Chapter 3
Overview of Evolutionary Algorithms

It is not the strongest or the most intelligent who will survive but those who can best manage change.
Charles Darwin

3.1 Goals of This Chapter

Every aspect of our daily life implies a search for the best possible action and the optimal choice for that act. Most standard optimization methods require the fulfilment of certain constraints, imply convergence issues or use single point movement. Among them, EAs represent a flexible and adaptable alternative, with classes of methods based on principles of evolution and heredity, with populations of potential solutions and only some basic knowledge of mathematics.

This chapter explains the concepts revolving around the field of evolutionary computation (EC) (Sect. 3.2), presents a general scheme of an EA and describes its main components (Sect. 3.3). The choices for representation and evolutionary operators are only summarized; more emphasis is given to the types employed further on within the approaches that are put forward by this book. Finally, Sect. 3.11 outlines the traditional EA approaches to classification as a standard reference for the envisaged contributions.

3.2 The Wheels of Artificial Evolution

Evolution is the driving force behind all life forms in nature, since, even under very cruel conditions, diverse species that make up our world manage to survive and adapt in their own niches. As nature represents a great source of inspiration for scientists, the development of an artificial optimization framework to mimic evolution and heredity and transform their laws into arithmetical operators came natural and it materialized through the field of EAs.

When solving a problem in the EC context, an environment is created and filled with a population of individuals. These represent a collection of candidate solutions for the considered task. Learning is viewed as a process of continuous adaptation of the individuals to the initially unknown environment. This acclimatization is achieved through reproduction, recombination and mutation. The fitness of the individuals is closely related to how well they adjust to the environment and represents their chance of survival and multiplication.

Provided that the environment can only host a limited number of individuals and given their capacity to reproduce, selection is inevitable if population size is forbidden to grow exponentially. Obviously, natural selection favors those individuals that best adapt to the environmental settings – historically called by Darwin as *survival of the fittest* – hence the best ones survive and reproduce and evolution progresses step by step. Occasional mutations take place in order to introduce new testing material. Thus, the constitution of the population changes as time passes and it evolves, offering in the end the most adequate solution(s) for the considered problem.

As a correspondence between natural evolution and problem solving, the environment is said to represent the problem, individuals are candidate solutions and the fitness corresponds to the quality of the solution. The quality of a candidate determines the chance of the considered individual to be used as a seed for building new potential solutions. As in nature, mutation also sporadically attacks certain traits of an individual. The next generation is presumably a better one than that of its parents.

There are many approaches that simulate evolution: genetic algorithms (GAs), evolution strategies, genetic programming, evolutionary programming [Fogel, 1995], [Bäck, 1996], [Michalewicz, 1996], [Bäck et al, 1997], [Dumitrescu et al, 2000], [Sarker et al, 2002], [Eiben and Smith, 2003], [Schwefel et al, 2003]. All of them imply the use of selection of individuals in a population, reproduction, random variation and competition, which are the essences of evolution, both in nature or inside a computer [Fogel, 1997]. EAs are simple, general and fast, with several potential solutions that exist at the same time. They are semi-probabilistic techniques that combine local search - the exploitation of the best available solutions at some point - with global search - the exploration of the search space. The different properties of continuity, convexity or derivability that have been standardly required in classical optimization for an objective function are of no further concern within EAs.

Real-world tasks have immense solution search spaces and the high number of local optima hardens the process of finding the global optimum in reasonable time. Moreover, the problems can have dynamic components that can change the location of the optima in time and, therefore, the technique that is considered for solving them must adapt to the changes. Additionally, the final solution may have nonlinear constraints that have to be fulfilled (constrained problems) or may have objectives that are in conflict (multiobjective problems). EAs represent an appropriate alternative for solving such problems. They are population-based approaches, thus multiple regions of the search space can be simultaneously explored. This is especially advantageous when dealing with a multimodal search space where an EA keeps track of several optima in parallel and maintains diversity in the population of solutions, with the aim of performing a better exploration of the search space for finding the global optimum. When the considered problem is dynamic, the solutions in the population continuously adapt to the changing landscape and move towards the new optima. For multiobjective problems [Coello et al, 2007], [Branke et al, 2008], EAs provide, in the end of the evolution process, a set of trade-off solutions for the conflicting objectives, while traditional techniques only produce one solution at the end of a run. For constrained problems, EAs offer a set of feasible and unfeasible solutions. Probably their main advantage and the reason why they are frequently used nowadays is that they can be applied to any type of optimization problem, be that it is continuous or discrete. Another important advantage is that they can be easily hybridized with existing techniques.

