13 Fundamental Evolutionary Algorithms

The preceding chapter presented all relevant elements of evolutionary algorithms, namely guidelines of how to choose an encoding for the solution candidates, procedures how to select individuals based on their fitness, and genetic operators with which modified solution candidates can be obtained. Equipped with these ingredients, we proceed in this chapter to introducing basic forms of evolutionary algorithms, including classical genetic algorithms (in which solution candidates are encoded as bit strings, see Sect. 13.1), evolution strategies (which focus on numerical optimization, see Sect. 13.2) and genetic programming (which tries to derive function expressions or even (simple) program structures with evolutionary principles, see Sect. 13.3). Finally, we take a look at related population-based approaches (like ant colony and particle swarm optimization, see Chap. 14).

13.1 Genetic Algorithms

In nature, all genetic information is described in an essentially quaternary code, based on the four *nucleotides* adenine, cytosine, guanine and thymine, which are stringed together in a DNA sequence on a backbone of phosphate-deoxyribose (so-called *primary structure* of a nucleic acid). Although there are also higher level structures (like the fact that the nucleotides are organized in so-called *codons*, which are triplets of nucleotides), this is the basis of the genetic code. Transferring this structure to computer science, it seems natural to base all encoding on the ultimately binary structure of information in a computer. That is, we use chromosomes that are bit strings. This is the distinctive feature of so-called **genetic algorithms** (GA).

In principle, of course, all evolutionary algorithms running on a computer can be seen as genetic algorithms in this sense, simply because a computer ultimately encodes all information in bits, that is, in zeros and ones. For example, the chromosomes we considered for the n-queens problem, which are arrays of integers,

R. Kruse et al., *Computational Intelligence*, Texts in Computer Science,
DOI 10.1007/978-1-4471-7296-3_13

are stored using a binary representation of these numbers. To be concrete, the chromosome (4, 2, 0, 6, 1, 7, 5, 3), which describes a solution of the 8-queens problem, could be stored (using 3 bits for each number) as 100 010 000 110 001 111 101 011. The difference between a genetic algorithm and the evolutionary algorithm we considered for the n-queens problem consists in the fact that in a genetic algorithm we consider merely the bit string that a chromosome is and ignore any higher level structure. (In the above concrete example: the fact that bit triplets are interpreted together, each indicating a file (column) in which a queen is placed.) That is, we completely separate the encoding from the genetic mechanisms, while in an evolutionary algorithm certain aspects of the encoding are considered, for instance, to restrict the genetic operators. (In the above example: in the evolutionary algorithm for the 8-queens problem, we allow as cut points for a crossover operator only the points between bit triplets, because these are the interpretable information units, while in a genetic algorithm we allow cuts between arbitrary bits.)

A typical genetic algorithm works like this:

Algorithm 13.1 (*Genetic Algorithm*)

```
function genalg ($f$: function, $\mu$: int, $p_x$: real, $p_m$: real) : array of bit;
begin
  $t \leftarrow 0$;                                  (* initialize the generation counter *)
  pop($t$) $\leftarrow$ create a population of $\mu$ random bit strings; (* $\mu$ must be even *)
  evaluate pop($t$) with the fitness function $f$;        (* compute initial fitness *)
  while termination criterion is not fulfilled do begin
    $t \leftarrow t + 1$;                            (* count the created generation *)
    pop($t$) $\leftarrow \emptyset$;                 (* build the next generation *)
    pop′ $\leftarrow$ select $\mu$ individuals $\mathbf{s}_1, \ldots, \mathbf{s}_\mu$ from pop($t$)
             with roulette wheel selection;
    for $i \leftarrow 1, \ldots, \mu/2$ do begin     (* process individuals in pairs *)
      $u \leftarrow$ choose random number from $U([0, 1))$;
      if $u \leq p_x$ then crossover_1point($\mathbf{s}_{2i-1}, \mathbf{s}_{2i}$); end
      mutate_bits($\mathbf{s}_{2i-1}$, $p_m$);        (* crossover rate $p_x$ and *)
      mutate_bits($\mathbf{s}_{2i}$,   $p_m$);        (* mutation rate $p_m$ *)
      pop($t$) $\leftarrow$ pop($t$) $\cup \{\mathbf{s}_{2i-1}, \mathbf{s}_{2i}\}$;  (* add (modified) individuals *)
    end                                              (* to the next generation *)
    evaluate pop($t$) with the fitness function $f$;
  end                                                (* compute new fitness *)
  return best individual from pop($t$);              (* return the solution *)
end
```

That is, a genetic algorithm follows essentially the scheme of a general evolutionary algorithm (see Algorithm 11.1 on p. 197), only that it uses bit strings as its chromosomes and applies genetic operators to these chromosomes that ignore any higher level structure of the encoding. A genetic algorithm requires mainly three parameters: the population size μ, for which an even number is chosen in order to simplify the implementation of a random application of the crossover operator, the crossover probability p_x, with which it is decided for a pair of chromosomes whether they undergo crossover or not, and the mutation probability p_m, with which it is decided for each bit of a chromosome whether it is flipped or not.

