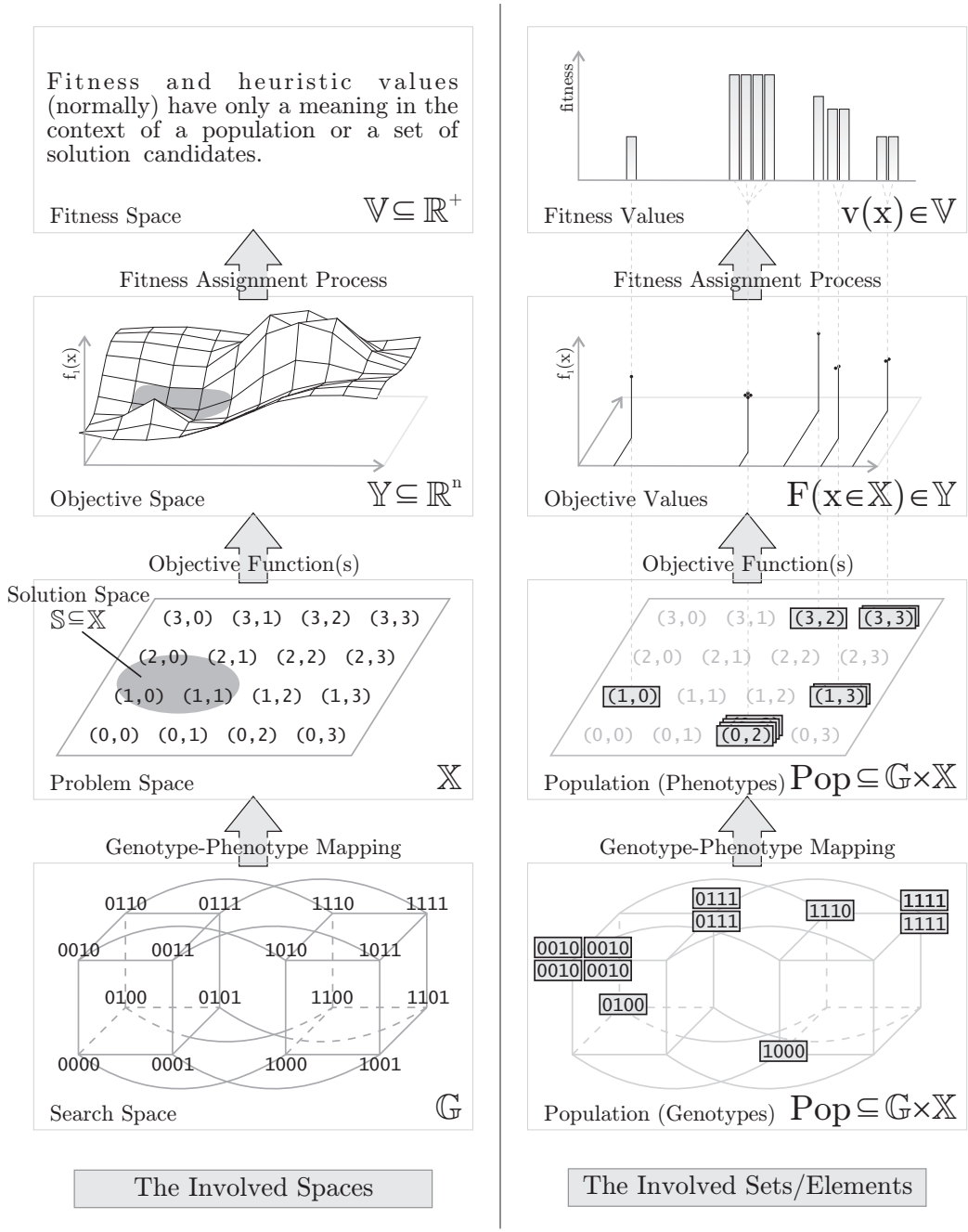

Figure 1.13: Spaces, Sets, and Elements involved in an optimization process.

Definition 1.19 (Solution Candidate). A solution candidate x is an element of the problem space $\mathbb{X}$ of a certain optimization problem.

In the context of evolutionary algorithms, solution candidates are usually called *phenotypes*. In this book, we will use both terms synonymously. Somewhere inside the problem space, the solutions of the optimization problem will be located (if the problem can actually be solved, that is).

Definition 1.20 (Solution Space). We call the union of all solutions of an optimization problem its solution space $\mathbb{S}$.

$$\mathbf{X}^\star \subseteq \mathbb{S} \subseteq \mathbb{X} \tag{1.22}$$

This solution space contains (and can be equal to) the global optimal set $\mathbf{X}^\star$. There may exist valid solutions $x \in \mathbb{S}$ which are not elements of the $\mathbf{X}^\star$, especially in the context of constraint optimization (see Section 1.2.3).

The Search Space

Definition 1.21 (Search Space). The search space $\mathbb{G}$ of an optimization problem is the set of all elements g which can be processed by the search operations.

As previously mentioned, the type of the solution candidates depends on the problem to be solved. Since there are many different applications for optimization, there are many different forms of problem spaces. It would be cumbersome to develop search operations time and again for each new problem space we encounter. Such an approach would not only be error-prone, it would also make it very hard to formulate general laws and to consolidate findings. Instead, we often reuse well-known search spaces for many different problems. Then, only a mapping between search and problem space has to be defined (see page 44). Although this is not always possible, it allows us to use more out-of-the-box software in many cases.

In dependence on genetic algorithms, we often refer to the search space synonymously as *genome*[25], a term coined by the German biologist Winkler [2241] as a portmanteau of the words gene[26] and chromosome [1267]. The genome is the whole hereditary information of organisms. This includes both, the genes and the non-coding sequences of the Deoxyribonucleic acid (DNA[27]), which is illustrated in Figure 1.14. Simply put, the DNA is a string of

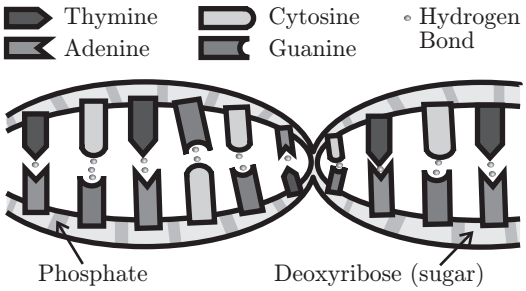

Figure 1.14: A sketch of a part of a DNA molecule.

base pairs that encodes the phenotypical characteristics of the creature it belongs to.

[25] http://en.wikipedia.org/wiki/Genome [accessed 2007-07-15]

[26] The words gene, genotype, and phenotype have, in turn, been introduced by the Danish biologist Johannsen [1056]. [2240]

[27] http://en.wikipedia.org/wiki/Dna [accessed 2007-07-03]

Definition 1.22 (Genotype). The elements $g \in \mathbb{G}$ of the search space $\mathbb{G}$ of a given optimization problem are called the *genotypes*.

The elements of the search space rarely are unstructured aggregations. Instead, they often consist of distinguishable parts, hierarchical units, or well-typed data structures. The same goes for the DNA in biology. It consists of *genes*, segments of nucleic acid, that contain the information necessary to produce RNA strings in a controlled manner[28]. A fish, for instance, may have a gene for the color of its scales. This gene, in turn, could have two possible "values" called *alleles*[29], determining whether the scales will be brown or gray. The genetic algorithm community has adopted this notation long ago and we can use it for arbitrary search spaces.

Definition 1.23 (Gene). The distinguishable units of information in a genotype that encode the phenotypical properties are called genes.

Definition 1.24 (Allele). An allele is a value of specific gene.

Definition 1.25 (Locus). The locus[30] is the position where a specific gene can be found in a genotype.

Figure 1.15 on page 45 refines the relations of genotypes and phenotypes from the initial example for the spaces in Figure 1.13 by also marking genes, alleles, and loci. In the car customizing problem also mentioned earlier, the first gene could identify the color of the automobile. Its locus would then be 0 and it could have the alleles `00`, `01`, `10`, and `11`, encoding for red, white, green, and blue, for instance. The second gene (at locus 1) with the alleles `0` or `1` may define whether or not the car comes with climate control, and so on.

