Chapter 3
Parameter Control

3.1 Introduction

Parameter control is an essential aspect of successful evolutionary search. Various parameter control and tuning methods have been proposed in the history of evolutionary computation, cf. Fig. 3.1 for a short taxonomy. The importance of parameter control has become famous for mutation rates. Mutation is a main source of evolutionary changes. Mutation rates control the magnitude of random changes of solutions. At the beginning of the history of evolutionary computation, researchers argued about proper settings. De Jong's [1] recommendation was the mutation strength $\sigma = 0.001$, Schaffer et al. [2] recommended the setting $0.005 \leq \sigma \leq 0.01$, and Grefenstette [3] $\sigma = 0.01$. Mühlenbein [4] suggested to set the mutation probability to $\sigma = 1/N$ depending on the length N of the representation. But early, the idea appeared to control the mutation rate during the optimization run, as the optimal rate might change during the optimization process, and different rates are reasonable for different problems. Objective of this chapter is to compare the parameter tuning and control techniques of a simple evolutionary algorithm (EA) on a simple function, i.e., *OneMax*, to allow insights into the interplay of mutation rates and parameter control mechanisms. *OneMax* is a maximization problem defined on $\{0, 1\}^N \rightarrow \mathbb{N}$ that counts the number of ones in bit string $\mathbf{x}$

$$OneMax(\mathbf{x}) = \sum_{i=1}^{N} x_i . \tag{3.1}$$

The optimal solution is $\mathbf{x}^* = (1, \ldots, 1)^T$ with fitness $f(\mathbf{x}) = N$.

O. Kramer, *A Brief Introduction to Continuous Evolutionary Optimization*,
SpringerBriefs in Computational Intelligence,
DOI: 10.1007/978-3-319-03422-5_3, © The Author(s) 2014

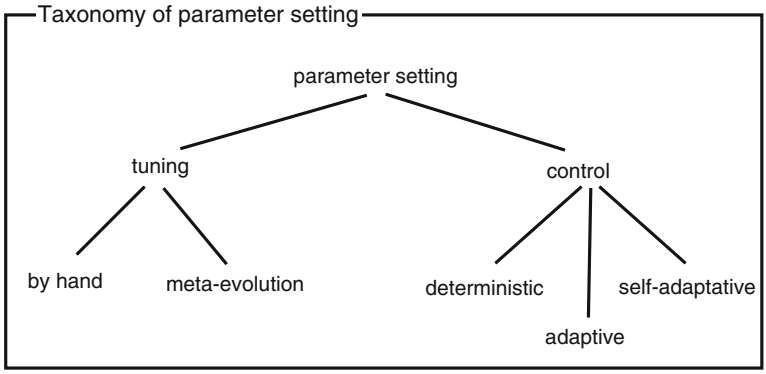

Fig. 3.1 Taxonomy of parameter setting of this work oriented to Eiben et al. [5] and complemented on the parameter tuning branch (cf. Kramer [6])

3.2 The (1+1)-EA

The $(1 + 1)$-EA works on bit string representations $\mathbf{x} = (x_1, \ldots, x_N)^T \in \{0, 1\}^N$ with only one individual, which is changed with bit-flip mutation. Bit-flip mutation means that each bit x_i of bit-string $\mathbf{x}$ is flipped with probability $\sigma = 1/N$. No recombination is employed, as no population is used. Furthermore, the selection operator can be reduced to a simple selection of the better one of two solutions. The pseudocode can be found in Algorithm 1. The number of fitness function calls of a (1+1)-EA complies with the number of generations.

Algorithm 1 Standard $(1 + 1)$-EA

1: choose $\mathbf{x} \in \{0, 1\}^N$ uniform at random
2: **repeat**
3: produce $\mathbf{x}'$ by flipping each bit of $\mathbf{x}$ with probability $1/N$
4: replace $\mathbf{x}$ with $\mathbf{x}'$ if $f(\mathbf{x}') \leq f(\mathbf{x})$
5: **until** termination condition

For the $(1 + 1)$-EA, a runtime analysis on the simple *OneMax* problem demonstrates its properties. The runtime analysis is based on the method of fitness-based partitions, and shows that the $(1 + 1)$-EA's runtime is upper bounded by $O(N \log N)$ on *OneMax* [7].