It may be tempting to perceive EAs as the applicable solver for any optimization problem, which is false.

> If there is already a traditional method that solves a given problem, EAs should not be used [Schwefel, 1997].

At least one should not expect their alternative solving to be both better or less computationally expensive. The computational effort indeed represents an important drawback of EAs as many candidate solutions have to be evaluated in the evolutionary process. But even if EAs alone do not necessarily provide the best possible solution for the problem at hand, they can be used for adding further improvements to solutions obtained by other means. Artificial evolution thus represents an optimization process that does not reach perfection, but still can obtain highly precise solutions to a large scale of optimization problems. Conversely, an EC technique is better than a random search strategy.

There is a continuous interest for researchers in this field as well as from completely different areas towards the application of EAs for solving practical optimization problems. They are also called *the Swiss army knife* of metaheuristics and they owe their charm to their simplicity and flexibility.

3.3 What's What in Evolutionary Algorithms

Artificial evolution performs a precise algorithmic cycle which closely follows its natural counterpart. Figure 3.1 intuitively shows a typical evolutionary flow. Given a population of individuals, the environmental pressure causes natural selection, the survival of the fittest, and consequently the average fitness along the generations gradually increases. Supposing the fitness function has to be maximized, a set of randomly generated candidate solutions are created in the domain of the function and are evaluated – the better individuals are considered those with higher values for the fitness function. Based on the computed fitness values, a part of the individuals are selected to be the parents of a new generation of individuals. Descendants are obtained through variation, i.e., by applying recombination and/or mutation to the previously chosen individuals. Recombination takes place between two or more individuals and one or more descendants (or offspring) are obtained; descendants borrow particularities from each of the parents. When mutation is applied to a potential solution, the result is one new individual that is usually only slightly different from its parent. After applying the variation operators, a set of new individuals is obtained that will fight for survival with the old ones for a place in the next generation. The candidate solutions that are fitter are again advantaged in this competition. The evolutionary process – parent selection - variation - survival selection – resumes and usually stops after a predefined computational limit is reached.

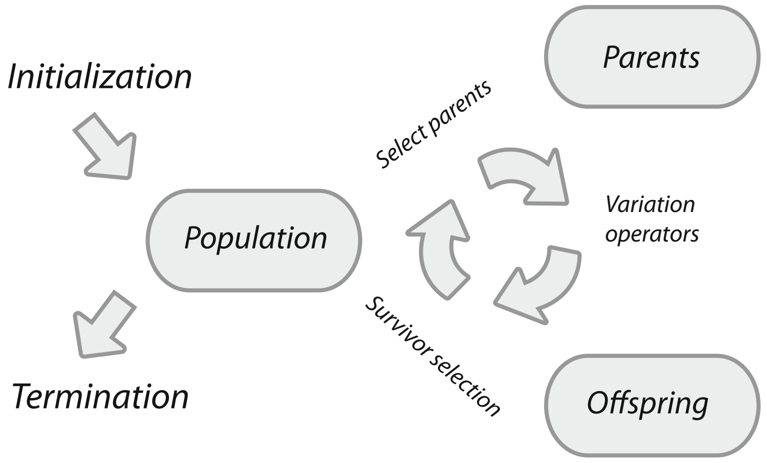

Fig. 3.1 The cycle of an EA

We have introduced several EA-related concepts in the previous paragraph. They are summarized below:

1. Initialization
2. Representation of candidate solutions
3. Population model
4. Fitness function
5. Selection
   - Parent selection strategy
   - Survival selection strategy
6. Variation operators
7. Termination criterion

An EA for a specific problem is completely built only when each of these components is carefully specified.

A general formulation of a canonical EA is outlined by Algorithm 3.1.

Although initialization and a termination criterion are also common for the traditional approaches to optimization, the presence of selection and variation operators needs further argumentation apart from that of perfectly mimicking nature. Variation operators have the role of introducing new candidates into the population and, in this way, explore the search space. But by applying them alone, worse solutions might be encountered; the task of avoiding the decrease in quality of the populations with respect to the fitness evaluations is achieved by selection. Selection exploits the fitter candidate solutions, so the average value of the population fitness consequently increases. In conclusion, variation operators create diversity in the population, having an explorative role, while selection favors the better individuals, having therefore an exploitative task.