The Search Operations

In some problems, the search space $\mathbb{G}$ may be identical to the problem space $\mathbb{X}$. If we go back to our previous examples, for instance, we will find that there exist a lot of optimization strategies that work directly on vectors of real numbers. When minimizing a real function, we could use such an approach (Evolution Strategies, for instance, see Chapter 5 on page 227) and set $\mathbb{G} = \mathbb{X} = \mathbb{R}$. Also, the configurations of cars may be represented as bit strings: Assume that such a configuration consists of k features, which can either be included or excluded from an offer to the customer. We can then search in the space of binary strings of this length $\mathbb{G} = \mathbb{B}^k = \{\texttt{true}, \texttt{false}\}^k$, which is exactly what genetic algorithms (discussed in Section 3.1 on page 141) do. By using their optimization capabilities, we do not need to mess with the search and selection techniques but can rely on well-researched standard operations.

Definition 1.26 (Search Operations). The search operations searchOp are used by optimization algorithms in order to explore the search space $\mathbb{G}$.

We subsume all search operations which are applied by an optimization algorithm in order to solve a given problem in the set *Op*. Search operations can be defined with different arities[31]. Equation 1.23, for instance, denotes an n-ary operator, i. e., one with n arguments. The result of a search operation is one element of the search space.

$$\text{searchOp} : \mathbb{G}^n \mapsto \mathbb{G} \tag{1.23}$$

[28] http://en.wikipedia.org/wiki/Gene [accessed 2007-07-03]
[29] http://en.wikipedia.org/wiki/Allele [accessed 2007-07-03]
[30] http://en.wikipedia.org/wiki/Locus_%28genetics%29 [accessed 2007-07-03]
[31] http://en.wikipedia.org/wiki/Arity [accessed 2008-02-15]

Mutation and crossover in genetic algorithms (see Chapter 3) are examples for unary and binary search operations, whereas Differential Evolution utilizes a ternary operator (see Section 5.5). Optimization processes are often initialized by creating random genotypes – usually the results of a search operation with zero arity (no parameters).

Search operations often involve randomized numbers. In such cases, it makes no sense to reason about their results like $\exists g_1, g_2 \in \mathbb{G} : g_2 = \text{searchOp}(g_1) \wedge \ldots$ Instead, we need to work with probabilities like $\exists g_1, g_2 \in \mathbb{G} : g_2 = P(\text{searchOp}(g_1)) > 0 \wedge \ldots$ Based on Definition 1.26, we will use the notation $Op(x)$ for the application of any of the operations searchOp $\in Op$ to the genotype x. With $Op^k(x)$ we denote k successive applications of (possibly different) search operators. If the parameter x is left away, i. e., just Op^k is written, this chain has to start with a search operation with zero arity. In the style of Badea and Stanciu [111] and Skubch [1897, 1898], we now can define:

Definition 1.27 (Completeness). A set Op of search operations searchOp is *complete* if and only if every point g_1 in the search space $\mathbb{G}$ can be reached from every other point $g_2 \in \mathbb{G}$ by applying only operations searchOp $\in Op$.

$$\forall g_1, g_2 \in \mathbb{G} \Rightarrow \exists k \in \mathbb{N} : P\big(g_1 = Op^k(g_2)\big) > 0 \tag{1.24}$$

Definition 1.28 (Weak Completeness). A set Op of search operations searchOp is *weakly complete* if and only if every point g in the search space $\mathbb{G}$ can be reached by applying only operations searchOp $\in Op$. A weakly complete set of search operations hence includes at least one parameterless function.

$$\forall g \in \mathbb{G} \Rightarrow \exists k \in \mathbb{N} : P\big(g = Op^k\big) > 0 \tag{1.25}$$

If the set of search operations is not complete, there are points in the search space which cannot be reached. Then, we are probably not able to explore the problem space adequately and possibly will not find satisfyingly good solution.

Definition 1.29 (Adjacency (Search Space)). A point g_2 is adjacent to a point g_1 in the search space $\mathbb{G}$ if it can be reached by applying a single search operation searchOp to g_1. Notice that the adjacency relation is not necessarily symmetric.

$$\text{adjacent}(g_2, g_1) = \begin{cases} \texttt{true} \text{ if } \exists \text{searchOp} \in Op : P(\text{searchOp}(g_1) = g_2) > 0 \\ \texttt{false} \text{ otherwise} \end{cases} \tag{1.26}$$

The Connection between Search and Problem Space

If the search space differs from the problem space, a translation between them is furthermore required. In our car example, we would need to transform the binary strings processed by the genetic algorithm to objects which represent the corresponding car configurations and can be processed by the objective functions.

Definition 1.30 (Genotype-Phenotype Mapping). The genotype-phenotype mapping (GPM, or ontogenic mapping [1619]) $\text{gpm} : \mathbb{G} \mapsto \mathbb{X}$ is a left-total[32] binary relation which maps the elements of the search space $\mathbb{G}$ to elements in the problem space $\mathbb{X}$.

$$\forall g \in \mathbb{G}\ \exists x \in \mathbb{X} : \text{gpm}(g) = x \tag{1.27}$$

The only hard criterion we impose on genotype-phenotype mappings in this book is left-totality, i. e., that they map each element of the search space to at least one solution candidate. They *may* be functional relations if they are deterministic. Although it is possible to create mappings which involve random numbers and, hence, cannot be considered to be

[32] See Equation 27.51 on page 461 to 5 on page 462 for an outline of the properties of binary relations.

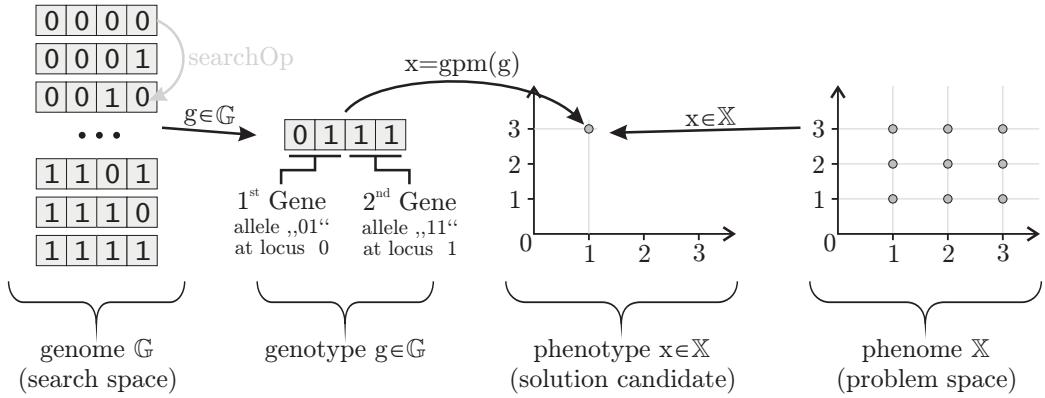

Figure 1.15: The relation of genome, genes, and the problem space.

functions in the mathematical sense of Section 27.7.1 on page 462. Then, Equation 1.27 would need to be rewritten to Equation 1.28.

$$\forall g \in \mathbb{G}\ \exists x \in \mathbb{X} : P(\text{gpm}(g) = x) > 0 \tag{1.28}$$

Genotype-phenotype mappings *should* further be surjective [1694], i. e., relate at least one genotype to each element of the problem space. Otherwise, some solution candidates can never be found and evaluated by the optimization algorithm and there is no guarantee whether the solution of a given problem can be discovered or not. If a genotype-phenotype mapping is injective, which means that it assigns distinct phenotypes to distinct elements of the search space, we say that it is free from *redundancy*. There are different forms of redundancy, some are considered to be harmful for the optimization process, others have positive influence[33]. Most often, GPMs are not bijective (since they are neither necessarily injective nor surjective). Nevertheless, if a genotype-phenotype mapping is bijective, we can construct an inverse mapping $\text{gpm}^{-1} : \mathbb{X} \mapsto \mathbb{G}$.

$$\text{gpm}^{-1}(x) = g \Leftrightarrow \text{gpm}(g) = x\ \forall x \in \mathbb{X}, g \in \mathbb{G} \tag{1.29}$$

Based on the genotype-phenotype mapping, we can also define an adjacency relation for the problem space, which, of course, is also not necessarily symmetric.

Definition 1.31 (Adjacency (Problem Space)). A point x_2 is adjacent to a point x_1 in the problem space $\mathbb{X}$ if it can be reached by applying a single search operation searchOp to their corresponding elements in the problem space.

$$\text{adjacent}(x_2, x_1) = \begin{cases} \texttt{true} & \text{if } \exists g_1, g_2 : x_1 = \text{gpm}(g_1) \wedge x_2 = \text{gpm}(g_2) \wedge \text{adjacent}(g_2, g_1) \\ \texttt{false} & \text{otherwise} \end{cases} \tag{1.30}$$

By the way, we now have the means to define the term *local optimum* clearer. The original Definition 1.8 only applies to single objective functions, but with the use of the adjacency relation adjacent, the prevalence criterion $\succ$, and the connection between the search space and the problem space gpm, we clarify it for multiple objectives.