Theorem 3.1 *The expected runtime of a* $(1 + 1)$*-EA on OneMax is* $O(N \log N)$.

The solution space $\{0, 1\}^N$ is divided into $N + 1$ sets $A_0, \ldots, A_N$. A partition A_i contains all solution with $OneMax(\mathbf{x}) = i$. If the currently best solution $\mathbf{x}$ belongs to A_{N-k}, still k 0-bits have to be flipped leading to improvements. The probability for

another bit not to be flipped is $1 - \frac{1}{N}$, i.e., the probability that the ones are not flipped is $(1 - \frac{1}{N})^{(N-k)}$. Hence, the probability for success is at least $\frac{k}{N}(1 - \frac{1}{N})^{(N-k)} \geq \frac{k}{eN}$ for the next step. The expected runtime is upper bounded by eN/k. For different values of k, we get

$$\sum_{k=1}^{N} \frac{eN}{k} = eN \cdot \sum_{k=1}^{N} \frac{1}{k} = O(N \log N). \tag{3.2}$$

□

In the remainder of this chapter, we will experimentally analyze and compare a selection of important parameter control and tuning techniques.

3.3 A Study on Mutation Rates

The question comes up, if our experiments can confirm the theoretical result, i.e., if the mutation rate $1/N$ leads to $N \log N$ generations in average. For this sake, we test the $(1 + 1)$-EA with various mutation rates on *OneMax* with various problem sizes. This extensive analysis is similar to tuning by hand, which is probably the most frequent parameter tuning method. Figure 3.2 shows the analysis with problem sizes $N = 10, 20$, and 30. The results show that the optimal mutation rate is close to $1/N$, which leads to the runtime of $O(N \log N)$. Our experiments confirm this result with the exception of a multiplicative constant, i.e., the runtime is about two times higher than $N \log N$. In the the following section, we employ evolutionary computation to search for optimal mutation rates, an approach called meta-evolution.

3.4 Meta-Evolution

Meta-evolution is a parameter tuning method that employs evolutionary computation to tune evolutionary parameters. The search for optimal parameters is treated as optimization problem. We employ a $(\mu + \lambda)$-ES [8] to tune the mutation rate of an inner $(1 + 1)$-EA. The $(\mu + \lambda)$-ES employs arithmetic recombination and isotropic

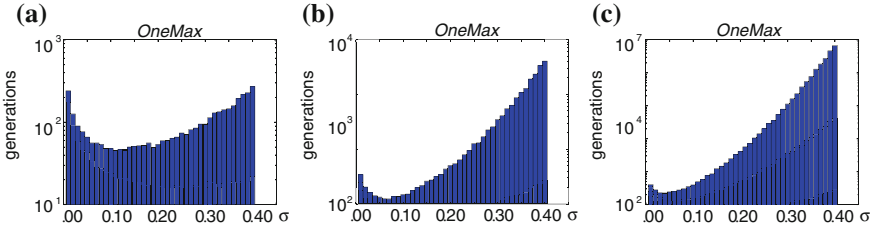

Fig. 3.2 Analysis of mutation strength σ for $(1 + 1)$-EA on *OneMax* for three problem sizes. **a** $(1 + 1)$-EA, N = 10, **b** $(1 + 1)$-EA, N = 20, **c** $(1 + 1)$-EA, N = 30

Table 3.1 Experimental results of meta-evolutionary approach of a (10 + 100)-EA tuning the mutation rates of a (1 + 1)-EA on *OneMax*

N	t	σ^*	Gen.
5	8.80	0.252987	37
10	31.84	0.134133	14
20	90.92	0.071522	42
30	170.60	0.055581	41

Gaussian mutation $\mathbf{x}' = \mathbf{x} + \mathcal{N}(0, \sigma)$ with a decreasing σ depending on generation t. Algorithm 2 shows the pseudocode of the meta-evolutionary approach.