Algorithm 3.1 A standard EA.

```
Require: An optimization problem
Ensure: The best obtained individual(s)
  begin
  Initialization;
  Evaluation;
  while termination condition is not satisfied do
     Selection for reproduction;
     Recombination;
     Mutation;
     Evaluation;
     Survivor selection for the next generation;
  end while
  return best obtained solution(s)
  end
```

In order to reach the optimum of the problem to be solved, a good equilibrium between exploration and exploitation has to be established [Eiben and Smith, 2003]. When performing too much exploitation and only little exploration, some promising regions from the search space might remain unexplored, so the best solutions might not be found at all. In such a situation, there is a high probability that the search process remains blocked into a local optimum. Otherwise, if there is too much exploration and only little exploitation, the search process is significantly slowed down and the time needed for the convergence of the algorithm to the optimum might become too large.

It must also be underlined that EAs are stochastic optimizers. When selection is applied, the fitter individuals have higher chances to be chosen than the less fit ones. However, even the weak individuals, with respect to the fitness evaluation, have a (smaller) chance to be selected. In the same way, when recombination is applied to two or more candidates, the offspring genes are randomly chosen from each of the parents. In the case of mutation, the parts of the individual that are changed are also picked in a random fashion.

In the following sections, we will present the evolutionary components in detail. The most common choices are briefly discussed and those used in further experiments throughout this book are emphasized.

3.4 Representation

The first thing to do when solving a problem through EAs is to set up a bridge between the problem space and the EA space. The possible solutions of the original problem (phenotypes) have to be encoded into individuals within the EA (genotypes).

While sometimes the phenotypic and the genotypic spaces may coincide, other times they can be completely different. For instance, if the optimization problem is in an integer domain, a real-valued genotype representation can be chosen, so the two spaces would be the same, or it could be decided for a binary representation. A solution of the form ”21” from the phenotypic space would then be represented as ”10101” in the genotypic space. The way the individual representation is chosen depends very much on the problem to be solved. In conclusion, a mapping is therefore necessary to transfer the solution of the problem into the EA space (encoding) and an inverse mapping will be also needed, in order to transform the EA result back into the problem solution (decoding).

Genotypes or individuals are also called chromosomes, but throughout the book we will mostly refer to them as individuals. The individual is composed of genes. A gene is located at a particular position in the chromosome. A gene may contain several values or may have several forms. Each value of a gene is referred to as an allele of that gene.

A binary representation is used for problems searching for 0/1 values (such as truth values) for certain variables. It is however often inappropriately chosen for solving other tasks, just because it is typical of the widely used EA subclass of GAs.

An integer encoding is most appropriate when solutions have to be represented in a discrete space. An example would be the search for optimal values for a set of variables that are instantiated with integers. Permutation problems where one must determine an optimal permutation of elements that lead to a potential solution can also be represented by integer values.

When the values represented by genes come from a continuous space, a real-valued encoding is employed. Every individual is then a vector of real components and each of them takes values in a certain domain and must be kept in that interval all throughout evolution.

The chosen representation is strongly related to the type of recombination and mutation operators that are used. Since the techniques we discuss in this book use only binary and real-valued encodings, we will summarize only some variation operators that correspond to these representations.

3.5 The Population Model

The role of the population is that of preserving all candidate solutions at one time; it consists of a set of genotypes that are not necessarily all different from each other. The population as a unit is the one that evolves, not the individuals. When selection is applied for choosing the parents of the population that will form the next generation, it picks the fitter candidates from the current population. Also, the resulting offspring are intended to replace the former individuals in a fitter future population.

The population size represents the number of individuals that the population contains. Usually, the population size, which is a parameter of the EA, is constant from the start to the end of the algorithm. However, an algorithm that contains a population with decreasing size is presented in Chap. 4.

The initialization of the population implies addressing population size together with representation. Each gene of every individual generally takes a random value from its domain of representation. Sometimes, the EA may start using as an initial population a fixed set of candidate solutions obtained by other methods.

