Chapter 2
Evolution Strategies

2.1 Introduction

Many real-world problems have multiple local optima. Such problems are called multimodal optimization problems and are usually difficult to solve. Local search methods, i.e., methods that greedily improve solutions based on search in the neighborhood of a solution, often only find an arbitrary local optimum that may not be the global one. The most successful methods in global optimization are based on stochastic components, which allow escaping from local optima and overcome premature stagnation. A famous class of global optimization methods are ES. They are exceptionally successful in continuous solution spaces. ES belong to the most famous evolutionary methods for blackbox optimization, i.e., for optimization scenarios, where no functional expressions are explicitly given and no derivatives can be computed.

ES imitate the biological principle of evolution [1] and can serve as an excellent introduction to learning and optimization. They are based on three main mechanisms oriented to the process of Darwinian evolution, which led to the development of all species. Evolutionary concepts are translated into algorithmic operators, i.e., recombination, mutation, and selection.

First, we define an optimization problem formally. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be the fitness function to be minimized in the space of solutions $\mathbb{R}^d$. The problems we consider in this work are minimization problems unless explicitly stated, i.e., high fitness corresponds to low fitness function values. The task is to find a solution $\mathbf{x}^* \in \mathbb{R}^d$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^d$. A desirable property of an optimization method is to find the optimum $\mathbf{x}^*$ with fitness $f(\mathbf{x}^*)$ within a finite and preferably low number of function evaluations. Problem f can be an arbitrary optimization problem. However, we concentrate on continuous ones.

This chapter is structured as follows. Section 2.2 gives short introduction to the basic principles of EAs. The history of evolutionary computation is sketched in Sect. 2.3. The evolutionary operators are presented in the following sections, i.e., recombination in Sect. 2.4, mutation in Sect. 2.5, and selection in Sect. 2.6,

O. Kramer, *Machine Learning for Evolution Strategies*,
Studies in Big Data 20, DOI 10.1007/978-3-319-33383-0_2

respectively. Step size control is an essential part of the success of EAs and is introduced in Sect. 2.7 with Rechenberg's rule. As the (1+1)-ES has an important part to play in this book, Sect. 2.8 is dedicated to this algorithmic variant. The chapter closes with conclusions in Sect. 2.9.

2.2 Evolutionary Algorithms

If derivatives are available, Newton methods and variants are the proper algorithmic choices. From this class of methods, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [2] belongs to the state-of-the-art techniques in optimization. In this book, we concentrate on blackbox optimization problems. In blackbox optimization, the problem does not have to fulfill any assumptions or limiting properties. For such general optimization scenarios, evolutionary methods are a good choice. EAs belong to the class of stochastic derivative-free optimization methods. Their biological motivation has made them very popular. They are based on recombination, mutation, and selection. After decades of research, a long history of applications and theoretical investigations have proven the success of evolutionary optimization algorithms.

Algorithm 1 EA

initialize $\mathbf{x}_1, \ldots, \mathbf{x}_\mu$
repeat
  for $i = 1$ **to** λ **do**
    select ρ parents
    recombination $\rightarrow \mathbf{x}'_i$
    mutate $\mathbf{x}'_i$
    evaluate $\mathbf{x}'_i \rightarrow f(\mathbf{x}'_i)$
  end for
  select μ parents from $\{\mathbf{x}'_i\}_{i=1}^{\lambda} \rightarrow \{\mathbf{x}_i\}_{i=1}^{\mu}$
until termination condition

In the following, we shortly sketch the basic principles of EAs oriented to Algorithm 1. Evolutionary search is based on a set $\{\mathbf{x}_1, \ldots, \mathbf{x}_\mu\}$ of parental and a set $\{\mathbf{x}_1, \ldots, \mathbf{x}_\lambda\}$ of offspring candidate solutions, also called individuals. The solutions are iteratively subject to changes and selection of the best offspring candidates. In the generational loop, λ offspring solutions are generated. For each offspring solution, the recombination operator selects ρ parents and combines their parts to a new candidate solution. The mutation operator adds random changes, i.e. noise, to the preliminary candidate resulting in solution $\mathbf{x}'_i$. Its quality in solving the optimization problem is called fitness and is evaluated on fitness function $f(\mathbf{x}'_i)$. All candidate solutions of a generation are put into offspring population $\{\mathbf{x}'_i\}_{i=1}^{\lambda}$. At the

end of a generation, μ solutions are selected and constitute the novel parental population $\{\mathbf{x}_i\}_{i=1}^{\mu}$ that is basis of the following generation.