Definition 1.32 (Local Optimum). A (local) optimum $x_l^\star \in \mathbb{X}$ of a set of objective functions F function is not worse than all points adjacent to it.

$$\forall x_l^\star \in \mathbb{G} \Rightarrow \left(\forall x \in \mathbb{X} : \text{adjacent}(x, x_l^\star) \Rightarrow \overline{x \succ x_l^\star}\right) \tag{1.31}$$

[33] See Section 1.4.5 on page 67 for more information.

The Objective Space and Optimization Problems

After the appropriate problem space has been defined, the search space has been selected and a translation between them (if needed) was created, we are almost ready to feed the problem to a global optimization algorithm. The main purpose of such an algorithm obviously is to find as many elements as possible from the solution space – We are interested in the solution candidates with the best possible evaluation results. This evaluation is performed by the set F of n objective functions $f \in F$, each contributing one numerical value describing the characteristics of a solution candidate x.[34]

Definition 1.33 (Objective Space). The objective space $\mathbb{Y}$ is the space spanned by the codomains of the objective functions.

$$F = \{f_i : \mathbb{X} \mapsto Y_i \, : \, 0 < i \leq n, Y_i \subseteq \mathbb{R}\} \Rightarrow \mathbb{Y} = Y_1 \times Y_2 \times .. \times Y_n \tag{1.32}$$

The set F maps the elements x of the problem space $\mathbb{X}$ to the objective space $\mathbb{Y}$ and, by doing so, gives the optimizer information about their qualities as solutions for a given problem.

Definition 1.34 (Optimization Problem). An optimization problem is defined by a five-tuple $(\mathbb{X}, F, \mathbb{G}, Op, \text{gpm})$ specifying the problem space $\mathbb{X}$, the objective functions F, the search space $\mathbb{G}$, the set of search operations Op, and the genotype-phenotype mapping gpm. In theory, such an optimization problem *can* always be solved if Op is complete and the gpmis surjective.

Generic search and optimization algorithms find optimal elements if provided with an optimization problem defined in this way. Evolutionary algorithms, which we will discuss later in this book, are generic in this sense. Other optimization methods, like genetic algorithmsfor example, may be more specialized and work with predefined search spaces and search operations.

Fitness as a Relative Measure of Utility

When performing a multi-objective optimization, i. e., $n = |F| > 1$, the elements of $\mathbb{Y}$ are vectors in $\mathbb{R}^n$. In Section 1.2.2 on page 27, we have seen that such vectors cannot always be compared directly in a consistent way and that we need some (comparative) measure for what is "good". In many optimization techniques, especially in evolutionary algorithms, this measure is used to map the objective space to a subset $\mathbb{V}$ of the positive real numbers $\mathbb{R}^+$. For each solution candidate, this single real number represents its *fitness* as solution for the given optimization problem. The process of computing such a fitness value is often not solely depending on the absolute objective values of the solution candidates but also on those of the other phenotypes known. It could, for instance, be position of a solution candidate in the list of investigated elements sorted according to the Pareto relation. Hence, fitness values often only have a meaning inside the optimization process [712] and may change by time, even if the objective values stay constant. In deterministic optimization methods, the value of a heuristic function which approximates how many modifications we will have to apply to the element in order to reach a feasible solution can be considered as the fitness.

Definition 1.35 (Fitness). The fitness[35] value $v(x) \in \mathbb{V}$ of an element x of the problem space $\mathbb{X}$ corresponds to its utility as solution or its priority in the subsequent steps of the optimization process. The space spanned by all possible fitness values $\mathbb{V}$ is normally a subset of the positive real numbers $\mathbb{V} \subseteq \mathbb{R}^+$.

[34] See also Equation 1.3 on page 27.

[35] http://en.wikipedia.org/wiki/Fitness_(genetic_algorithm) [accessed 2008-08-10]

The origin of the term fitness has been borrowed biology[36] [1915, 1624] by the evolutionary algorithms community. When the first applications of genetic algorithms were developed, the focus was mainly on single-objective optimization. Back then, they called this single function fitness function and thus, set objective value ≡ fitness value. This point of view is obsolete in principle, yet you will find many contemporary publications that use this notion. This is partly due the fact that in simple problems with only one objective function, the old approach of using the objective values directly as fitness, i. e., $v(x) = f(x)\ \forall x \in \mathbb{X}$, can sometimes actually be applied. In multi-objective optimization processes, this is not possible and fitness assignment processes like those which we are going to elaborate on in Section 2.3 on page 111 are applied instead.

In the context of this book, fitness is subject to minimization, i. e., elements with smaller fitness are "better" than those with higher fitness. Although this definition differs from the biological perception of fitness, it complies with the idea that optimization algorithms are to find the minima of mathematical functions (if nothing else has been stated).

Futher Definitions

In order to ease the discussions of different global optimization algorithms, we furthermore define the data structure *individual*. Especially evolutionary algorithms, but also many other techniques, work on sets of such individuals. Their fitness assignment processes determine fitness values for the individuals relative to all elements of these *populations*.

Definition 1.36 (Individual). An individual p is a tuple $(p.g, p.x)$ of an element $p.g$ in the search space $\mathbb{G}$ and the corresponding element $p.x = \text{gpm}p.g$ in the problem space $\mathbb{X}$.

Besides this basic individual structure, many practical realizations of optimization algorithms use such a data structure to store additional information like the objective and fitness values. Then, we will consider individuals as tuples in $\mathbb{G} \times \mathbb{X} \times Z$, where Z is the space of the additional information stored – $Z = \mathbb{Y} \times \mathbb{V}$, for instance. In the algorithm definitions later in this book, we will often access the phenotypes $p.x$ without explicitly using the genotype-phenotype mapping, since the relation of $p.x$ and $p.g$ complies to Definition 1.36.

Definition 1.37 (Population). A population *Pop* is a list of individuals used during an optimization process.

$$Pop \subseteq \mathbb{G} \times \mathbb{X} : \ \forall p = (p.g, p.x) \in Pop \Rightarrow p.x = \text{gpm}(p.g) \tag{1.33}$$

As already mentioned, the fitness $v(x)$ of an element x in the problem space $\mathbb{X}$ often not solely depends on the element itself. Normally, it is rather a relative measure putting the features of x into the context of a set of solution candidates x. We denote this by writing $v(x, X)$. It is also possible that the fitness involves the whole individual data, including the genotypic and phenotypic structures. We can denote this by writing $v(p, Pop)$.

1.3.2 Fitness Landscapes and Global Optimization

A very powerful metaphor in global optimization is the fitness landscape[37]. Like many other abstractions in optimization, fitness landscapes have been developed and extensively been researched by evolutionary biologists [2261, 1099, 775, 502]. Basically, they are visualizations of the relationship between the genotypes or phenotypes in a given population and their corresponding reproduction probability. The idea of such visualizations goes back to Wright [2261], who used level contours diagrams in order to outline the effects of selection,

[36] http://en.wikipedia.org/wiki/Fitness_%28biology%29 [accessed 2008-02-22]
[37] http://en.wikipedia.org/wiki/Fitness_landscape [accessed 2007-07-03]

mutation, and crossover on the capabilities of populations to escape local optimal configurations. Similar abstractions arise in many other areas [1954], like in physics of disordered systems like spin-glasses [208, 1402], for instance.

In Chapter 2, we will discuss evolutionary algorithms, which are optimization methods inspired by natural evolution. The evolutionary algorithm research community has widely adopted the fitness landscapes as relation between individuals and their objective values [1431, 623]. Langdon and Poli [1242][38] explain that fitness landscapes can be imagined as a view on a countryside from far above. The height of each point is then analogous to its objective value. An optimizer can then be considered as a short-sighted hiker who tries to find the lowest valley or the highest hilltop. Starting from a random point on the map, she wants to reach this goal by walking the minimum distance.

As already mentioned, evolutionary algorithms were first developed as single-objective optimization methods. Then, the objective values were directly used as fitness and the "reproduction probability", i. e., the chance of a solution candidate for being subject of further investigation, was proportional to them. In multi-objective optimization applications with more sophisticated fitness assignment and selection processes, this simple approach does not reflect the biological metaphor correctly anymore.