3.6 Fitness Evaluation

The role of the fitness function (or evaluation function) is to measure the extent by which individuals adapted to the environment. It is mostly used by selection and thereby makes improvements possible.

The fitness function is defined on the genotypic space and its values are usually real numbers, in order to be able to make comparisons between the qualities of different individuals. Returning to the previous example of a binary representation, and presuming that a real-valued function is to be optimized, e.g. $f(x) = x^2$, the fitness of the genotype 10101 is $21^2 = 441$.

In many cases, the objective function, which is the name used in the original context of the problem, coincides with the fitness function or the fitness function is a transformation of the given objective function.

3.7 The Selection Operator

Selection appears twice during an evolutionary cycle. First, there is selection for reproduction, when parents of the next generation are chosen (parent or mating selection). Secondly, there is selection for replacement, when individuals that will form the next generation are chosen from the offspring and the current population (survivor selection). Both selection types are responsible for quality enhancement.

3.7.1 Selection for Reproduction

The role of parent selection is that of choosing which of the individuals in the current population should be considered to undergo variation in order to create offspring, based on their quality. Parent selection is typically probabilistic: high-quality individuals have a good chance to be selected for reproduction, while low-quality ones have small chances of becoming parents.

The selection operator does not create new candidate solutions. It is solely responsible for selecting relatively good solutions from the population and discarding the remaining candidates. As the population size usually remains constant, the selection conducts to the placement of multiple copies of certain individuals in a new population by removing inferior solutions.

The basic idea is that individuals with a better fitness must have a higher probability of being selected. Nevertheless, selection operators differ in the way the chances are assigned to better solutions. Some operators sort the individuals in the population according to their fitness and then deterministically choose some few best individuals. Meanwhile, other operators assign a selection probability to each individual which is dependent on its fitness. In this case, there exists the possibility of selecting a bad solution and, at the same time, of rejecting a good one. However, this can be an advantage: the fittest individuals in a population can be connected to a suboptimal region in the fitness landscape and, by using a deterministic selection, the EA would evidently converge to the wrong, suboptimal solution. If, conversely, a probabilistic selection is employed here, diversity is maintained for a higher number of generations by selecting some less fit individuals. Therefore, more exploration is performed and the EA would eventually be prevented from converging to a wrong solution.

Two of the most popular schemes, i.e. the proportional and the tournament selections, are briefly mentioned in the following paragraphs.

Proportional selection implies that the number of copies an individual will have is directly proportional to its fitness. A solution having double the fitness of another solution will also have twice as many copies in the selected population. The most commonly used form of implementing selection probabilities within the proportional type is the roulette-wheel (or Monte-Carlo) mechanism, where each individual in the population occupies a section on a roulette that has a size directly proportional to its fitness. Then, the wheel is spun as many times as the population size and at every turn the solution indicated by the wheel is selected. Evidently, solutions with better fitness have higher chances to be selected (and therefore to have several copies in the population) as their sections on the wheel are proportional to their fitness.

Algorithm 3.2 puts forward the basic steps for implementing the proportional scheme. A probability selection is computed for each individual like in (3.1), by referring the sum of the fitness evaluations for all individuals in the population. Then, the sections of the roulette-wheel are computed as in (3.2). Next, the wheel is turned n times by randomly generating a number in the [0, 1] interval and determining its correspondingly chosen section: that is the one pointing to the individual that is selected.

$$P_{sel}(x_i) = \frac{f(x_i)}{\sum_{j=1}^{n} f(x_j)} \tag{3.1}$$

$$a_i = \sum_{j=1}^{i} P_{sel}(x_j) \tag{3.2}$$

There are however several limitations of the proportional selection scheme. First of all, if there exists a very fit individual in comparison to all the others in the population, proportional selection selects a very high number of copies of that individual and this leads to a loss of diversity in the population which conducts to premature convergence. On the other hand, if all candidates have very similar evaluations, which usually happens later on in the evolutionary process, the roulette-wheel will be marked approximately equally for all individuals in the population and all of

