *Evolutionary Computation 2: Advanced Algorithms and Operators*, Institute of Physics Publishing.

[6] Banzhaf, W., Nordin, P., Keller, R. E. and Francone, F. D. (1998), *Genetic Programming – An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*, Morgan Kaufmann Publishers.

[7] Beasley, D. (2000), "Possible Applications of Evolutionary Computation", In Bäck et al. (2000a), Chapter 2, pp. 4–19.

[8] Bentley, P. J. and Corne, D. W. (2001), *Creative Evolutionary Systems*, Morgan Kaufmann.

[9] Bentley, P. J. (1999), *Evolutionary Design by Computers*, Morgan Kaufmann.

[10] Bertsimas, D. and Tsitsiklis, J. (1993), "Simulated Annealing", *Stat. Sci.*, 8(1), pp. 10–15.

[11] Beyer, H.-G. (2001), *Theory of Evolution Strategies*, Springer-Verlag.

[12] Bremermann, H. J. (1962), "Optimization through Evolution and Recombination", in M. C. Yovits, G. T. Jacobi and D. G. Goldstein (eds.), *Self-Organizing Systems*, Spartan, Washington, D.C., pp. 93–106.

[13] Černý, V. (1985), "Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm", *J. of Optim. Theory and App.*, 45(1), pp. 41–51.

[14] Colby, C. (1997), *Introduction to Evolutionary Biology*, TalkOrigins, [OnLine] (July, 11th, 2002) www.talkorigins.org/faqs/faq-intro-to-biology.html.

[15] Cramer, N. L. (1985), "A Representation for the Adaptive Generation of Simple Sequential Programs", In J. J. Grefenstette (ed.), *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Erlbaum.

[16] Darwin, C. R. (1859), *The Origin of Species*, Wordsworth Editions Limited (1998).

[17] Dasgupta, D. and Michalewicz, Z. (1997), *Evolutionary Algorithms in Engineering Applications*, Springer-Verlag.

[18] Davis, L. (ed.) (1991), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold.

[19] Dawkins, R. (1986), *The Blind Watchmaker*, Penguin Books.

[20] Dennett, D. C. (1995), *Darwin's Dangerous Idea: Evolution and the Meanings of Life*, Penguim Books.

[21] Eigen, M. (1992), *Steps Towards Life*, Oxford University Press.

[22] Elton, C. (1997), *Animal Ecology*, Sidgwick and Jackson, London.

[23] Fogel, D. B. (1992), *Evolving Artificial Intelligence*, Doctoral Dissertation, UCSD.

[24] Fogel, D. B. (2000a), *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, 2nd Ed., The IEEE Press.

[25] Fogel, D. B. (2000b), *Evolutionary Computation: Principles and Practice for Signal Processing*, SPIE – The International Society for Optical Engineering.

[26] Fogel, D. B. (ed.) (1998), *Evolutionary Computation: The Fossil Record*, The IEEE Press.

[27] Fogel, L. J., Owens, A. J. and Walsh, M. J. (1966), *Artificial Intelligence through Simulated Evolution*. John Wiley.

[28] Fraser, A. S. (1959), "Simulation of Genetic Systems by Automatic Digital Computers", *Aust. Jour. Of Biol. Sci.*, **10**, pp. 489–499.

[29] Freitas, A. A. and Rozenberg, G. (2002), *Data Mining and Knowledge Discovery with Evolutionary Algorithms*, Springer-Verlag.

[30] Friedberg, R. M. (1958), "A Learning Machine: Part I", *IBM Jour. of Research and Development*, **2**, pp. 2–13.

[31] Futuyma, D. J. (1998), *Evolutionary Biology*, 3rd Ed., Sinauer Associates, Inc.

[32] Gardner, E. J., Simmons, M. J. and Snustad, D. P. (1991), *Principles of Genetics*, John Wiley & Sons, Inc.

[33] Goldberg, D. E. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Addisson-Wesley Reading, MA.

[34] Griffiths, A. J. F., Miller, J. H., Suzuki, D. T., Lewontin, R. C. and Gelbart, W. M. (1996), *An Introduction to Genetic Analysis*, W. H. Freeman and Company.

[35] Hingston, P., Barone, L. and While, L. (2002), "Evolving Crushers", *Proc. of the IEEE Congress on Evolutionary Computation*, **2**, pp. 1109–1114.

[36] Holland, J. J. (1975), *Adaptation in Natural and Artificial Systems*, MIT Press.

[37] Karr, C. L. and Freeman, L. M. (1998), *Industrial Applications of Genetic Algorithms*, CRC Press.

[38] Kinnear Jr., K. E. (ed.) (1994), *Advances in Genetic Programming*, MIT Press.

[39] Kirkpatrick, S., Gerlatt, C. D. Jr. and Vecchi, M. P. (1983), "Optimization by Simulated Annealing", *Science*, 220, pp. 671–680.

[40] Koza, J. R. (1992), *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press.

[41] Koza, J. R. (1994a), *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press.

[42] Koza, J. R. (1994b), "Genetic Programming as a Means for Programming Computers by Natural Selection", *Statistics and Computing*, **4**(2), pp. 87–112.