In the context of this book we will book, we therefore deviate from this view. Since it would possibly be confusing for the reader if we used a different definition for fitness landscapes than the rest of the world, we introduce the new term *problem landscape* and keep using the term *fitness landscape* in the traditional manner. In Figure 1.19 on page 57, you can find some examples for fitness landscapes.

Definition 1.38 (Problem Landscape).
The problem landscape $\Phi : \mathbb{X} \times \mathbb{N} \mapsto [0,1] \subset \mathbb{R}^+$ maps all the points x in a problem space $\mathbb{X}$ to the cumulative probability of reaching them until (inclusively) the τ^{th} evaluation of a solution candidate. The problem landscape thus depends on the optimization problem and on the algorithm applied in order to solve the problem.

$$\Phi(x,\tau) = P\big(x \text{ has been visited until the } \tau^{th} \text{ individual evaluation}\big) \ \forall x \in \mathbb{X}, \tau \in \mathbb{N} \quad (1.34)$$

This definition of problem landscape is very similar to the performance measure definition used by Wolpert and Macready [2244, 2245] in their No Free Lunch Theorem which will be discussed in Section 1.4.10 on page 76. In our understanding, problem landscapes are not only closer to the original meaning of fitness landscapes in biology, they also have another advantage. According to this definition, all entities involved in an optimization process directly influence the problem landscape. The choice of the search operations in the search space $\mathbb{G}$, the way the initial elements are picked, the genotype-phenotype mapping, the objective functions, the fitness assignment process, and the way individuals are selected for further exploration all have impact on Φ. We can furthermore make the following assumptions about $\Phi x \tau$, since it is basically a some form of cumulative distribution function (see Definition 28.18 on page 470).

$$\Phi(x,\tau_1) \geq \Phi(x,\tau_2) \ \forall \tau_1 < \tau_2 \wedge x \in \mathbb{X}, \tau_1, \tau_2 \in \mathbb{N} \quad (1.35)$$

$$0 \leq \Phi(x,\tau) \leq 1 \ \forall x \in \mathbb{X}, \tau \in \mathbb{N} \quad (1.36)$$

Referring back to Definition 1.34, we can now also define what optimization algorithms are.

Definition 1.39 (Optimization Algorithm). An optimization algorithm is a transformation $(\mathbb{X}, F, \mathbb{G}, Op, \text{gpm}) \mapsto \Phi$ of an optimization problem $(\mathbb{X}, F, \mathbb{G}, Op, \text{gpm})$ to a problem landscape Φ that will find at least one *local* optimum $x_l^\star$ for each optimization problem

[38] This part of [1242] is also online available at `http://www.cs.ucl.ac.uk/staff/W.Langdon/FOGP/intro_pic/landscape.html` [accessed 2008-02-15].

$(\mathbb{X}, F, \mathbb{G}, Op, \text{gpm})$ with a weakly complete set of search operations Op and a surjective genotype-phenotype mapping gpm if granted infinite processing time and if such an optimum exists (see Equation 1.37).

$$\exists x_l^\star \in \mathbb{X} : \lim_{\tau \to \infty} \Phi(x_l^\star, \tau) = 1 \tag{1.37}$$

An optimization algorithm is characterized by

1. the way it assigns fitness to the individuals,
2. the ways it selects them for further investigation,
3. the way it applies the search operations, and
4. the way it builds and treats its state information.

The first condition in Definition 1.40, the completeness of Op, is mandatory because the search space $\mathbb{G}$ cannot be explored fully otherwise. If the genotype-phenotype mapping gpm is not surjective, there exist points in the problem space $\mathbb{X}$ which can never be evaluated. Only if both conditions hold, it is guaranteed that an optimization algorithm can find at least one local optimum.

The best optimization algorithm for a given problem $(\mathbb{X}, F, \mathbb{G}, Op, \text{gpm})$ is the one with the highest values of $\Phi(\mathbf{x}^\star, \tau)$ for the optimal elements $\mathbf{x}^\star$ in the problem space and for the lowest values of τ. It may be interesting that this train of thought indicates that finding the best optimization algorithm for a given optimization problem is, itself, a multi-objective optimization problem.

Definition 1.40 (Global Optimization Algorithm). Global optimization algorithms are optimization algorithms that employs measures that prevent convergence to local optima and increase the probability of finding a global optimum.

For a perfect global optimization algorithm (given an optimization problem with weakly complete search operations and a surjective genotype-phenotype mapping), Equation 1.38 would hold. In reality, it can be considered questionable whether such an algorithm can actually be built.

$$\forall x_1, x_2 \in \mathbb{X} : x_1 \succ x_2 \Rightarrow \lim_{\tau \to \infty} \Phi(x_1, \tau) > \lim_{\tau \to \infty} \Phi(x_2, \tau) \tag{1.38}$$

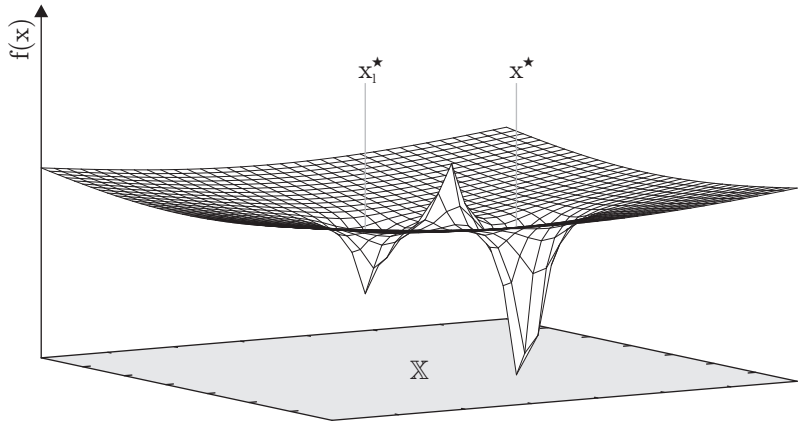

Figure 1.16: An example optimization problem.

Let us now give a simple example for problem landscapes and how they are influenced by the optimization algorithm applied to them. Figure 1.16 illustrates one objective function,

defined over a finite subset $\mathbb{X}$ of the two-dimensional real plane, which we are going to optimize. We use the problem space $\mathbb{X}$ also as search space $\mathbb{G}$, so we can do not need a genotype-phenotype mapping. For optimization, we will use a very simple hill climbing algorithm[39], which initially randomly creates one solution candidate uniformly distributed in $\mathbb{X}$. In each iteration, it creates a new solution candidate from the known one using an unary search operation. The old and the new candidate are compared, and the better one is kept. Hence, we do not need to differentiate between fitness and objective values. In the example, *better* means *has lower fitness*. In Figure 1.16, we can spot one local optimum $x_l^\star$ and one global optimum $\mathbf{x}^\star$. Between them, there is a hill, an area of very bad fitness. The rest of the problem space exhibits a small gradient into the direction its center. The optimization algorithm will likely follow this gradient and sooner or later discover $x_l^\star$ or $\mathbf{x}^\star$. The chances of $x_l^\star$ are higher, since it is closer to the center of $\mathbb{X}$.

With this setting, we have recorded the traces of two experiments with 1.3 million runs of the optimizer (8000 iterations each). From these records, we can approximate the problem landscapes very good.

In the first experiment, depicted in Figure 1.17, we used a search operation $\text{searchOp}_1 : \mathbb{X} \mapsto \mathbb{X}$ which created a new solution candidate normally distributed around the old one. In all experiments, we had divided $\mathbb{X}$ in a regular lattice. $\text{searchOp}_2 : \mathbb{X} \mapsto \mathbb{X}$, used in the second experiment, the new solution candidates are direct neighbors of the old ones in this lattice. The problem landscape Φ produced by this operator is shown in Figure 1.18. Both operators are complete, since each point in the search space can be reached from each other point by applying them.

$$\text{searchOp}_1(x) \equiv (x_1 + \text{random}_n(), x_2 + \text{random}_n()) \tag{1.39}$$

$$\text{searchOp}_1(x) \equiv (x_1 + \text{random}_u(-1, 1)\,, x_2 + \text{random}_u(-1, 1)) \tag{1.40}$$

In both experiments, the first probabilities of the elements of the search space of being discovered are very low, near to zero in the first few iterations. To put it precise, since our problem space is a 36×36 lattice, this probability is $1/36^2$ in the first iteration. Starting with the tenth or so iteration, small peaks begin to form around the places where the optima are located. These peaks grow

Well, as already mentioned, this idea of problem landscapes and optimization reflects solely the author's views. Notice also that it is not always possible to define problem landscapes for problem spaces which are *uncountable infinitely* large. Since the local optimum $x_l^\star$ at the center of the large basin and the gradient points straighter into its direction, it has a higher probability of being found than the global optimum $\mathbf{x}^\star$. The difference between the two search operators tested becomes obvious starting with approximately the 2000^{th} iteration. In the hill climber with the operator utilizing the normal distribution, the Φ value of the global optimum begins to rise farther and farther, finally surpassing the one of the local optimum. Even if the optimizer gets trapped in the local optimum, it will still eventually discover the global optimum and if we had run this experiment longer, the according probability would have converge to 1. The reason for this is that with the normal distribution, all points in the search space have a non-zero probability of being found from all other points in the search space. In other words, all elements of the search space are adjacent.