The optimization process is repeated until a termination condition is reached. Typical termination conditions are defined via fitness values or via an upper bound on the number of generations. In the following, we will shortly present the history of evolutionary computation, introduce evolutionary operators, and illustrate concepts that have proven well in ES.

2.3 History

In the early 1950s, the idea came up to use algorithms for problem solving that are oriented to the concept of evolution. In Germany, the history of evolutionary computation began with ES, which were developed by Rechenberg and Schwefel in the sixties and seventies of the last century in Berlin [3–5]. At the same time, Holland introduced the evolutionary computation concept in the United States known as genetic algorithms [6]. Also Fogel introduced the idea at that time and called this approach evolutionary programming [7]. For about 15 years, the disciplines developed independently from each other before growing together in the 1980s. Another famous branch of evolutionary computation was proposed in the nineties of the last century, i.e., genetic programming (GP) [8]. GP is about evolving programs by means of evolution. These programs can be based on numerous programming concepts and languages, e.g., assembler programs or data structures like trees. Genetic programming operators are oriented to similar principles like other EAs, but adapted to evolving programs. For example, recombination combines elements of two or more programs. In tree representations, subtrees are exchanged. Mutation changes a program. In assembler code, a new command may be chosen. In tree representations, a new subtree can be generated. Mutation can also lengthen or shorten programs.

Advanced mutation operators, step size mechanisms, and methods to adapt the covariance matrix like the CMA-ES [9] have made ES one of the most successful optimizers in derivative-free continuous optimization. For binary, discrete, and combinatorial representations, other concepts are known. Annual international conferences like the Genetic and Evolutionary Computation Conference (GECCO), the Congress on Evolutionary Computation (CEC), and EvoStar in Europe contribute to the understanding and distribution of EAs as solid concepts and search methods.

Related to evolutionary search are estimation of distribution algorithms (EDAs) and particle swarm optimization (PSO) algorithms. Both are based on randomized operators like EAs, while PSO algorithms are also nature-inspired. PSO models the flight of solutions in the solution space with velocities, while being oriented to the best particle positions. All nature-inspired methods belong to the discipline computational intelligence, which also comprises neural networks and fuzzy-logic. Neural networks are inspired by natural neural processing, while fuzzy logic is a logic inspired by the fuzzy way of human language and concepts.

2.4 Recombination

Recombination, also known as crossover, mixes the genetic material of parents. Most evolutionary algorithms also make use of a recombination operator that combines the information of two or more candidate solutions $\mathbf{x}_1, \ldots, \mathbf{x}_\rho$ to a new offspring solution. Hence, the offspring carries parts of the genetic material of its parents. Many recombination operators are restricted to two parents, but also multi-parent recombination variants have been proposed in the past that combine information of ρ parents. The use of recombination is discussed controversially within the building block hypothesis by Goldberg [10], Holland [11]. The building block hypothesis assumes that good solution substrings called building blocks are spread over the population in the course of the evolutionary process, while their number increases.

For bit strings and similar representations, multi-point crossover is a common recombination operator. It splits up the representations of two ore more parents at multiple positions and combines the parts alternately to a new solution.

Typical recombination operators for continuous representations are dominant and intermediate recombination. Dominant recombination randomly combines the genes of all parents. With ρ parents $\mathbf{x}_1, \ldots, \mathbf{x}_\rho \in \mathbb{R}^d$, it creates the offspring solution $\mathbf{x}' = (x'_1, \ldots, x'_d)^T$ by randomly choosing the i-th component

$$x'_i = (x_i)_j, \quad j \in \text{ random } \{1, \ldots, \rho\}. \tag{2.1}$$

Intermediate recombination is appropriate for numerical solution spaces. Given ρ parents $\mathbf{x}_1, \ldots, \mathbf{x}_\rho$ each component of the offspring vector $\mathbf{x}'$ is the arithmetic mean of the components of all ρ parents

$$x'_i = \frac{1}{\rho} \sum_{j=1}^{\rho} (x_i)_j. \tag{2.2}$$

The characteristics of offspring solutions lie between their parents. Integer representations may require rounding procedures for generating valid solutions.