13.1.1 The Schema Theorem

So far, we answered the question why evolutionary algorithms work only by providing plausibility and intuition-based arguments. Due to their restriction to bit strings, genetic algorithms, however, allow for a more formal investigation, which was first proposed in Holland (1975). This leads to the famous **schema theorem**.

Since genetic algorithms work on bit strings only, they allow us to confine our considerations to binary chromosomes. More precisely, we consider **schemata**, that is, partly specified binary chromosomes. We then investigate how the number of chromosomes matching a schema evolve over several generations of a genetic algorithm. The objective of this investigation is to derive a rough stochastic statement that describes how a genetic algorithm explores the search space.

In order to keep things simple, we confine ourselves (following Holland 1975) to bit strings of a fixed length L. In addition, we generally assume the specific form of a genetic algorithm as it was presented in pseudocode in the preceding section. That is, we assume that chromosomes enter the intermediate population pop′ by fitness-proportionate selection (to be precise, by roulette-wheel selection as it was introduced in Sect. 12.2.1) and that one-point crossover (see Sect. 12.3.2), applied with probability p_x to chromosome pairs, and bit mutation (see Sect. 12.3.1), using the mutation probability p_m, are employed as the genetic operators.

We start with the necessary technical definitions of *schema* and *matching*.

Definition 13.1 (*Schema*) A *schema* h is a character string of length L over the alphabet $\{0, 1, *\}$, that is, $h \in \{0, 1, *\}^L$. The character $*$ is called *wildcard character* or *don't-care symbol*.

Definition 13.2 (*Matching*) A (binary) chromosome $c \in \{0, 1\}^L$ *matches* a schema $h \in \{0, 1, *\}^L$, written as $c \lhd h$, if and only if it coincides with h at all positions where h is 0 or 1. Positions at which h is $*$ are not taken into account (which explains the names "don't care symbol" or "wildcard character" for the character $*$).

As an illustration consider the following simple example: let h be a schema of length $L = 10$ and c_1, c_2 two different chromosomes of this length, which look like this:

$$h = \texttt{**0*11*10*}$$
$$c_1 = \texttt{1100111100}$$
$$c_2 = \texttt{1111111111}$$

Clearly, chromosome c_1 matches schema h, i.e., $c_1 \lhd h$, because c_1 differs from h only at positions where h is $*$. On the other hand, chromosome c_2 does not match h, i.e., $c_2 \ntrianglelefteq h$, because c_2 contains 1 s at positions where h is 0 (positions 3 and 9).

Generally, there are 2^L possible chromosomes and 3^L different schemata. Every chromosome matches $\sum_{i=0}^{L} \binom{L}{i} = 2^L$ schemata (because any choice of i positions, at which the bit in the chromosome is replaced with a $*$, yields a schema that the chromosome matches). Based on this fact, Holland (1975) started from the idea that the observation of one chromosome corresponds to the observation of many schemata at the same time. This is what Holland called **implicit parallelism**.

Note that a population of size μ can, in principle, match close to $\mu 2^L$ different schemata. However, the number of actually matched schemata is usually a lot smaller, especially in later generations of a genetic algorithm, because the selective pressure produces similar chromosomes. (Assuming that similar chromosomes have similar fitness, selecting individuals with a high fitness has a tendency to select similar chromosomes, cf. Fig. 12.7 on p. 224 as an illustration.)

In order to understand schemata better, let us take a look at two interpretations. Geometrically, a schema can be seen as describing a hyperplane in a unit hypercube. Not general hyperplanes, though, but only hyperplanes that are parallel or orthogonal to the sides of the hypercube. This is illustrated in Fig. 13.1 for three dimensions: the schema `*00` describes the edge connecting the corners `000` and `100` (bottom front). The left face of the cube is captured by the schema `0**`. The schema `***`, which consists of wildcard characters only, describes the whole cube.

An alternative interpretation considers the domain of the fitness function. Suppose we are given a unary fitness function $f : [0, 1] \to \mathbb{R}$ and that the argument of this function is encoded as a binary number (in the usual way—for reasons of simplicity, we disregard here that such a code introduces **Hamming cliffs** (see Sect. 12.1.1), which can be avoided by using a Gray code). In this case a schema corresponds to a "strip pattern" in the domain of the function f. This is illustrated in Fig. 13.2 for the two schemata `0**...*` (left diagram) and `**1*...*` (right diagram).