Algorithm 2 Meta-(1 + 1)-EA

```
initialize mutation rates σ_1, ..., σ_μ ∈ P, τ
repeat
  for i = 1 to λ do
    select ρ parents from P
    create σ_i by recombination
    decrease τ
    mutate σ_i = σ_i + τ · N(0, 1)
    run (1 + 1)-EA with σ_i
    add σ_i to P'
  end for
  select μ parents from P' → P
until termination condition
```

In our experimental analysis, we employ a (10+100)-ES optimizing the mutation rate of the underlying (1 + 1)-EA that solves problem *OneMax* for various problem sizes N. The ES starts with an initial mutation rate of $\tau = 0.2$. In each generation, τ is decreased deterministically by multiplication, i.e., $\tau = \tau \cdot 0.95$. The inner (1 + 1)-EA employs the evolved mutation rate σ of the upper ES and is run 25 times with this setting. The average number of generations until the optimum of *OneMax* is found employing the corresponding σ is the fitness $f(\sigma)$. The ES terminates after 50 generations. Table 3.1 shows the experimental results of the meta-evolutionary approach. The table shows the average number t of generations until the optimum has been found by the (1+1)-EA in the last generation of the ES, the evolved mutation rate σ^* and the number of generations, the ES needed to find σ^*. The achieved speed of convergence by the inner (1 + 1)-EA, e.g., 170.6 generations for $N = 30$ is a fast result.

3.5 Rechenberg's 1/5th Rule

An example for an adaptive control of endogenous strategy parameters is the 1/5th success rule for ES by Rechenberg [9]. The idea of Rechenberg's 1/5th rule is to increase the mutation rate, if the success probability is larger than 1/5th, and to

decrease it, if the success probability is smaller. The success probability can be measured w.r.t. a fix number G of generations. If the number of successful generations, i.e., the offspring employs a better fitness than the parent, of a $(1+1)$-EA is g, then g/G is the success rate. If $g/G > 1/5$, σ is increased by $\sigma = \sigma \cdot \tau$ with $\tau > 1$, otherwise, it is decreased by $\sigma = \sigma/\tau$. Algorithm 3 shows the pseudocode of the $(1+1)$-EA with Rechenberg's 1/5th rule. The objective is to stay in the so called *evolution window* guaranteeing nearly optimal progress.

Algorithm 3 $(1+1)$-EA with Rechenberg's 1/5th rule

```
choose x ∈ {0,1}^N uniform at random
repeat
  for i = 1 to G do
    produce x' by flipping each bit of x with probability σ
    replace x with x' if f(x') ≤ f(x) and set g+ = 1
  end for
  if g/G > 1/5 then
    σ = σ · τ
  else
    σ = σ/τ
  end if
until termination condition
```

Figure 3.3 shows the corresponding experimental results for various values of τ and $N = 10, 20$, and 30. The results show that Rechenberg's rule is able to automatically tune the mutation rate and reach almost as good results as the runs with tuned settings. We can observe that smaller settings for τ, i.e., settings close to 1.0 achieve better results than larger settings in all cases. Further experiments have shown that settings over $\tau > 10.0$ lead to very long runtimes (larger than 10^5 generations). In such cases, σ cannot be fine-tuned to allow a fast approximation of the optimum.