2.5 Mutation

Mutation is the second main source of evolutionary changes. The idea of mutation is to add randomness to the solution. According to Beyer and Schwefel [3], a mutation operator is supposed to fulfill three conditions. First, from each point in the solution space each other point must be reachable. This condition shall guarantee that the optimum can be reached in the course of the optimization run. Second, in unconstrained solution spaces a bias is disadvantageous, because the direction to the optimum is unknown. By avoiding a bias, all directions in the solution space can be reached with the same probability. This condition is often hurt in practical

optimization to accelerate the search. Third, the mutation strength should be adjustable, in order to adapt exploration and exploitation to local solution space conditions, e.g., to accelerate the search when the success probability is high.

For bit string representations, the bit-flip mutation operator is usual. It flips each bit, i.e., changes a 0 to 1 and a 1 to 0 with probability $1/d$, if d is the number of bits. In $\mathbb{R}^d$ the Gaussian mutation operator is common. Let $\mathcal{N}(0, 1)$ represent a randomly drawn Gaussian distributed number with expectation 0 and standard deviation 1. Mutation adds Gaussian noise to each solution candidate using

$$\mathbf{x}' = \mathbf{x} + \mathbf{z}, \tag{2.3}$$

with a mutation vector $\mathbf{z} \in \mathbb{R}^d$ based on sampling

$$\mathbf{z} \sim \sigma \cdot \mathcal{N}(0, 1). \tag{2.4}$$

The standard deviation σ plays the role of the mutation strength and is also known as step size. The isotropic Gaussian mutation with only one step size uses the same standard deviation for each component x_i. The convergence towards the optimum can be improved by adapting σ according to local solution space characteristics. In case of high success rates, i.e., a large number of offspring solutions being better than their parents, big step sizes are advantageous, in order to explore the solution space as fast as possible. This is often reasonable at the beginning of the search. In case of low success rates, smaller step sizes are appropriate. This is often adequate in later phases of the search during convergence to the optimum, i.e., when good evolved solutions should not be destroyed. An example for an adaptive control of step sizes is the 1/5-th success rule by Rechenberg [4] that increases the step size, if the success rate is over 1/5-th, and decreases it, if the success rate is lower. The Rechenberg rule will be introduced in more detail in Sect. 2.7.

2.6 Selection

The counterpart of the variation operators mutation and recombination is selection. ES usually do not employ a competitive selection operator for mating selection. Instead, parental solutions are randomly drawn from the set of candidate solutions. But for survivor selection, the elitist selection strategies comma and plus are used. They choose the μ-best solutions as basis for the parental population of the following generation. Both operators, plus and comma selection, can easily be implemented by sorting the population with respect to the solutions' fitness. Plus selection selects the μ-best solutions from the union $\{\mathbf{x}_i\}_{i=1}^{\mu} \cup \{\mathbf{x}'_i\}_{i=1}^{\lambda}$ of the last parental population $\{\mathbf{x}_i\}_{i=1}^{\mu}$ and the current offspring population $\{\mathbf{x}'_i\}_{i=1}^{\lambda}$, and is denoted as $(\mu+\lambda)$-ES. In contrast, comma selection, i.e. (μ, λ)-ES, selects exclusively from the offspring population, neglecting the parental population, even if the parents have a superior

fitness. Forgetting superior solutions may sound irrational. But good solutions can be only local optima. The evolutionary process may fail to leave them without the ability to forget.

2.7 Rechenberg's 1/5th Success Rule

The adjustment of parameters and adaptive operator features is of crucial importance for reliable results and the efficiency of evolutionary heuristics. Furthermore, proper parameter settings are important for the comparison of different algorithms. The problem arises how evolutionary parameters can be tuned and controlled. The change of evolutionary parameters during the run is called online parameter control and is reasonable, if the conditions of the fitness landscape change during the optimization process. In deterministic control, parameters are adjusted according to a fixed time scheme, e.g., depending on the generation number like proposed by Fogarty [12] and by Bäck and Schütz [13]. However, it may be useful to reduce the mutation strengths during the evolutionary search, in order to allow convergence. For example, running a (1+1)-ES with isotropic Gaussian mutations with constant step sizes σ, the optimization process will become slow after a certain number of generations. A deterministic scheme may fail to hit the optimal speed the step sizes are reduced and may also not be able to increase them if necessary.