In order to carry out our plan of tracking the evolution of chromosomes that match a schema, we have to examine how selection and applying genetic operators (that is, one-point crossover and bit mutation) influence these chromosomes. We do so in three steps. In the first step, we consider the effect of selection, in the second step the effect of one-point crossover, and in the third step the effect of bit mutation. We distinguish the populations in these steps (and quantities referring to them) by splitting

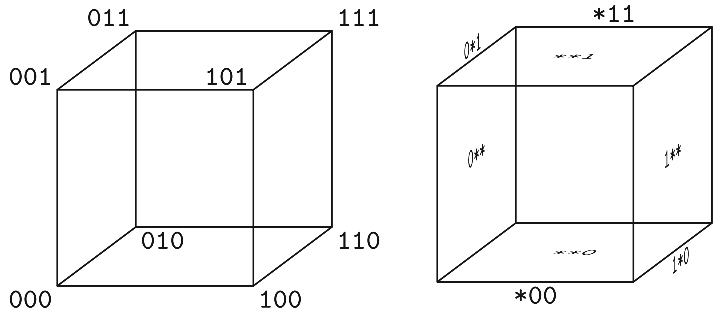

Fig. 13.1 Geometric representation of schemata as hyperplanes in a unit hypercube

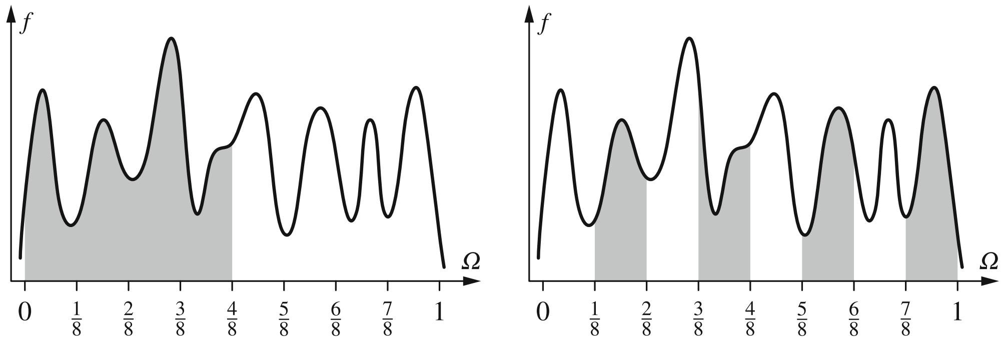

Fig. 13.2 Representation of schemata as domain of a function. On the *left* the schema `0**...*` is shown, on the *right* the schema `**1*...*`

the transition from time t to time $t+1$ into the steps t (original population), $t+\Delta t_s$ (population after selection, that is, intermediate population pop′), $t+\Delta t_s+\Delta t_x$ (population after selection and crossover), $t+\Delta t_s+\Delta t_x+\Delta t_m = t+1$ (population after selection, crossover and mutation, which is identical to the new population at time $t+1$). In these steps, we are mainly interested in the (expected) number of chromosomes that match a schema h. We denote these numbers by $N(h, t')$, where $t' \in \{t, t+\Delta t_s, t+\Delta t_s+\Delta t_x, t+1\}$. Our objective is to derive a (stochastic) relationship between $N(h,t)$ and $N(h,t+1)$.

In order to capture the effect of selection, we have to consider what fitness the chromosomes have that match a schema h. With fitness proportionate selection, the expected number of offspring of a chromosome s is $\mu \cdot f_{\text{rel}}(s)$. Hence the expected number of chromosomes that match schema h after selection is

$$N(h, t+\Delta t_s) = \sum_{s\in\text{pop}(t), s\lhd h} \mu \cdot f_{\text{rel}}^{(t)}(s).$$

In order to express this number without referring to individual chromosomes s, it is convenient to define the *mean relative fitness* of chromosomes that match a schema.

Definition 13.3 (*Mean Relative Fitness*) The *mean relative fitness* of chromosomes that match schema h in the population pop(t) is

$$f_{\text{rel}}^{(t)}(h) = \frac{\sum_{s\in\text{pop}(t), s\lhd h} f_{\text{rel}}^{(t)}(s)}{N(h,t)}.$$

With this definition, we can write $N(h, t+\Delta t_s)$ as

$$N(h, t+\Delta t_s) = N(h,t)\cdot\mu\cdot f_{\text{rel}}^{(t)}(h).$$

By inserting the definition of relative fitness (see p. 221), we can transform the expression $\mu \cdot f_{\text{rel}}(h)$ on the right hand side according to

$$\mu\cdot f_{\text{rel}}(h) = \frac{\sum_{s\in\text{pop}(t), s\lhd h} f_{\text{rel}}(s)}{N(h,t)}\cdot\mu = \frac{\frac{\sum_{s\in\text{pop}(t), s\lhd h} f(s)}{\sum_{s\in\text{pop}(t)} f(s)}}{N(h,t)}\cdot\mu = \frac{\frac{\sum_{s\in\text{pop}(t), s\lhd h} f(s)}{N(h,t)}}{\frac{\sum_{s\in\text{pop}(t)} f(s)}{\mu}} = \frac{\overline{f_t(h)}}{\overline{f_t}}.$$

where $\overline{f_t(h)}$ is the mean fitness of the chromosomes that match schema h in generation t and $\overline{f_t}$ is the mean fitness of all chromosomes in generation t. Therefore, we can write the expected number of chromosomes that match schema h after selection as

$$N(h, t + \Delta t_s) = N(h, t) \cdot \frac{\overline{f_t(h)}}{\overline{f_t}}.$$