Algorithm 3.2 Proportional selection.

```
Require: The population that consists of $n$ individuals
Ensure: $n$ individuals selected for reproduction
  begin
  for $i = 1$ to $n$ do
    Compute the probability $P_{sel}(x_i)$ to select individual $x_i$ as in (3.1);
  end for
  for $i = 1$ to $n$ do
    Compute the roulette section $a_i$ for individual $x_i$ as in (3.2);
  end for
  $i \leftarrow 1$;
  while $i \leq n$ do
    Generate a random number $r \in [0, 1]$;
    $j \leftarrow 1$;
    while $a_j < r$ do
      $j \leftarrow j + 1$;
    end while
    $select_i \leftarrow x_j$;
    $i \leftarrow i + 1$;
  end while
  return the selected individuals
  end
```

them will have almost the same chances of being selected (the effect of random selection). On a different level, it cannot handle negative values for fitness, as they should correspond to sections on the roulette-wheel. Finally, it cannot handle minimization problems directly, but they have to be transformed into a maximization formulation. A way to avoid these last two drawbacks is by using a scaling scheme, where the fitness of every solution is mapped into another interval before marking the roulette wheel [Goldberg, 1989].

These issues of proportional selection are avoided when using the tournament type. For a number of times equal to the population size, one chooses the best solution in a tournament of k individuals. In the simplest form, k is 2, the scheme is called binary tournament selection and the best one of each two solutions is chosen for as many times as to form a population of the same size as before. This selection operator does not depend on whether the fitness values are positive or negative and, when one deals with a minimization problem (instead of a maximization one), the only difference is that the individuals with the smaller fitness value are now selected (instead of the ones with the higher fitness score). The absolute performances of individuals do not count, it is only the actual values they exhibit in relation to one another that are important. Therefore, in this type of selection there is no need for global knowledge on the population as in the proportional scheme.

Due to its simplicity and flexibility, it is the scheme we use the most throughout the book and it is outlined in Algorithm 3.3.

Algorithm 3.3 Tournament selection.

```
Require: The population that consists of $n$ individuals
Ensure: $n$ individuals selected for reproduction
  begin
  $i \leftarrow 1$;
  while $i \leq n$ do
     Choose $k$ individuals;
     Take the fittest one $x$ from them;
     $select_i \leftarrow x$;
     $i \leftarrow i + 1$;
  end while
  return the selected individuals
  end
```

An important parameter and advantage of this selection is given by the tournament size k. For higher values of k, there is a stronger selection pressure, which triggers the choice of above average individuals. Smaller values of k on the contrary also give weaker individuals the chance of being selected, which eventually leads to a better exploration of the search space.

Ranking selection is similar to the proportional scheme, but instead of using the direct fitness values, the individuals are ordered according to their performance and attributed a corresponding rank. It can also treat negative evaluation results and, when minimization is required, the only difference is that ranking has to be inversely performed.

3.7.2 Selection for Replacement

Another situation when selection takes place is when it is decided which individuals from the current population and their offspring are retained to form the population of the next generation. In order to preserve the same population size after offspring are obtained via the variation operators, survivor selection has to intervene. This selection decision is usually elitist (the best individuals are preferred) and takes into account the fitness of all individuals (new and old), favoring the fitter candidate solutions.

3.8 Variation: The Recombination Operator

We have seen that the selection operator has the task of focusing search on the most promising regions of the search space. On the other side, the role of the variation operators is to create new candidate solutions from the old ones and increase population diversity. They are representation dependent, as for various encodings, different operators have to be defined [Bäck, 1996], [Bäck et al, 1997], [Eiben and Smith, 2003].

We will thus further discuss the standard operators for introducing variation, i.e., recombination and mutation, along with their most common representation-related choices. Those of particular need for the upcoming approaches of this book are described at larger extent.

Recombination or crossover is a variation operator that is responsible for forming offspring by combining the genes of parent individuals. Recombination represents a stochastic operator, since choices like which parts are to be inherited from one parent and which from the other or the way the two parts are combined depend on a pseudo-random number generator.

When mating (usually two) individuals with different attributes, an offspring (or two) that combine those features is (are) obtained. The aim is to explore the space between the two individuals, in search of a potential solution that has a better quality.

In a more general scenario, recombination can even take place between p individuals ($p > 2$) and p offspring may be obtained [Bäck et al, 1997]; one offspring alone can also be constructed by combining traits from the p parents.