The step sizes have to be adapted in order to speed up the optimization process in a more flexible way. A powerful scheme is the 1/5th success rule by Rechenberg [4], which adapts step size σ for optimal progress. In case of the (1+1)-ES, the optimization process runs for a number T of generations. During this period, step size σ is kept constant and the number T_s of successful generations is counted. From T_s, the success probability p_s is estimated by

$$p_s = T_s / T \tag{2.5}$$

and step size σ is changed according to

$$\sigma = \begin{cases} \sigma/\tau, & \text{if } p_s > 1/5 \\ \sigma \cdot \tau, & \text{if } p_s < 1/5 \\ \sigma, & \text{if } p_s = 1/5 \end{cases}$$

with $0 < \tau < 1$. This control of the step size is implemented in order adapt to local solution space characteristics and to speed up the optimization process in case of large success probabilities. Figure 2.1 illustrates the Rechenberg rule with $T = 5$. The fitness is increasing from left to right. The blue candidate solutions are the successful solutions of a (1+1)-ES, the grey ones are discarded due to worse fitness. Seven mutation steps are shown, at the beginning with a small step size, illustrated by the smaller light circles. After five mutations, the step size is increased, as the success rate is larger than 1/5, i.e. $p_s = 3/5$, illustrated by larger dark circles. Bigger steps

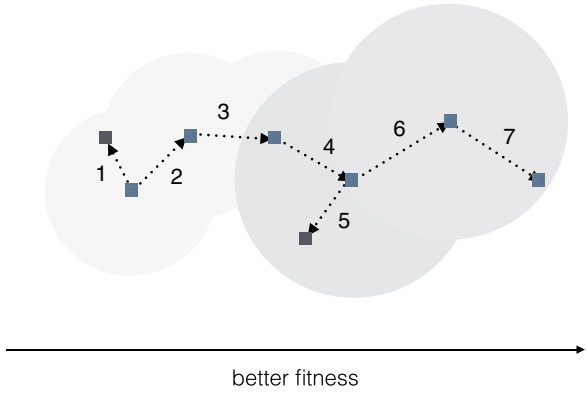

Fig. 2.1 Illustration of step size adaptation with Rechenberg's success rule. Three of five steps are successful resulting in a success probability of $p_s = 3/5$ and an increase of step size σ

into the direction of the optimum are possible. The objective of Rechenberg's step size adaptation is to stay in the evolution window guaranteeing optimal progress. The optimal value for τ depends on various factors such as the number T of generations and the dimension d of the problem.

A further successful concept for step size adaptation is self-adaptation, i.e., the automatic evolution of the mutation strengths. In self-adaptation, each candidate is equipped with an own step size, which is subject to recombination and mutation. Then, the objective variable is mutated with the inherited and modified step size. As solutions consist of objective variables and step sizes, the successful ones are selected as parents for the following generation. The successful step sizes are spread over the population.

2.8 (1+1)-ES

The (1+1)-ES with Gaussian mutation and Rechenberg's step size control is the basis of the evolutionary algorithms used in this book. We choose this method to reduce side effects. The more complex algorithms are, the more probable are side effects changing the interactions with machine learning extensions. We concentrate on the (1+1)-ES, which is well understood from a theoretical perspective. Algorithm 2 shows the pseudocode of the (1+1)-ES. After initialization of $\mathbf{x}$ in $\mathbb{R}^d$, the evolutionary loop begins. The solution $\mathbf{x}$ is mutated to $\mathbf{x}'$ with Gaussian mutation and step size σ that is adapted with Rechenberg's 1/5th rule. The new solution $\mathbf{x}'$ is accepted, if its fitness is better than or equal to the fitness of its parent $\mathbf{x}$, i.e., if $f(\mathbf{x}') \leq f(\mathbf{x})$. Accepting the solution in case of equal fitness is reasonable to allow random walk on plateaus, which are regions in solution space with equal fitness. For the (1+1)-ES and all variants used in the remainder of this book, the fitness for each new solution $\mathbf{x}'$ is computed only once, although the condition in Line 6 might suggest another fitness function call is invoked.