In order to incorporate the effect of the genetic operators, we need measures with which we can compute probabilities that the match to a schema is preserved (or destroyed). For one-point crossover, we have to consider how likely it is that the involved parent chromosomes are cut in such a way that all fixed elements of a schema under consideration are inherited from the same parent. In this case, the created offspring matches the schema if the corresponding parent does. Otherwise, the match to the schema can get lost as the following example demonstrates:

$$\begin{array}{lcl}
h = \texttt{***0*|1*1**} & & \texttt{***0*1*1**} = h \\
h \rhd c_1 = \texttt{00000|11111} & \rightarrow & \texttt{1111111111} = c_1' \not\lhd h \\
h \not\rhd c_2 = \texttt{11111|00000} & \rightarrow & \texttt{0000000000} = c_2' \not\lhd h
\end{array}$$

Chromosome c_1 matches the schema h, but c_2 does not. A one-point crossover at the point marked by | creates two offspring chromosomes, both of which do not match the schema. With a different cut point, however, offspring c_1' matches the schema

$$\begin{array}{lcl}
h = \texttt{***|0*1*1**} & & \texttt{***0*1*1**} = h \\
h \rhd c_1 = \texttt{000|0011111} & \rightarrow & \texttt{1110011111} = c_1' \lhd h \\
h \not\rhd c_2 = \texttt{111|1100000} & \rightarrow & \texttt{0001100000} = c_2' \not\lhd h
\end{array}$$

Obviously, whether an offspring chromosome matches the schema can depend crucially on the location of the cut point relative to the fixed characters of the schema. This gives rise to the notion of the *defining length* of a schema

Definition 13.4 (*Defining Length of a Schema*) The *defining length* $\mathrm{deflen}(h)$ of a schema h is the difference between the position of the last and the first non-$*$ in h.

For instance, $\mathrm{deflen}(\texttt{**0*11*10*}) = 9 - 3 = 6$, because the first character that is not a wildcard, here a 0, is at position 3 and the last character that is not a wildcard, here also a 0, is at position 9. The defining length is the difference of these numbers.

In one-point crossover, all possible cut points are equally likely. Therefore the probability that the cut point splits a chromosome in such a way that some of the fixed characters of a schema lie on one side of the cut and some on the other (and therefore are *not* all inherited from the same parent chromosome) is $\frac{\mathrm{deflen}(h)}{L-1}$, because there are $\mathrm{deflen}(h)$ cut points between the first and the last fixed character and $L - 1$ possible cut points in total (as L is the length of the chromosomes). In contrast, the probability that all fixed characters of the schema lie on the same side of the cut (and therefore *are* all inherited from the same parent, thus ensuring that the corresponding child matches the schema if the parent does) is $1 - \frac{\mathrm{deflen}(h)}{L-1}$.

In order to derive an expression for $N(h, t + \Delta t_s + \Delta t_x)$, we have to consider whether a chromosome is subject to crossover (otherwise its matching status w.r.t.

the schema obviously remains unchanged) and if it is, whether the cut point lies in such a way that the fixed characters of the schema are inherited from the same parent (then the matching status of this parent is transferred to the child). W.r.t. the latter alternative, we also have to take into account that a cut point that lets a child inherit the fixed characters of a schema partially from a parent matching the schema and partially from the other may still match the schema, since it may happen that the characters inherited from the other parent match the schema. In particular, this situation occurs if both parents happen to match the schema. Finally, offspring chromosomes that match a schema can be created from parents, both of which do not match the schema, as can be seen from the following example:

$$\begin{array}{lcl} h & = \texttt{***0*|1*1**} & & \texttt{***0*1*1**} = h \\ h \ntriangleright c_1 & = \texttt{00010|11111} & \rightarrow & \texttt{1110111111} = c_1' \lhd h \\ h \ntriangleright c_2 & = \texttt{11101|00100} & \rightarrow & \texttt{0001000100} = c_2' \ntriangleleft h \end{array}$$

Clearly, the reason why a match to the schema is created is that both of the parent chromosomes match the schema *partially* and that these partial matches are combined. Note, however, that at most one child can match the schema in such a case.