The operator based on the uniform distribution is only able to create points in the direct neighborhood of the known points. Hence, if an optimizer gets trapped in the local optimum, it can never escape. If it arrives at the global optimum, it will never discover the local one. In Fig. 1.18.l, we can see that $\Phi(x_l^\star, 8000) \approx 0.7$ and $\Phi(\mathbf{x}^\star, 8000) \approx 0.3$. One of the two points will be the result of the optimization process.

From the example we can draw four conclusions:

1. Optimization algorithms discover good elements with higher probability than elements with bad characteristics. Well, this is what they should do.

[39] Hill climbing algorithms are discussed thoroughly in Chapter 10.

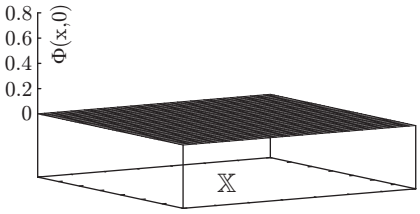

Fig. 1.17.a: $\Phi(x, 1)$

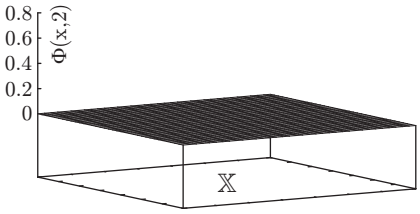

Fig. 1.17.b: $\Phi(x, 2)$

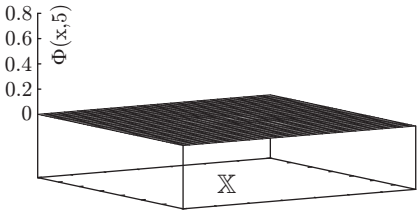

Fig. 1.17.c: $\Phi(x, 5)$

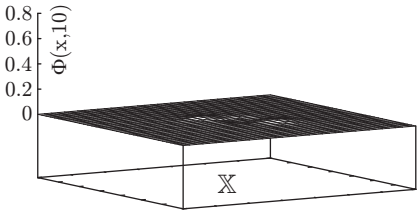

Fig. 1.17.d: $\Phi(x, 10)$

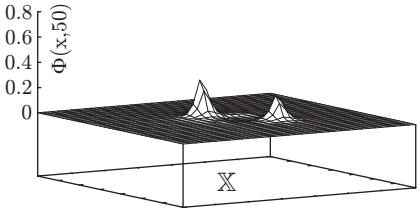

Fig. 1.17.e: $\Phi(x, 50)$

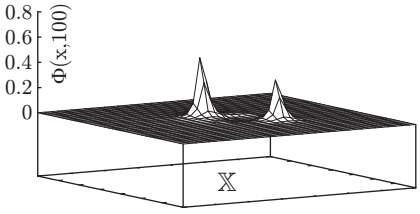

Fig. 1.17.f: $\Phi(x, 100)$

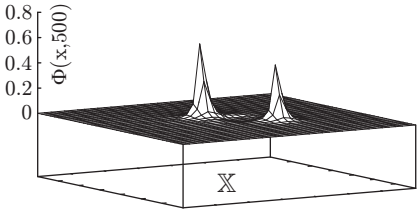

Fig. 1.17.g: $\Phi(x, 500)$

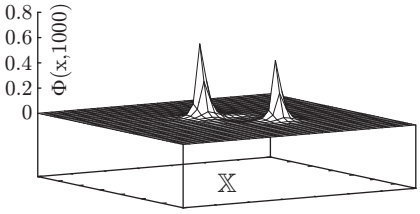

Fig. 1.17.h: $\Phi(x, 1000)$

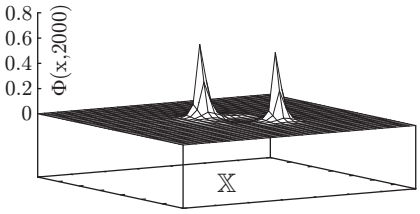

Fig. 1.17.i: $\Phi(x, 2000)$

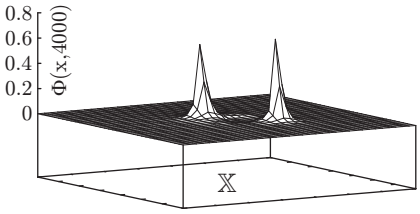

Fig. 1.17.j: $\Phi(x, 4000)$

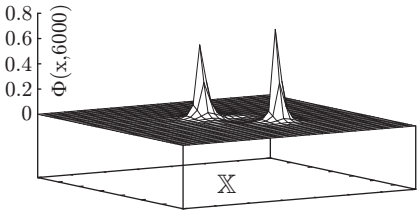

Fig. 1.17.k: $\Phi(x, 6000)$

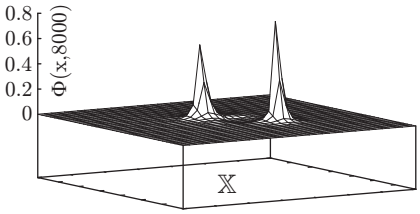

Fig. 1.17.l: $\Phi(x, 8000)$

Figure 1.17: The problem landscape of the example problem derived with searchOp_1.