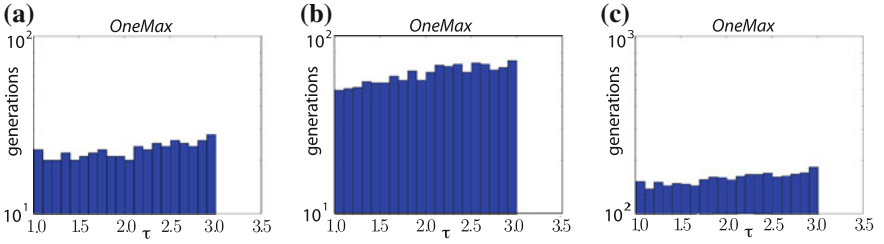

Fig. 3.3 Experimental results of parameter control with Rechenberg's 1/5th rule. **a** Rechenberg, N = 5, **b** Rechenberg, N = 10, **c** Rechenberg, N = 20

3.6 Self-Adaptation

Self-adaptation is an automatic evolutionary mutation rate control. It was originally introduced by Rechenberg and Schwefel [10] for ES, later independently in the United States by Fogel [11] for evolutionary programming. The most successful self-adaptively controlled parameters are mutation parameters. This is a consequence of the direct influence of the mutation operator on the exploration behavior of the optimization algorithm: Large mutation strengths cause large changes of solutions, decreasing mutation strengths allow an approximation of the optimum, in particular in continuous solution spaces.

The mutation rate σ is added to each individual $\mathbf{x}$ and is at the same time subject to recombination, mutation and selection. For a $(1 + 1)$-EA, self-adaptation means that the mutation rate σ is mutated with log-normal mutation

$$\sigma' = \sigma \cdot e^{\tau \mathcal{N}(0,1)} \tag{3.3}$$

with a control parameter τ. Afterwards, the mutation operator is applied. Appropriate mutation rates are inherited and employed in the following generation. The log-normal mutation allows an evolutionary search in the space of strategy parameters. It allows the mutation rates to scale in a logarithmic kind of way from values close to zero to infinity. Algorithm 4 shows the pseudocode of the SA-$(1 + 1)$-EA with σ-self-adaptation.

Algorithm 4 SA-$(1 + 1)$-EA

1: choose $\mathbf{x} \in \{0, 1\}^N$ uniform at random
2: choose $\sigma \in \{0, 1\}$ at random
3: **repeat**
4: produce $\sigma' = \sigma \cdot e^{\tau \mathcal{N}(0,1)}$
5: produce $\mathbf{x}'$ by flipping each bit of $\mathbf{x}$ with probability σ'
6: replace $\mathbf{x}$ with $\mathbf{x}'$ and σ with σ', if $f(\mathbf{x}') \leq f(\mathbf{x})$
7: **until** termination condition

Figure 3.4 shows typical developments[1] of fitness $f(\mathbf{x})$ and mutation rate σ of the SA-$(1 + 1)$-EA on $N = 10, 50$, and 100 for $\tau = 0.1$. Due to the plus selection scheme, the fitness is decreasing step by step. The results show that the mutation rate σ is adapting during the search. In particular, in the last phase of the search for $N = 100$, σ is fast adapting to the search conditions and accelerates the search significantly.

Table 3.2 shows the experimental results of the SA-(1+1)-EA with various settings for τ on *OneMax* with problem sizes $N = 10, 20, 30, 50$, and 100. The results show that the control parameter, i.e., the mutation rate τ of the mutation rate σ, has a significant impact on the success of the SA-$(1 + 1)$-EA. Both other setting, i.e.,

[1] employing a logarithmic scale

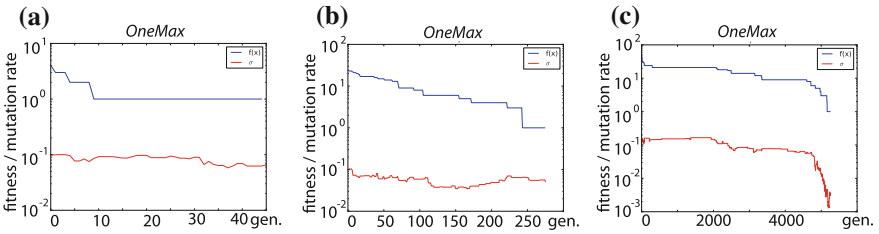

Fig. 3.4 SA-(1 + 1)-EA on *OneMax* with N = 10,50, and 100. **a** SA, N = 10, **b** SA, N = 50, **c** SA, N = 100