As a result of the above considerations we write

$$N(h, t + \Delta t_s + \Delta t_x) = \underbrace{(1 - p_x) \cdot N(h, t + \Delta t_s)}_{A} + \underbrace{p_x \cdot N(h, t + \Delta t_s) \cdot (1 - p_{\text{loss}})}_{B} + C,$$

where p_x is the probability that a chromosome is subject to crossover and p_{loss} is the probability that after applying one-point crossover the offspring does not match schema h anymore. A is the expected number of chromosomes that match schema h and *are not* subject to one-point crossover (and therefore still match the schema h). B is the expected number of chromosomes that *are* subject to one-point crossover and still match the schema h afterward. Finally, C is the expected number of chromosomes matching schema h that are gained by favorable recombinations of chromosomes that do not match schema h themselves.

Since the term C is almost impossible to estimate properly, we simply neglect it. As a consequence, we obtain only a lower bound for $N(h, t + \Delta t_s + \Delta t_x)$, which, however, is sufficient for our purposes. In order to derive an expression for p_{loss}, we draw on the fact that a loss of match is possible only if the randomly chosen cut point falls in such a way that the fixed characters of the schema are *not* all inherited from the same parent. As argued above, this probability is $\frac{\text{deflen}(h)}{L-1}$. Even in these cases, however, the result may still match the schema (see above). As it is difficult to obtain an expression for *all* possible cases in which a match is preserved, we confine ourselves to those cases in which both parents match the schema and therefore the location of the cut point is irrelevant. This provides us with the expression

$$p_{\text{loss}} \leq \frac{\text{deflen}(h)}{L - 1} \cdot \left(1 - \frac{N(h, t + \Delta t_s)}{\mu}\right).$$

The first factor captures the probability that the choice of the cut point is potentially harmful and the second factor captures the probability that the other parent does *not* match the schema and hence there is a certain chance that the result does not match the schema. Clearly, however, this product yields only an upper bound for p_{loss}, because it assumes that any one-point crossover with a potentially harmful cut point and another parent that does not match the schema destroys the match to the schema. This is certainly not the case, as the following example demonstrates:

$$\begin{array}{lll}
h = \texttt{***0*|1*1**} & & \texttt{***0*1*1**} = h \\
h \triangleright c_1 = \texttt{00000|11111} & \rightarrow & \texttt{1110111111} = c_1' \triangleleft h \\
h \ntriangleright c_2 = \texttt{11101|00100} & \rightarrow & \texttt{0000000100} = c_2' \ntriangleleft h
\end{array}$$

Even though chromosome c_2 does not match the schema h and the cut point of one-point crossover falls in such a way that some of the fixed characters of the schema h are inherited from c_1 and some from c_2, the offspring chromosome c_1' matches the schema h. The reason is, of course, that c_2 matches the schema h *partially*. However, capturing partial matches in this analysis is extremely difficult and therefore we confine ourselves to the upper bound for p_{loss} stated above.

Plugging the upper bound for p_{loss} into the formula for $N(h, t + \Delta t_s + \Delta t_x)$ yields

$$\begin{aligned}
&N(h, t + \Delta t_s + \Delta t_x) \\
&\geq (1 - p_x) \cdot N(h, t + \Delta t_s) \\
&\qquad + p_x \cdot N(h, t + \Delta t_s) \cdot \left(1 - \frac{\text{deflen}(h)}{L-1} \cdot \left(1 - \frac{N(h, t + \Delta t_s)}{\mu}\right)\right) \\
&= N(h, t + \Delta t_s) \left(1 - p_x + p_x \cdot \left(1 - \frac{\text{deflen}(h)}{L-1} \cdot \left(1 - \frac{N(h, t + \Delta t_s)}{\mu}\right)\right)\right) \\
&= N(h, t + \Delta t_s) \cdot \left(1 - p_x \frac{\text{deflen}(h)}{L-1} \cdot \left(1 - \frac{N(h, t + \Delta t_s)}{\mu}\right)\right) \\
&= N(h, t) \cdot \frac{\overline{f_t(h)}}{\overline{f_t}} \cdot \left(1 - p_x \frac{\text{deflen}(h)}{L-1} \cdot (1 - N(h, t) \cdot f_{\text{rel}}(h))\right),
\end{aligned}$$

where we exploited in the last step the relationship $N(h, t + \Delta t_s) = N(h, t) \cdot \mu \cdot f_{\text{rel}}(h)$. Note that we obtain only an inequality, because we used an upper bound for p_{loss} and because we neglected potential gains from recombinations of chromosomes that do not match the schema h (captured above by the term C, which is missing here).

Having incorporated the effect of crossover, we now turn to mutation. The effect of bit mutation can easily be captured by the *order* of a schema:

Definition 13.5 (*Order of a Schema*) The *order* $\text{ord}(h)$ of a schema h is the number of zeroes and ones in h, that is, $\text{ord}(h) = \#(h, 0) + \#(h, 1) = L - \#(h, *)$ where the operator # counts the number of occurrences of its second argument in its first.