The process usually takes place as follows. For every individual in the current population obtained after the application of the selection for reproduction, a random number in the [0, 1] interval is generated. If it is smaller than the given crossover probability, then the current individual is chosen for recombination. If the number of

Algorithm 3.4 The main steps of a recombination process using probability p_r.

```
Require: A population that consists of $n$ individuals
Ensure: A population obtained after the recombination process
  begin
  $i \leftarrow 0$;
  for $j = 1$ to $n$ do
     Generate $q$ in [0,1]; {select individuals for recombination in vector parent with probability $p_r$}
     if $q < p_r$ then
        $i \leftarrow i + 1$;
        $parent[i] \leftarrow individual[j]$;
     end if
  end for
  if $i$ is odd then
     Generate $q$ in [0,1]; {pairs of parents should be formed, so either add or remove an individual}
     if $q < 0.5$ then
        Add a random individual from population to parent;
     else
        Remove a random item from parent;
     end if
  end if
  Apply the chosen recombination type for each pair of parents;
  return the resulting population
  end
```

chosen individuals is odd, then they recombine as pairs which are randomly formed. Otherwise, one individual is deleted or another one is added from the parents pool, a decision which is also randomly taken. The steps are presented in Algorithm 3.4.

As concerns the different possibilities of recombination, several options can be distinguished, depending on the type of representation – binary, integer or real-valued. The choices are numerous, however only the most common schemes within the binary and real-valued representations are further outlined [Eiben and Smith, 2003], as some of them will be addressed in the algorithms of this book.

For the binary representation, an often used scheme is the one point recombination. A random position is generated which is the point of split for the two parents, as in Fig. 3.2. The resulting offspring take the first part from one parent and the other from the other, respectively. Suppose there are two parent individuals $c = (c_1, c_2, ..., c_m)$ and $d = (d_1, d_2, ..., d_m)$ and we take a random number k from the set $\{1, 2, ..., m\}$. The first offspring copies the first k genes from parent c and the remaining from d, like this: $o_1 = (c_1, c_2, ..., c_k, d_{k+1}, ..., d_m)$; conversely, the second offspring takes the first k genes from parent d and the remaining from c, i.e. $o_2 = (d_1, d_2, ..., d_k, c_{k+1}, ..., c_m)$.

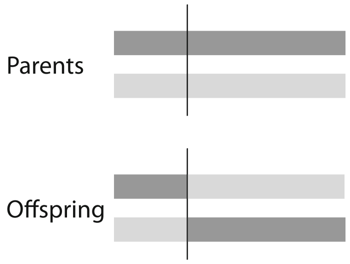

Fig. 3.2 One point crossover for binary representation. The first part of offspring 1 is copied from the first parent and the other section from the second parent, while for the second offspring the complementary parts are inherited from the two parents.

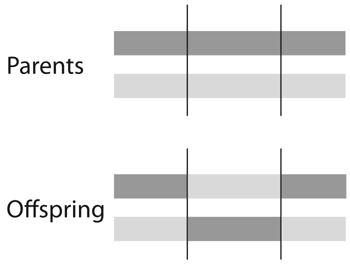

Fig. 3.3 Two point crossover for binary representation. The offspring take alternate sections from the parents. The same type of inheritance occurs when several cut points are considered.

A generalized form is the multiple point recombination, where several cut points are considered: an example of two point crossover can be visualized in Fig. 3.3. Furthermore, within adaptive recombination, the split points also undergo evolution by adapting to previous splits that took place. Additionally, segmented recombination is another variant of the multiple point recombination where the number of points can vary from one individual to another [Eiben and Smith, 2003].

Uniform recombination does not use split points. For every gene of the first offspring it is probabilistically decided which parent gives the value of that component, while the corresponding gene of the second offspring gets the value from the other parent. This operator could also be considered such that the values for both offspring individuals are computed in the same probabilistic manner, but independently.

Shuffle recombination is an add-on to an arbitrary binary recombination scheme and has the advantage of removing positional bias. The genes of the two parents are shuffled randomly, remembering their initial positions. The resulting individuals may then undergo any kind of binary crossover. Resulting offspring are un-shuffled.

Such recombination mechanisms are obviously not suitable for a real-valued encoding. Intermediate recombination presumes that the value for the gene of the offspring is a convex combination of the corresponding values of the parents. We have again two parents $c = (c_1, c_2, ..., c_m)$ and $d = (d_1, d_2, ..., d_m)$ and offspring gene $o_i = \alpha c_i + (1-\alpha)d_i$, for $\alpha \in [0,1]$. Parameter α can be randomly chosen at each application of the recombination operator or it can be fixed: it is very often set at 0.5, which leads to a uniform arithmetic recombination. Depending on the number of recombined genes for the offspring, there are three types of intermediate recombination: single (when one position is changed), simple (when we change all values from some point on) and total (when all genes are affected).