Table 3.2 Number of generations the SA-(1 + 1)-EA needs to reach the optimum

N	10	20	30	50	100
$\tau = 0.01$	48.3 ± 29.03	162.0 ± 83.1	359.0 ± 175.0	$2.4e3 \pm 552.8$	$> 10^5$
$\tau = 0.1$	46.1 ± 36.3	142.9 ± 47.1	274.0 ± 97.4	$1.0e3 \pm 770.7$	$3.6e3 \pm 3.3e3$
$\tau = 1.0$	$2.7e3 \pm 4.9e3$	$5.0e3 \pm 1.2e4$	$8.9e3 \pm 9.5e3$	$1.9e4 \pm 1.4e4$	$> 10^5$

$\tau = 0.01$ and $\tau = 1.0$ lead to worse results. In particular on the large problem instance with $N = 100$, both settings fail and lead to long optimization runs.

3.7 Conclusions

The success of evolutionary algorithms depends on the choice of appropriate parameter settings, in particular mutation rates. Although a lot of studies are known in literature, only few compare different parameter control techniques employing the same algorithmic settings on the same problems. But only such a comparison allows insights into the underlying mechanisms and common principles. The analysis has shown that optimally tuned mutation rates can automatically be found with meta-evolution. The effort spent into the search is comparatively high, but the final result is competitive or better than the control techniques. But more flexible and still powerful is the adaptive mutation rate control with Rechenberg's rule. Self-adaptation turns out to be the most flexible control technique with its automatic mutation rate control. Although self-adaptation depends on the control parameter τ, it is quite robust w.r.t. the problem size. It became famous in ES for continuous optimization and also has shown the best results in our parameter control study. As future work, we plan to extend our analysis to further EA variants, parameter control techniques, and problem types.

References

1. K.A.D. Jong, An analysis of the behavior of a class of genetic adaptive systems. Ph.D. thesis, University of Michigan, 1975
2. J.D. Schaffer, R. Caruana, L.J. Eshelman, R. Das, A study of control parameters affecting online performance of genetic algorithms for function optimization, in *Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA)*, pp. 51–60, 1989
3. J. Grefenstette, Optimization of control parameters for genetic algorithms. IEEE Trans. Syst. Man Cybern. **16**(1), 122–128 (1986)
4. H. Mühlenbein, How genetic algorithms really work: Mutation and hillclimbing, in *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature (PPSN)*, pp. 15–26, 1992
5. A.E. Eiben, R. Hinterding, Z. Michalewicz, Parameter control in evolutionary algorithms. IEEE Trans. Evol. Comput. **3**(2), 124–141 (1999)
6. O. Kramer, *Self-Adaptive Heuristics for Evolutionary Computation, Studies in Computational Intelligence* (Springer, Heidelberg, 2008)
7. S. Droste, T. Jansen, I. Wegener, On the analysis of the (1+1) evolutionary algorithm. Theoret. Comput. Sci. **276**(1–2), 51–81 (2002)
8. H.-G. Beyer, H.-P. Schwefel, Evolution strategies—A comprehensive introduction. Nat. Comput. **1**, 3–52 (2002)
9. I. Rechenberg, *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution* (Frommann-Holzboog, Stuttgart, 1973)
10. H.-P. Schwefel, *Adaptive Mechanismen in der biologischen Evolution und ihr Einfluss auf die Evolutionsgeschwindigkeit* (Interner Bericht der Arbeitsgruppe Bionik und Evolutionstechnik am Institut für Mess- und Regelungstechnik, TU Berlin, 1974)
11. D.B. Fogel, L.J. Fogel, J.W. Atma, Meta-evolutionary programming, in *Proceedings of 25th Asilomar Conference on Signals, Systems and Computers*, pp. 540–545, 1991