For instance, $\text{ord}(\texttt{**0*11*10*}) = 5$, because the chromosome contains 2 zeros and 3 ones and thus a total of 5 fixed (that is, not *don't care*) characters.

With the notion of the order of a schema, we can express the probability that a match to a schema does *not* get lost due to a bit mutation of a chromosome as $(1 - p_m)^{\text{ord}(h)}$. The reason is that a single bit gets flipped with the probability p_m

and thus remains unchanged with probability $1 - p_m$ (see Algorithm 12.1 on p. 233). If any of the fixed characters in the schema h, of which there are $\text{ord}(h)$, is flipped in the chromosome, the chromosome does not match the schema anymore. We do not care about the $L - \text{ord}(h)$ remaining bits, because the chromosome matches the schema regardless of their value. Since the bit flips are decided independently, the probability that none of the fixed bits in the schema is flipped is $(1 - p_m)^{\text{ord}(h)}$.

As a consequence, we can express the effect of bit mutation as

$$\begin{aligned} N(h, t+1) &= N(h, t + \Delta t_s + \Delta t_x + \Delta t_m) \\ &= N(h, t + \Delta t_s + \Delta t_x) \cdot (1 - p_m)^{\text{ord}(h)}. \end{aligned}$$

Note that alternative mutation models are easy to handle as well. For example, if at most one bit is flipped in a chromosome (so-called one-bit mutation, see Sect. 12.3.1), then the effect can be described by

$$\begin{aligned} N(h, t+1) &= N(h, t + \Delta t_s + \Delta t_x + \Delta t_m) \\ &= N(h, t + \Delta t_s + \Delta t_x) \cdot \left(1 - \frac{\text{ord}(h)}{L}\right), \end{aligned}$$

where $\text{ord}(h)/L$ is the probability that a fixed character in the schema h is flipped in the chromosome (assuming that all bits are equally likely).

Plugging in the result for $N(h, t + \Delta t_s + \Delta t_x)$ that we derived above, we finally obtain **schema theorem** (for bit mutation):

$$N(h, t+1) \geq N(h, t) \cdot \frac{\overline{f_t(h)}}{\overline{f_t}} \left(1 - p_x \frac{\text{deflen}(h)}{L - 1} \left(1 - \frac{N(h, t)}{\mu} \cdot \frac{\overline{f_t(h)}}{\overline{f_t}}\right)\right) (1 - p_m)^{\text{ord}(h)}.$$

The general form of this relationship between $N(h, t+1)$ and $N(h, t)$ is clearly

$$N(h, t+1) \geq N(h, t) \cdot g(h, t).$$

Simplifying, we may therefore say that the number of chromosomes that match a schema h is multiplied in each generation by some factor and thus develops exponentially in the course of several generations. If $g(h, t) > 1$, the number of matching chromosomes grows exponentially, if $g(h, t) < 1$, it decreases exponentially. Since the number of matching chromosomes cannot decrease for *all* schemata (simply because the population size is constant and the contained chromosomes must match some schemata), there must be schemata for which the number of matching chromosomes grows (unless the number of matching chromosomes stays the same for *all* schemata, which, however, implies that the population is essentially constant).

By considering the factors of $g(h, t)$, we can therefore try to derive properties of schemata for which the number of matching chromosomes grows particularly quickly (that is, for which $g(h, t)$ is large). Since $g(h, t)$ is a product, every factor should be as large as possible. Therefore, such schemata should have

• high mean fitness	(due to the factor $\overline{f_t(h)}/\overline{f_t}$),
• small defining length	(due to the factor $1 - p_x\, \text{deflen}(h)/(L-1)\ldots$), and
• low order	(due to the factor $(1 - p_m)^{\text{ord}(h)}$).

Such schemata are also called **building blocks**, due to which the schema theorem is sometimes also referred to as the **building block hypothesis**: the evolutionary search focuses on promising building blocks of solution candidates.

It should be kept in mind, though, that the schema theorem or the building block hypothesis applies in the derived form only to bit strings, fitness proportionate selection, one-point crossover and bit mutation. If we use different genetic operators, building blocks may be characterized by other properties than order or defining length. However, a high mean fitness is always among the characteristic features, since all selection methods favor such chromosomes, although differently strongly and not always in direct proportion to the fitness values.

It should also be noted that the schema theorem is open to many different lines of criticism. It widely neglects the interplay of different schemata, as well as the possibility of *epistasis* (the whole derivation implicitly assumes (very) low epistasis). It works with expected values that are strictly valid only for an infinite population size (which obviously cannot be achieved in practice, where effects of stochastic drift need to be taken into account). The factor $g(h, t)$ is clearly not constant, but changes over time due to its dependence on the population at time t, so that the claim of an exponential behavior over several generations is slightly dubious (especially, since saturation effects can be expected) etc.