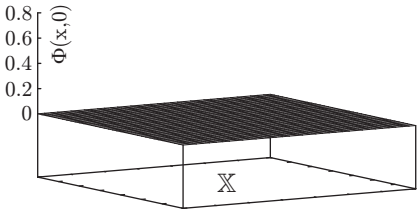

Fig. 1.18.a: $\Phi(x, 1)$

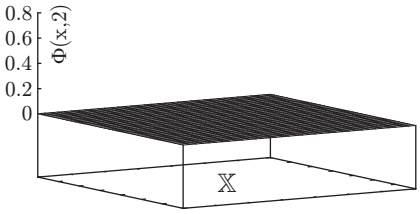

Fig. 1.18.b: $\Phi(x, 2)$

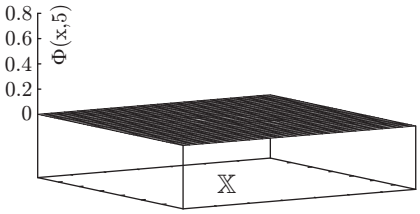

Fig. 1.18.c: $\Phi(x, 5)$

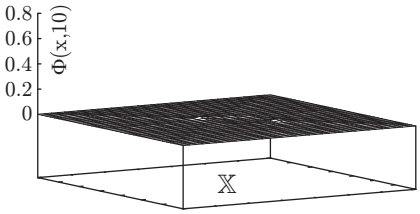

Fig. 1.18.d: $\Phi(x, 10)$

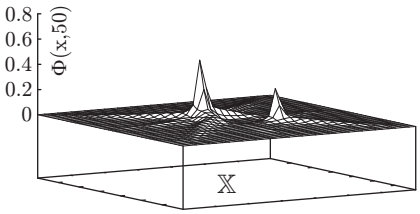

Fig. 1.18.e: $\Phi(x, 50)$

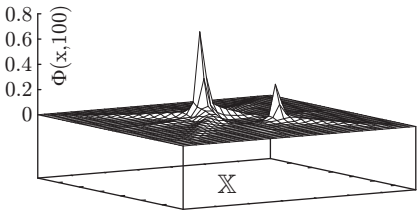

Fig. 1.18.f: $\Phi(x, 100)$

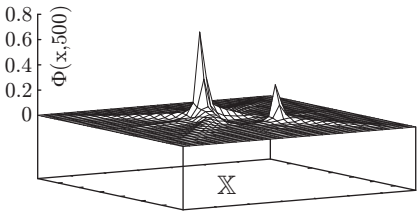

Fig. 1.18.g: $\Phi(x, 500)$

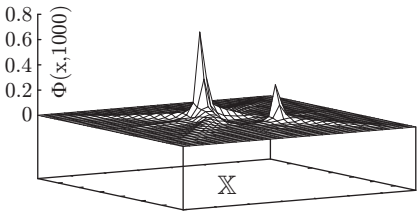

Fig. 1.18.h: $\Phi(x, 1000)$

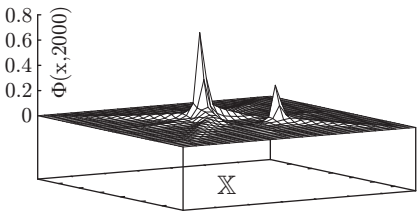

Fig. 1.18.i: $\Phi(x, 2000)$

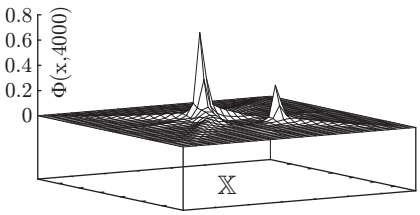

Fig. 1.18.j: $\Phi(x, 4000)$

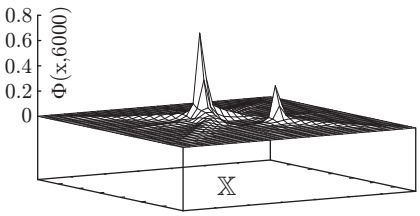

Fig. 1.18.k: $\Phi(x, 6000)$

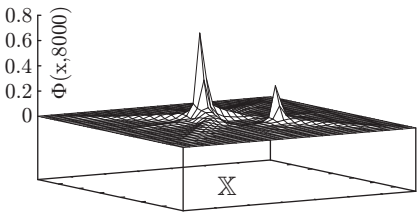

Fig. 1.18.l: $\Phi(x, 8000)$

Figure 1.18: The problem landscape of the example problem derived with searchOp_2.

2. The success of optimization depends very much on the way the search is conducted.
3. It also depends on the time (or the number of iterations) the optimizer allowed to use.
4. Hill climbing algorithms are no global optimization algorithms since they have no means of preventing getting stuck at local optima.

1.3.3 Gradient Descend

Definition 1.41 (Gradient). A gradient[40] of a scalar field $f : \mathbb{R}^n \mapsto \mathbb{R}$ is a vector field which points into the direction of the greatest increase of the scalar field. It is denoted by ∇f or $\text{grad}(f)$.

Optimization algorithms depend on some form of gradient in objective or fitness space in order to find good individuals. In most cases, the problem space $\mathbb{X}$ is not a vector space over the real numbers $\mathbb{R}$, so we cannot directly differentiate the objective functions with Nabla operator[41] ∇F. Generally, samples of the search space are used to approximate the gradient. If we compare to elements x_1 and x_2 of problem space and find $x_1 \succ x_2$, we can assume that there is some sort of gradient facing downwards from x_2 to x_1. When descending this gradient, we can hope to find an x_3 with $x_3 \succ x_1$ and finally the global minimum.

1.3.4 Other General Features

There are some further common semantics and operations that are shared by most optimization algorithms. Many of them, for instance, start out by randomly creating some initial individuals which are then refined iteratively. Optimization processes which are not allowed to run infinitely have to find out when to terminate. In this section we define and discuss general abstractions for such commonalities.

Iterations

Global optimization algorithms often iteratively evaluate solution candidates in order to approach the optima. We distinguish between evaluations τ and iterations t.

Definition 1.42 (Evaluation). The value $\tau \in \mathbb{N}_0$ denotes the number of solution candidates for which the set of objective functions F has been evaluated.

Definition 1.43 (Iteration). An iteration[42] refers to one round in a loop of an algorithm. It is one repetition of a specific sequence of instruction inside an algorithm.

Algorithms are referred to as *iterative* if most of their work is done by cyclic repetition of one main loop. In the context of this book, an iterative optimization algorithm starts with the first step $t = 0$. The value $t \in \mathbb{N}_0$ is the index of the iteration currently performed by the algorithm and $t + 1$ refers to the following step. One example for iterative algorithm is Algorithm 1.1. In some optimization algorithms like genetic algorithms, for instance, iterations are referred to as *generations*.

There often exists a well-defined relation between the number of performed solution candidate evaluations τ and the index of the current iteration t in an optimization process: Many global optimization algorithms generate and evaluate a certain number of individuals per generation.

[40] http://en.wikipedia.org/wiki/Gradient [accessed 2007-11-06]
[41] http://en.wikipedia.org/wiki/Del [accessed 2008-02-15]
[42] http://en.wikipedia.org/wiki/Iteration [accessed 2007-07-03]

Termination Criterion

The termination criterion terminationCriterion() is a function with access to all the information accumulated by an optimization process, including the number of performed steps t, the objective values of the best individuals, and the time elapsed since the start of the process. With terminationCriterion(), the optimizers determine when they have to halt.

Definition 1.44 (Termination Criterion). When the termination criterion function terminationCriterion() $\in \{\texttt{true}, \texttt{false}\}$ evaluates to `true`, the optimization process will stop and return its results.

Some possible criteria that can be used to decide whether an optimizer should terminate or not are [1975, 1634, 2325, 2326]:

1. The user may grant the optimization algorithm a maximum computation time. If this time has been exceeded, the optimizer should stop. Here we should note that the time needed for single individuals may vary, and so will the times needed for iterations. Hence, this time threshold can sometimes not be abided exactly.
2. Instead of specifying a time limit, a total number of iterations $\hat{t}$ or individual evaluations $\hat{\tau}$ may be specified. Such criteria are most interesting for the researcher, since she often wants to know whether a qualitatively interesting solution can be found for a given problem using at most a predefined number of samples from the problem space.
3. An optimization process may be stopped when no improvement in the solution quality could be detected for a specified number of iterations. Then, the process most probably has converged to a (hopefully good) solution and will most likely not be able to make further progress.
4. If we optimize something like a decision maker or classifier based on a sample data set, we will normally divide this data into a training and a test set. The training set is used to guide the optimization process whereas the test set is used to verify its results. We can compare the performance of our solution when fed with the training set to its properties if fed with the test set. This comparison helps us detect when most probably no further generalization can be achieved by the optimizer and we should terminate the process.
5. Obviously, we can terminate an optimization process if it has already yielded a sufficiently good solution.

In practical applications, we can apply any combination of the criteria above in order to determine when to halt. How the termination criterion is tested in an iterative algorithm is illustrated in Algorithm 1.1.

Algorithm 1.1: Example Iterative Algorithm

Input: [implicit] terminationCriterion(): the termination criterion
Data: t: the iteration counter

```
1 begin
2     t ⟵ 0
      // initialize the data of the algorithm
3     while ¬terminationCriterion() do
          // perform one iteration - here happens the magic
4         t ⟵ t + 1
5 end
```

Minimization

Many optimization algorithms have been developed for single-objective optimization in their original form. Such algorithms may be used for both, minimization or maximization. Without loss of generality we will present them as minimization processes since this is the most commonly used notation. An algorithm that maximizes the function f may be transformed to a minimization using $-f$ instead.

Note that using the prevalence comparisons as introduced in Section 1.2.4 on page 38, multi-objective optimization processes can be transformed into single-objective minimization processes. Therefore $x_1 \succ x_2 \Leftrightarrow \text{cmp}_F(x_1, x_2) < 0$.

Modeling and Simulating

While there are a lot of problems where the objective functions are mathematical expressions that can directly be computed, there exist problem classes far away from such simple function optimization that require complex models and simulations.

Definition 1.45 (Model). A model[43] is an abstraction or approximation of a system that allows us to reason and to deduce properties of the system.