Algorithm 2 (1+1)-ES

1: intialize $\mathbf{x}$
2: **repeat**
3: mutate $\mathbf{x}' = \mathbf{x} + \mathbf{z}$ with $\mathbf{z} \sim \sigma \cdot \mathcal{N}(0, 1)$
4: adapt σ with Rechenberg
5: evaluate $\mathbf{x}' \rightarrow f(\mathbf{x}')$
6: replace $\mathbf{x}$ with $\mathbf{x}'$ if $f(\mathbf{x}') \leq f(\mathbf{x})$
7: **until** termination condition

The evolutionary loop is repeated until a termination condition is reached. The (1+1)-ES shares similarities with simulated annealing, but employs Gaussian mutation with step size adaptation, while not using the cooling scheme for accepting a worse solution. For multimodal optimization, restarts may be required as the diversity of the (1+1)-ES without a population is restricted to only one single solution. The (1+1)-ES is the basic algorithm for the optimization approaches introduced in Chaps. 3, 4, 7, 9, and 10.

2.9 Conclusions

Evolutionary algorithms are famous blackbox optimization algorithms. They are based on a population of candidate solutions or one single solution in case of the (1+1)-ES. Evolutionary optimization algorithms are inspired by the idea of natural selection and Darwinian evolution. They have their roots in the fifties and sixties of the last century. Meanwhile, ES have developed to outstandingly successful blackbox optimization methods for continuous solution spaces. For exploration of the solution space, recombination operators combine the properties of two or more solutions. Mutation operators use noise to explore the solution space. Selection exploits the search by choosing the best solutions to be parents for the following generation. Parameter control techniques like Rechenberg's 1/5th success rule improve the convergence towards the optimum. In case of success, larger steps can be taken in solution space, while in case of stagnation, progress is more probable when the search concentrates on the close environment of the current solution.

Extensions allow ES to search in constrained or multi-objective solution spaces. In multi-objective optimization, the task is to evolve a Pareto set of non-dominated solutions. Various strategies are available for this sake. An example is NSGA-ii [14] that maximizes the Manhattan distance of solutions in objective space to achieve a broad coverage on the Pareto front. Theoretical results are available for discrete and continuous algorithmic variants and solution spaces. An example for a simple result is the runtime of a (1+1)-EA with bit flip mutation on the function OneMax that maximizes the number of ones in a bit string. The expected runtime is upper bound by $O(d \log d)$, if d is the length of the bit string [15]. The idea of the proof is

based on the lemma for fitness-based partitions, which divides the solution space into disjoint sets of solutions with equal fitness and makes assertions about the expected time required to leave these partitions.

This book will concentrate on extensions of ES with machine learning methods to accelerate and support the search. The basic mechanisms of ES will be extended by covariance matrix estimation, fitness function surrogates, constraint function surrogates, and dimensionality reduction approaches for optimization, visualization, and niching.

References

1. Kramer, O., Ciaurri, D.E., Koziel, S.: Derivative-free optimization. In: Computational Optimization and Applications in Engineering and Industry. Springer (2011)
2. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer (2000)
3. Beyer, H., Schwefel, H.: Evolution strategies—A comprehensive introduction. Nat. Comput. **1**(1), 3–52 (2002)
4. Rechenberg, I.: Evolutionsstrategie—Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart (1973)
5. Schwefel, H.-P.: Numerische Optimierung von Computer-Modellen mittel der Evolutionsstrategie. Birkhaeuser, Basel (1977)
6. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
7. Fogel, D.B.: Evolving artificial intelligence. PhD thesis, University of California, San Diego (1992)
8. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
9. Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: International Conference on Evolutionary Computation, pp. 312–317 (1996)
10. Goldberg, D.: Genetic Algorithms in Search. Optimization and Machine Learning. Addison-Wesley, Reading, MA (1989)
11. Holland, J.H.: Hidden Order: How Adaptation Builds Complexity. Addison-Wesley, Reading, MA (1995)
12. Fogarty, T.C.: Varying the probability of mutation in the genetic algorithm. In: Proceedings of the 3rd International Conference on Genetic Algorithms, pp. 104–109. Morgan Kaufmann Publishers Inc, San Francisco (1989)
13. Bäck, T., Schüz, M.: Intelligent mutation rate control in canonical genetic algorithms. In: Proceedings of the 9th International Symposium on Foundation of Intelligent Systems, ISMIS 1996, pp. 158–167. Springer (1996)
14. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II. In: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, PPSN VI 2000, pp. 849–858. Paris, France, 18–20 Sept 2000
15. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoret. Comput. Sci. **276**(1–2), 51–81 (2002)