3.9 Variation: The Mutation Operator

Mutation is a unary variation operator. It is also a stochastic one, so the genes whose values are considered to be changed are chosen in a probabilistic manner. When applied to an individual, the resulting offspring contains minor modifications as opposed to the initial individual. Through mutation, individuals that cannot be generated using recombination may be introduced into the population, as it makes all the values of a gene available for the search process.

For every individual in the current population and each gene of that individual, a random number in the [0, 1] interval is generated. If the mutation probability is higher than the generated number, then that gene suffers mutation (see Algorithm 3.5).

Sometimes global search is intended in the initial evolutionary phases and local search (fine tuning) is planned towards the final steps of the EA. In this respect, the mutation probability may decrease with the increase in the number of generations. Another distinct situation concerns those cases where the change of a gene lying at the beginning of an individual could make a significant modification to the individual in question, while the same change at the end of the individual would induce a less significant alteration. Then, while the number of generations passes, the probability of mutation of the first genes in every individual may decrease and that of the final ones increase.

Algorithm 3.5 Mutation applied with probability p_m.

```
Require: A population that consists of $n$ individuals, each containing $m$ genes
Ensure: A population obtained after the mutation process
  begin
  for $i$ = 1 to $n$ do
    for $j$ = 1 to $m$ do
      Generate $q$ in [0,1];
      if $q < p_m$ then
        Apply the chosen mutation operator to the gene $j$ of the current individual $i$;
      end if
    end for
  end for
  return the modified population
  end
```

Depending on the specific task and the considered representation, there are several forms of the mutation operator as well. Again only the situations with binary and real-valued encodings are further outlined and the most frequent types are described [Eiben and Smith, 2003].

For the binary encoding, a strong mutation (also called bit flip perturbation) presumes that, when a gene undergoes mutation, 1 changes into 0 and 0 into 1. Thus, the value of the gene to be mutated will change through the formula $c_i = 1 - c_i$. An intuitive representation can be observed in Fig. 3.4.

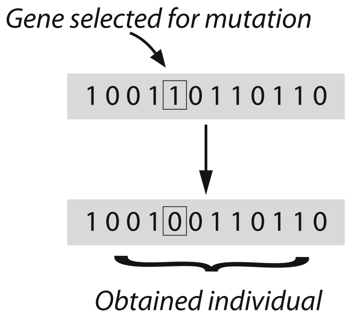

Fig. 3.4 Mutation for binary representation. A position having the value $v \in \{0, 1\}$ is randomly chosen in the individual to be mutated and its value is changed to 1 - v.

Weak mutation presumes that the above change does not take place automatically as before, but 1 or 0 is probabilistically chosen and attributed to that position. In this way, the newly generated value could be identical to the old one, so no effective change would actually occur.

For the real-valued individuals, mutation customarily performs a small perturbation in the value of the selected gene. This is induced randomly by a number generated to follow a normal distribution with mean zero and standard deviation given by a parameter called mutation strength. A value of the gene i of an individual c is thus changed according to the formula $c_i = c_i + N(0, ms)$, where ms is the mutation strength parameter.

3.10 Termination Criterion

The stop condition of a typical EA may refer to several criteria like:

- reaching a previously set number of generations,
- not exceeding a predefined number of iterations without achieving any fitness improvement,
- finding a solution with a given accuracy,
- consuming a preset number of fitness evaluation calls,
- allowing the algorithm to run for a certain amount of time,
- population diversity falling below a given threshold,
- a stop button etc.

The solution to the algorithm is the best individual with respect to the fitness function from the last generation, or the best from the entire process, or, sometimes, the entire (or a subset of the) population from the last generation [Bäck, 1996], [Dumitrescu et al, 2000].

3.11 Evolutionary Algorithms for Classification

As the practical side of this book targets classification, the classical evolutionary techniques that had been tailored for this direction will be mentioned. Note that, although there are EA approaches acting both as stand-alone or in hybridization with classification-specific methods, we will present only the former ones. For the latter category, we will specifically outline those related to SVMs in the future Chap. 6 and 7.