Models are often simplifications or idealization of real-world issues. They are defined by leaving away facts that probably have only minor impact on the conclusions drawn from them. In the area of global optimization, we often need two types of abstractions:

1. The models of the potential solutions shape the problem space $\mathbb{X}$. Examples are
   a) programs in Genetic Programming, for example for the Artificial Ant problem,
   b) construction plans of a skyscraper,
   c) distributed algorithms represented as programs for Genetic Programming,
   d) construction plans of a turbine,
   e) circuit diagrams for logical circuits, and so on.
2. Models of the environment in which we can test and explore the properties of the potential solutions, like
   a) a map on which the Artificial Ant will move which is driven by the evolved program,
   b) an abstraction from the environment in which the skyscraper will be built, with wind blowing from several directions,
   c) a model of the network in which the evolved distributed algorithms can run,
   d) a physical model of air which blows through the turbine,
   e) the model of an energy source the other pins which will be attached to the circuit together with the possible voltages on these pins.

Models themselves are rather static structures of descriptions and formulas. Deriving concrete results (objective values) from them is often complicated. It often makes more sense to bring the construction plan of a skyscraper to life in a simulation. Then we can test the influence of various wind strengths and directions on building structure and approximate the properties which define the objective values.

Definition 1.46 (Simulation). A simulation[44] is the computational realization of a model. Whereas a model describes abstract connections between the properties of a system, a simulation realizes these connections.

Simulations are executable, live representations of models that can be as meaningful as real experiments. They allow us to reason if a model makes sense or not and how certain objects behave in the context of a model.

[43] http://en.wikipedia.org/wiki/Model_%28abstract%29 [accessed 2007-07-03]
[44] http://en.wikipedia.org/wiki/Simulation [accessed 2007-07-03]

1.4 Problems in Optimization

1.4.1 Introduction

The classification of optimization algorithms in Section 1.1.1 and the table of contents of this book enumerate a wide variety of optimization algorithms. Yet, the approaches introduced here resemble only a small fraction of the actual number of available methods. It is a justified question to ask why there are so many different approaches, why is this variety needed? One possible answer is simply because there are so many different kinds of optimization tasks. Each of them puts different obstacles into the way of the optimizers and comes with own, characteristic difficulties.

In this chapter we want to discuss the most important of these complications, the major problems that may be encountered during optimization. Some of subjects in the following text concern global optimization in general (multi-modality and overfitting, for instance), others apply especially to nature-inspired approaches like genetic algorithms (epistasis and neutrality, for example). Neglecting even a single one them during the design or process of optimization can render the whole efforts invested useless, even if highly efficient optimization techniques are applied. By giving clear definitions and comprehensive introductions to these topics, we want to raise the awareness of scientists and practitioners in the industry and hope to help them to use optimization algorithms more efficiently.

In Figure 1.19, we have sketched a set of different types of fitness landscapes (see Section 1.3.2) which we are going to discuss. The objective values in the figure are subject to minimization and the small bubbles represent solution candidates under investigation. An arrow from one bubble to another means that the second individual is found by applying one search operation to the first one.

The Term "Difficult"

Before we go more into detail about what makes these landscapes *difficult*, we should establish the term in the context of optimization. The degree of difficulty of solving a certain problem with a dedicated algorithm is closely related to its *computational complexity*[45], i. e., the amount of resources such as time and memory required to do so. The computational complexity depends on the number of input elements needed for applying the algorithm. This dependency is often expressed in form of approximate boundaries with the Big-**O**-family notations introduced by Bachmann [96] and made popular by Landau [1236]. Problems can further be divided into *complexity classes*. One of the most difficult complexity classes owning to its resource requirements is $\mathcal{NP}$, the set of all decision problems which are solvable in polynomial time by non-deterministic Turing machines [773]. Although many attempts have been made, no algorithm has been found which is able to solve an $\mathcal{NP}$-complete [773] problem in polynomial time on a deterministic computer. One approach to obtaining near-optimal solutions for problems in $\mathcal{NP}$ in reasonable time is to apply metaheuristic, randomized optimization procedures.

As already stated, optimization algorithms are guided by objective functions. A function is *difficult* from a mathematical perspective in this context if it is not continuous, not differentiable, or if it has multiple maxima and minima. This understanding of difficulty comes very close to the intuitive sketches in Figure 1.19.

In many real world applications of metaheuristic optimization, the characteristics of the objective functions are not known in advance. The problems are usually $\mathcal{NP}$ or have

[45] see Section 30.1.3 on page 550

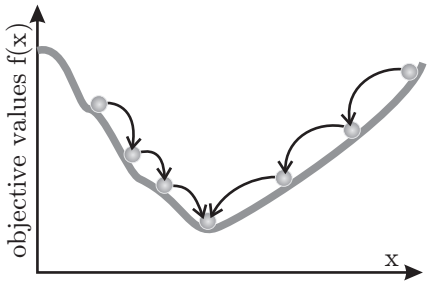

Fig. 1.19.a: Best Case

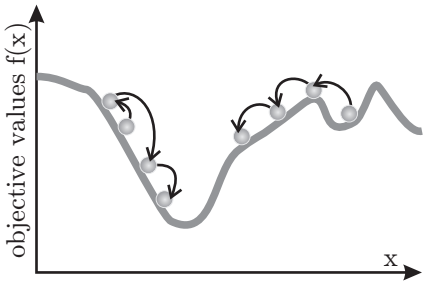

Fig. 1.19.b: Low Variation

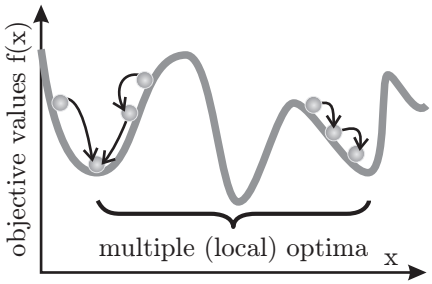

Fig. 1.19.c: Multimodal

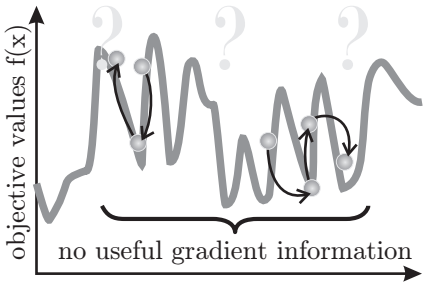

Fig. 1.19.d: Rugged

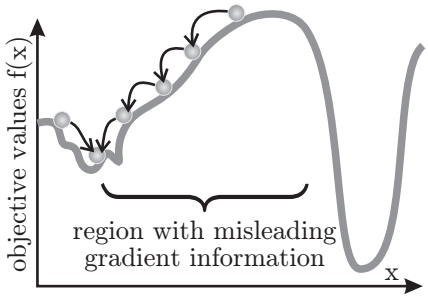

Fig. 1.19.e: Deceptive

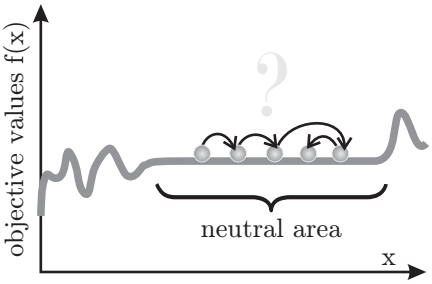

Fig. 1.19.f: Neutral

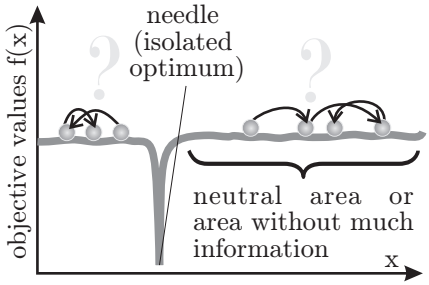

Fig. 1.19.g: Needle-In-A-Haystack

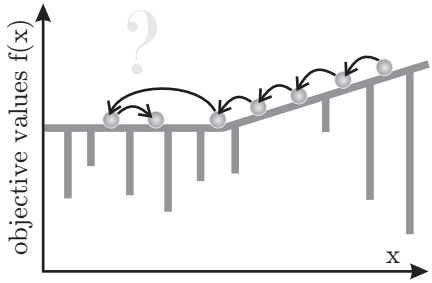

Fig. 1.19.h: Nightmare

Figure 1.19: Different possible properties of fitness landscapes (minimization).

unknown complexity. It is therefore only rarely possible to derive boundaries for the performance or the runtime of optimizers in advance, let alone exact estimates with mathematical precision.

Most often, experience, rules of thumb, and empirical results based on models obtained from related research areas such as biology are the only guides available. In this chapter, we discuss many such models and rules, providing a better understanding of when the application of a metaheuristic is feasible and when not, as well as with indicators on how to avoid defining problems in a way that makes them *difficult*.