13.1.2 The Two-Armed Bandit Argument

The schema theorem implies that a genetic algorithm achieves a near-optimal balance between exploration of the search space and exploitation of good solution candidates. As an argument for this claim, Holland used the **two-armed bandit** model as an analogy, that is, a slot machine with two independent arms (Holland 1975; Michell 1998). The two arms have different expected (per trial) payoffs μ_1 and μ_2 with variances σ_1^2 and σ_2^2, respectively, all of which are unknown. Without loss of generality we may assume $\mu_1 > \mu_2$, though. It is also unknown, however, which of the two arms of the slot machine has the higher payoff (that is, it is unknown whether μ_1 is assigned to the left or to the right arm). Suppose we may play N games with such a slot machine. What is the best strategy to maximize our winnings?

If we knew which arm has the greater payoff, we would simply use that arm for all N games, thus clearly maximizing our expected winnings. However, since we do not know which arm is better, we must invest some trials into gathering information about which arm might be the one with the higher payoff. For example, we may choose to use $2n$ trials, $2n < N$, for this task, in which we play both arms equally often (that is, we use each arm n times). Afterward, we evaluate which arm has given us the higher average payoff per trial (exploration). In the remaining $N - 2n$ trials we then exclusively play the arm that has the higher observed payoff (exploitation). Our original question can thus be reformulated as: how should we choose n relative to N in order to maximize our (expected) winnings or, equivalently, to minimize our (expected) loss relative to always having chosen the better arm? In other words, how should we balance exploration (initial $2n$ trials) and exploitation (final $N - 2n$ trials)?

Clearly, there are two types of losses involved here: (1) inevitable loss in the information gathering phase, in which we play the worse arm n times (regardless of which arm is the worse one, since we play both arms equally often), and (2) loss due to the fact that we determine the better arm only based on an empirical payoff estimate, which may point us to the wrong arm as the better one. The former refers to the fact that in the $2n$ trials we devote to exploration we necessarily lose

$$L_1(N,n) = n(\mu_1 - \mu_2),$$

because we do n trials with the arm with lower payoff μ_2 instead of using the arm with the higher payoff μ_1. The loss from the remaining $N - 2n$ trials can only be given in stochastic terms. Let p_n be the probability that the average payoffs per trial, as determined empirically in the first $2n$ trials, actually identify the correct arm. (The index n of this probability is due to the fact that it obviously depends on the choice of $2n$: the larger $2n$, the higher the probability that the empirical payoff estimate from the $2n$ exploration trials identifies the correct arm.) That is, with probability p_n we use the arm actually having the higher payoff μ_1 for the remaining $N - 2n$ trials, while with probability $1 - p_n$ we use the arm actually having the lower payoff μ_2 in these trials. In the former case, there is no additional loss (beyond what is incurred in the exploration phase), while in the latter case we lose

$$L_2(N,n) = (N - 2n)(\mu_1 - \mu_2)$$

in the exploitation phase, because we choose the wrong arm (that is, the one actually having the lower payoff μ_2). Therefore the expected total loss is

$$\begin{aligned} L(N,n) &= \underbrace{L_1(N,n)}_{\text{exploration loss}} + (1 - p_n) \underbrace{L_2(N,n)}_{\text{incorrect exploitation loss}} \\ &= n(\mu_1 - \mu_2) + (1 - p_n)(N - 2n)(\mu_1 - \mu_2) \\ &= (\mu_1 - \mu_2)(np_n + (1 - p_n)(N - n)). \end{aligned}$$

The final form nicely captures that in case the better arm is correctly identified (probability p_n), we lose winnings from n times using the worse arm in the exploration phase, while in case the better arm is incorrectly identified (probability $1 - p_n$), we lose winnings from $N - n$ trials with the worse arm (n of which happen in the exploration phase and $N - 2n$ happen in the exploitation phase).

We now have to minimize the loss function $L(N,n)$ w.r.t. n. The main problem here is to express the probability p_n in terms of n (since p_n clearly depends on n: the longer the exploration phase, the higher the chance that it yields a correct decision). Going into details is beyond the scope of this book, so we only present the final result (Holland 1975; Michell 1998): n should be chosen according to

$$n \approx c_1 \ln\left(\frac{c_2 N^2}{\ln(c_3 N^2)}\right),$$

where c_1, c_2 and c_3 are certain positive constants. By rewriting this expression, we can turn it into (Holland 1975; Michell 1998)

$$N - n \approx e^{n/2c_1} \sqrt{\frac{\ln(c_3 N^2)}{c_2}} - n.$$

Since with growing n the term $e^{n/2c_1}$ dominates the expression on the right hand side, this equation can be simplified (accepting further approximation) by

$$N - n \approx e^{cn}.$$

In other words, the total number of trials $N - n$ that are executed with the arm that is observed to be better should increase exponentially compared to the number of trials n that are executed with the arm that is observed to be worse.