The aim of a classification technique may be further conceived as to stepwise learn a set of rules that model the training set as good as possible. When the learning stage is finished, the obtained rules are applied to previously unseen samples within the test set to predict their classes.

EAs may consequently encode IF-THEN rules, while certain mechanisms model their behavior and interaction towards learning. An evolutionary classifier then represents a machine learning system that uses an EA as a rule discovery component [Michalewicz, 1996]. The IF-THEN rules (or productions) are represented through a population that is evolved by an appropriate EA. The rules cover the space of possible inputs and are evolved in order to successfully be applied to the problem to be solved – the fields of application may range from data mining to robotics. On a broader sense, and in connection with the definition of classification from the introductory chapter, an evolutionary classification approach is concerned with the discovery of IF-THEN rules that reproduce the correspondence between given samples and corresponding classes. Given an initial set of training samples, the system learns the patterns, i.e. evolves the classification rules, which are then expected to predict the class of new examples. An IF-THEN rule is imagined as a first-order logic implication where the condition part is made of attributes and the conclusion part is represented by the class.

There are three classical families of evolutionary classifiers [Bacardit and Butz, 2007]:

The Pittsburgh learning strategy [Smith, 1980], [de Jong et al, 1994], [Michalewicz, 1996]: aims to evolve a complete classification rule collection by encoding the entire set into each individual.

The Michigan approach [Holland, 1986], [Wilson, 1995], [Michalewicz, 1996]: considers every individual as the representative of one rule, uses reinforcement learning to reward/penalize the collaboration between rules and achieves the complete optimal rule set as the output of the EA.

Iterative rule learning [Venturini, 1993], [Aguilar-Ruiz et al, 2003], [Bacardit and Butz, 2007]: for each class of the problem, one separate EA run is performed, with all individuals encoding rules for that class and adjusting themselves against the training samples labeled accordingly.

In a Pittsburgh-type evolutionary classifier, each individual represents an entire set of rules. The individuals compete among themselves and only the strong sets survive and reproduce. The Pittsburgh approach uses a typical EA for conducting the learning. What remains to be solved is the representation problem and the way individuals adapt to their environment. Usually, operators from propositional logic, like disjunction and/or conjunction, also appear within the encoding.

In a Michigan-style evolutionary classifier, each individual of the population represents a unique, distinct rule, so the EA evolves a set of rules. Then, the population represents the rule set needed to solve the problem. The goal here is not to obtain the best individual, but to find the best set of individuals (rules) in the end of the algorithm. Usually, representation is divided into two parts – one is the condition part and contains the values for the attributes that appear in the condition of the rule and the other part consists of the conclusion of the rule. A credit assignment system is used in order to reward the better rules or, at the same time, to penalize the worse ones. When new rules are produced through mutation and/or recombination, crowding methods are usually utilized in order to introduce them into the population; in this way, they replace only very similar individuals.

By iterative rule learning, the evolution of a rule for each class is obtained by an equal number of runs of an EA. In every such run, the individuals (rules of one class) are evolved against the training samples of the corresponding class. The fitness expression refers both accuracy and a generality measure that makes the individual cover as much samples as possible.

Finally, note that rules in an IF-THEN format are the traditional means of representing individuals for classification by EAs. Other more complex representations can be employed, like, for instance, a genetic programming approach to rule discovery [Giordana et al, 1994], [Freitas, 1997], [Bojarczuk et al, 2000]. The internal nodes of the individual encode mathematical functions, while the leaf nodes refer the attributes. Given a certain individual, the output of the tree is computed and, if it is greater than a given threshold, a certain outcome of the classification task is predicted.

3.12 Concluding Remarks

We have targeted to show why EAs are easy and straightforward for any task, while their performance as general optimizers is very competitive. Their power and success lie in the simplicity of their mathematical functioning, the naturalness of underlying metaphor and the easy tailoring for any given problem.

After having gone through the introductory chapter into evolutionary computing, the following knowledge has been acquired:

- A general idea of the concepts revolving around EAs.
- The components of a typical EA are subsequently enumerated and are followed by brief descriptions.
- The traditional EA applications for classification are summarized, to set the context for introducing the new evolutionary approaches for this task in Chap. 4, 5, 6 and 7.