[43] Krebs, C. J. (1994), *Ecology: The Experimental Analysis of Distribution and Abundance*, 4th Ed., Harper Collins College Publishers.

[44] Man, K. F., Tang, K. S. and Kwong, S. (1999), *Genetic Algorithms: Concepts and Designs*, Springer Verlag.

[45] Mayr, E. (1988), *Toward a New Philosophy of Biology: Observations of an Evolutionist*, Belknap, Cambridge, MA.

[46] Mendel, G. (1865), "Versuche uber pflanzenhybriden", *J. Hered.*, **42**, pp. 1–47, English translation "Experiments in Plant Hybridization", Harvard University Press, Cambridge, MA.

[47] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M. N., Teller, A.H. and Teller, E. (1953), "Equations of State Calculations by Fast Computing Machines", *J. Chem. Phys.*, 21, pp. 1087–1092.

[48] Michalewicz, Z. (1996), *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag.

[49] Michalewicz, Z. and Fogel, D. B. (2000), *How To Solve It: Modern Heuristics*, Springer-Verlag, Berlin.

[50] Mitchell, M. (1996), *An Introduction to Genetic Algorithms*, The MIT Press.

[51] Nam, D. K., Seo, Y. D., Park, L. J., Park, C. H. and Kim, B. S. (2001), "Parameter Optimization of an On-Chip Voltage Reference Circuit Using Evolutionary Programming", *IEEE Trans. Evolutionary Computation*, **5**(4), pp 414–421.

[52] Nolfi, S. and Floreano, D. (2000), *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines (Intelligent Robotics and Autonomous Agents)*, MIT Press.

[53] O'Neill, M. and Ryan, C. (2003), *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*, Kluwer Academic Publishers.

[54] Paley, W. (1802), *Natural Theology – or Evidences of the Existence and Attributes of the Deity Collected from the Appearances of Nature*, Oxford: J. Vincent.

[55] Pincus, M. (1970), "A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems", *Oper. Res.*, 18, pp. 1225–1228.

[56] Rechenberg, I. (1965), "Cybernetic Solution Path of an Experimental Problem", *Roy. Aircr. Establ. Libr. Transl.*, 1122, Farnborough, Hants, UK.

[57] Russel, P. J. (1996), *Genetics*, 4th Ed., Harper Collins College Publishers.

[58] Russell, S. J. and Norvig, P. (1995), *Artificial Intelligence A Modern Approach*, Prentice Hall, New Jersey, U.S.A.

[59] Sarle, W. (1993), *Kangoroos*, article posted on *comp.ai.neural-nets* on the 1st September.

[60] Schwefel, H.-P. (1965), *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik*, Diploma Thesis, Technical University of Berlin, March.

[61] Schwefel, H.-P. (1981), *Numerical Optimization of Computer Models*, Wiley, Chichester.

[62] Schwefel, H.-P. (1995), *Evolution and Optimum Seeking*, Wiley: New York.

[63] Wolpert, D. H. and Macready, W. G. (1997), "No Free Lunch Theorems for Optimization", *IEEE Trans. on Evolutionary Computation*, **1**(1), pp. 67–82.

[64] Wright, S. (1968–1978), *Evolution and the Genetics of Population*, 4 vols., University of Chicago Press, Chicago, IL.

[65] Zebulum, R. S., Pacheco, M. A., Vellasco, M. M. B. and Zebulum, R. S. (2001), *Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*, CRC Press.

CHAPTER 4

NEUROCOMPUTING

"Inside our heads is a magnificent structure that controls our actions and somehow evokes an awareness of the world around ... It is hard to see how an object of such unpromising appearance can achieve the miracles that we know it to be capable of."
(R. Penrose, The Emperor's New Mind, Vintage, 1990; p. 483)

"Of course, something about the tissue in the human brain is necessary for our intelligence, but the physical properties are not sufficient ... Something in the patterning of neural tissue is crucial."
(S. Pinker, How the Mind Works, The Softback Preview, 1998; p. 65)

4.1 INTRODUCTION

How does the brain process information? How is it organized? What are the biological mechanisms involved in brain functioning? These form just a sample of some of the most challenging questions in science. Brains are especially good at performing functions like pattern recognition, (motor) control, perception, flexible inference, intuition, and guessing. But brains are also slow, imprecise, make erroneous generalizations, are prejudiced, and are incapable of explaining their own actions.

Neurocomputing, sometimes called *brain-like computation* or *neurocomputation*, but most often referred to as *artificial neural networks* (ANN)[1], can be defined as information processing systems (computing devices) designed with inspiration taken from the nervous system, more specifically the brain, and with particular emphasis on problem solving. S. Haykin (1999) provides the following definition:

> "A[n artificial] neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use." (Haykin, 1999; p. 2)

Many other definitions are available, such as

> "A[n artificial] neural network is a circuit composed of a very large number of simple processing elements that are neurally based." (Nigrin, 1993; p. 11)

> "... neurocomputing is the technological discipline concerned with parallel, distributed, adaptive information processing systems that develop infor-

[1] Although neurocomputing can be viewed as a field of research dedicated to the *design* of brain-like computers, this chapter uses the word neurocomputing as a synonym to artificial neural networks.