This result, though obtained for a two-armed bandit, can be transferred to multi-armed bandits. In this more general form it is then applied to the schemata that are considered in the schema theorem: the arms of the bandit correspond to different schemata, their payoff to the (average) fitness of chromosomes matching them. A chromosome in a population that matches a schema is seen as a trial of the corresponding bandit arm. Recall, however, that a chromosome matches many schemata, thus exhibiting an inherently parallel exploration of the space of schemata.

As we saw in the preceding section, the schema theorem states that the number of chromosomes matching schemata with better than average fitness grows essentially exponentially over several generations. The two- or multi-armed bandit argument now says that this is an optimal strategy to balance exploration of schemata (playing all arms of the bandit) and their exploitation (playing the arm or arms that have been observed to be better than the others).

13.1.3 The Principle of Minimal Alphabets

The **principle of minimal alphabets** is sometimes invoked to claim that binary encodings, as used by genetic algorithms, are "optimal" in a certain sense. The core idea is that the number of possible schemata should be maximized relative to the size of the search space (or the population size), so that the parallelism inherent in the search for schemata is maximally effective. That is, with the chromosomes of the population a number of schemata should be covered that is as large as possible.

If chromosomes are defined as strings of length L over an alphabet $\mathscr{A}$, then the ratio of the number of schemata to the size of the search space is $(|\mathscr{A}| + 1)^L / |\mathscr{A}|^L$. Clearly, this ratio is maximized if the size $|\mathscr{A}|$ of the alphabet is minimized. Since the smallest usable alphabet has $|\mathscr{A}| = 2$, binary codings optimize this ratio.

A more intuitive form of this argument was put forward by Goldberg (1989): the larger the size of the alphabet, the more difficult it is to find meaningful schemata, because a schema is matched by a larger number of chromosomes. Since a schema averages over the fitness of the matching chromosomes, the quality of a schema may be tainted by some bad chromosomes (low fitness) that happen to match the same schema. Therefore one should strive to minimize the number of chromosomes that are matched by a schema. Since a schema h matches $(L - \mathrm{ord}(h))^{|\mathscr{A}|}$ chromosomes, we should use an alphabet of minimal size. Again, since the smallest usable alphabet has $|\mathscr{A}| = 2$, binary codings can be expected to be optimal.

Whether these arguments are convincing is debatable. At least one has to admit that there is a tradeoff between maximizing the number of schemata relative to the search space and expressing the problem in a more natural manner with the help of larger alphabets (Goldberg 1991). Furthermore, a strong argument *in favor* of larger alphabets has been put forward by Antonisse (1989).

13.2 Evolution Strategies

Evolution strategies (ES) (Rechenberg 1973) are the oldest form of an evolutionary algorithm. They focus on numerical optimization problems and therefore work exclusively with chromosomes that are arrays of real-valued numbers. Their name points to the evolution-strategic principles we mentioned in Sect. 11.2: in (natural) evolution not only the organisms are optimized, but also the mechanisms of evolution. These include parameters like reproduction and mortality rates, life spans, vulnerability to mutations, mutation step sizes etc. Apart from the focus on numerical optimization problems, it is a distinctive feature of evolution strategies that in many forms they adapt mutation step sizes as well as their direction.

To be more precise, we are given a function $f : \mathbb{R}^n \to \mathbb{R}$, for which we want to find an optimal argument vector $\mathbf{x} = (x_1, \dots, x_n)$, that is, an argument vector that yields a maximum or minimum of the function f. Chromosomes are therefore such vectors of real-valued numbers. Evolution strategies often (though not always) abandon **crossover**, that is, there may be no recombination of chromosomes. Rather they focus on **mutation** as the core variation operator. Mutation in evolution strategies consists generally in adding a random vector $\mathbf{r}$, each element r_i, $i = 1, \dots, n$, of which is the realization of a normally distributed random variable with mean zero (independent of the element index i) and variances σ_i^2 or standard deviations σ_i. The variances σ_i^2 may or may not depend on the element index i (one variance for the whole vector or a specific variance for each element) and may or may not depend on the generation counter t (time dependent or independent variance).

As we study in more detail below, the variances may also be coupled to the chromosome and may be subject to mutation themselves. In this way, an adaptation of the mutation step sizes and step directions can be realized. Intuitively, we may say that chromosomes with a “suitable” mutation variance—that is, a variance that causes steps of “suitable width” in the region of the search space in which the chromosome is located—are more likely to produce good offspring. As a consequence, this variance can be expected to spread in the population, at least among individuals located in the same region of the search space. It should be noted, though, that the adaptation of the mutation parameters is thus indirect and therefore cannot be expected to be as effective and efficient as the optimization of the function arguments itself. Nevertheless it can help the search process considerably.