1.4.2 Premature Convergence

Introduction

An optimization algorithm has *converged* if it cannot reach new solution candidates anymore or if it keeps on producing solution candidates from a "small"[46] subset of the problem space. Meta-heuristc global optimization algorithms will usually converge at some point in time. In nature, a similar phenomenon can be observed according to [1196]: The *niche preemption principle* states that a niche in a natural environment tends to become dominated by a single species [1347]. One of the problems in global optimization (and basically, also in nature) is that it is often not possible to determine whether the best solution currently known is a situated on local or a global optimum and thus, if convergence is acceptable. In other words, it is usually not clear whether the optimization process can be stopped, whether it should concentrate on refining the current optimum, or whether it should examine other parts of the search space instead. This can, of course, only become cumbersome if there are multiple (local) optima, i. e., the problem is *multimodal* as depicted in Fig. 1.19.c.

A mathematical function is multimodal if it has multiple maxima or minima [1863, 2327, 512]. A set of objective functions (or a vector function) F is multimodal if it has multiple (local or global) optima – depending on the definition of "optimum" in the context of the corresponding optimization problem.

The Problem

An optimization process has *prematurely converged* to a local optimum if it is no longer able to explore other parts of the search space than the area currently being examined *and* there exists another region that contains a superior solution [2075, 1824]. Figure 1.20 illustrates examples for premature convergence.

The existence of multiple global optima itself is not problematic and the discovery of only a subset of them can still be considered as successful in many cases. The occurrence of numerous local optima, however, is more complicated.

Domino Convergence

The phenomenon of *domino convergence* has been brought to attention by Rudnick [1773] who studied it in the context of his BinInt problem [1773, 2036] which is discussed in Section 21.2.5. In principle, domino convergence occurs when the solution candidates have features which contribute to significantly different degrees to the total fitness. If these features are encoded in separate genes (or building blocks) in the genotypes, they are likely to be treated with different priorities, at least in randomized or heuristic optimization methods.

Building blocks with a very strong positive influence on the objective values, for instance, will quickly be adopted by the optimization process (i. e., "converge"). During this time, the alleles of genes with a smaller contribution are ignored. They do not come into play until

[46] according to a suitable metric like numbers of modifications or mutations which need to be applied to a given solution in order to leave this subset

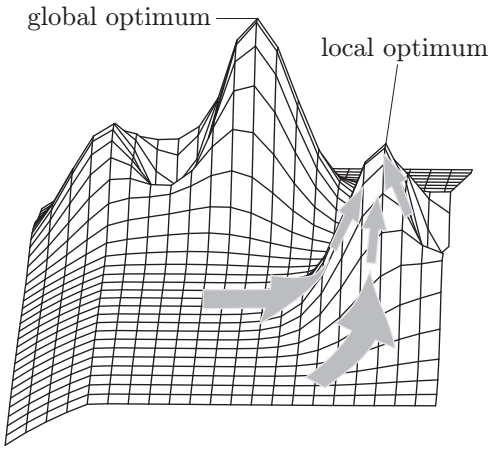

Fig. 1.20.a: Example 1: Maximization

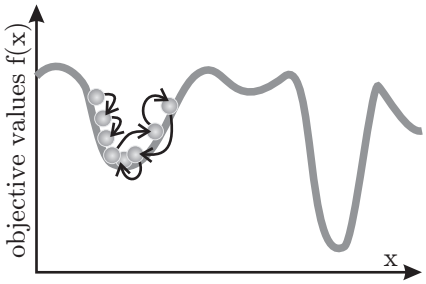

Fig. 1.20.b: Example 2: Minimization

Figure 1.20: Premature convergence in the objective space.

the optimal alleles of the more "important" blocks have been accumulated. Rudnick [1773] called this sequential convergence phenomenon *domino convergence* due to its resemblance to a row of falling domino stones [2036].

Let us consider the application of a genetic algorithm in such a scenario. Mutation operators from time to time destroy building blocks with strong positive influence which are then reconstructed by the search. If this happens with a high enough frequency, the optimization process will never get to optimize the lower salient blocks because repairing and rediscovering those with higher importance takes precedence. Thus, the mutation rate of the EA limits the probability of finding the global optima in such a situation.

In the worst case, the contributions of the less salient genes may almost look like noise and they are not optimized at all. Such a situation is also an instance of premature convergence, since the global optimum which would involve optimal configurations of all building blocks will not be discovered. In this situation, restarting the optimization process will not help because it will always turn out the same way. Example problems which are often likely to exhibit domino convergence are the Royal Road and the aforementioned BinInt problem, which you can find discussed in Section 21.2.4 and Section 21.2.5, respectively.

One Cause: Loss of Diversity

In biology, diversity is *the variety and abundance of organisms at a given place and time* [1598, 1348]. Much of the beauty and efficiency of natural ecosystems is based on a dazzling array of species interacting in manifold ways. Diversification is also a good strategy utilized by investors in the economy in order to increase their wealth.

In population-based global optimization algorithms, maintaining a set of diverse solution candidates is very important as well. Losing diversity means approaching a state where all the solution candidates under investigation are similar to each other. Another term for this state is convergence. Discussions about how diversity can be measured have been provided by Routledge [1771], Cousins [459], Magurran [1348], Morrison and De Jong [1462], Paenke et al. [1598], and Burke et al. [309, 311].

Preserving diversity is directly linked with maintaining a good balance between exploitation and exploration [1598] and has been studied by researchers from many domains, such as

1. Genetic Algorithms [1558, 1750, 1751],
2. Evolutionary Algorithms [253, 254, 1262, 1471, 1943, 1892],

3. Genetic Programming [510, 871, 872, 310, 311, 273],
4. Tabu Search [812, 816], and
5. Particle Swarm Optimization [2226].

Exploration vs. Exploitation

The operations which create new solutions from existing ones have a very large impact on the speed of convergence and the diversity of the populations [637, 1910]. The step size in Evolution Strategy is a good example of this issue: setting it properly is very important and leads over to the "exploration versus exploitation" problem [940] which can be observed in other areas of global optimization as well.[47]

In the context of optimization, *exploration* means finding new points in areas of the search space which have not been investigated before. Since computers have only limited memory, already evaluated solution candidates usually have to be discarded in order to accommodate the new ones. Exploration is a metaphor for the procedure which allows search operations to find novel and maybe better solution structures. Such operators (like mutation in evolutionary algorithms) have a high chance of creating inferior solutions by destroying good building blocks but also a small chance of finding totally new, superior traits (which, however, is not guaranteed at all).

Exploitation, on the other hand, is the process of improving and combining the traits of the currently known solutions, as done by the crossover operator in evolutionary algorithms, for instance. Exploitation operations often incorporate small changes into already tested individuals leading to new, very similar solution candidates or try to merge building blocks of different, promising individuals. They usually have the disadvantage that other, possibly better, solutions located in distant areas of the problem space will not be discovered.

Almost all components of optimization strategies can either be used for increasing exploitation or in favor of exploration. Unary search operations that improve an existing solution in small steps can often be built, hence being exploitation operators. They can also be implemented in a way that introduces much randomness into the individuals, effectively making them exploration operators. Selection operations[48] in Evolutionary Computation choose a set of the most promising solution candidates which will be investigated in the next iteration of the optimizers. They can either return a small group of best individuals (exploitation) or a wide range of existing solution candidates (exploration).

Optimization algorithms that favor exploitation over exploration have higher convergence speed but run the risk of not finding the optimal solution and may get stuck at a local optimum. Then again, algorithms which perform excessive exploration may never improve their solution candidates well enough to find the global optimum or it may take them very long to discover it "by accident". A good example for this dilemma is the Simulated Annealing algorithm discussed in Chapter 12 on page 263. It is often modified to a form called *simulated quenching* which focuses on exploitation but loses the guaranteed convergence to the optimum. Generally, optimization algorithms should employ at least one search operation of explorative character and at least one which is able to exploit good solutions further. There exists a vast body of research on the trade-off between exploration and exploitation that optimization algorithms have to face [638, 945, 622, 1494, 49, 538].

Countermeasures

There is no general approach which can prevent premature convergence. The probability that an optimization process gets caught in a local optimum depends on the characteristics of the problem to be solved and the parameter settings and features of the optimization algorithms applied [2051, 1775].

[47] More or less synonymously to exploitation and exploration, the terms *intensifications* and *diversification* have been introduced by Glover [812, 816] in the context of Tabu Search.

[48] Selection will be discussed in Section 2.4 on page 121.