8

Parameter Control

The issue of setting the values of evolutionary algorithm parameters *before* running an EA was treated in the previous chapter. In this chapter we discuss how to do this *during* a run of an EA, in other words, we elaborate on controlling EA parameters on-the-fly. This has the potential of adjusting the algorithm to the problem while solving the problem. We provide a classification of different approaches based on a number of complementary features and present examples of control mechanisms for every major EA component. Thus we hope to both clarify the points we wish to raise and also to give the reader a feel for some of the many possibilities available for controlling different parameters.

8.1 Introduction

In the previous chapter we argued that parameter tuning can greatly increase the performance of EAs. However, the tuning approach has an inherent drawback: parameter values are specified before the run of the EA and these values remain fixed during the run. But a run of an EA is an intrinsically dynamic, adaptive process. The use of rigid parameters that do not change their values is thus in contrast to this spirit. Additionally, it is intuitively obvious, and has been empirically and theoretically demonstrated on many occasions, that different values of parameters might be optimal at different stages of the evolutionary process. For instance, large mutation steps can be good in the early generations, helping the exploration of the search space, and small mutation steps might be needed in the late generations for fine-tuning candidate solutions. This implies that the use of static parameters itself can lead to inferior algorithm performance.

A straightforward way to overcome the limitations of static parameters is by replacing a parameter p by a function $p(t)$, where t is the generation counter (or any other measure of elapsed time). However, as discussed in Chap. 7, the problem of finding optimal static parameters for a particular problem

A.E. Eiben, J.E. Smith, *Introduction to Evolutionary Computing*,
Natural Computing Series, DOI 10.1007/978-3-662-44874-8_8

is already hard. Designing optimal dynamic parameters (that is, functions for $p(t)$) may be even more difficult. Another drawback to this approach is that the parameter value $p(t)$ changes are caused by a 'blind' deterministic rule triggered by the progress of time t, unaware of the current state of the search. A well-known instance of this problem occurs in simulated annealing (Sect. 8.4.5) where a so-called cooling schedule has to be set before the execution of the algorithm.

Mechanisms for modifying parameters during a run in an 'informed' way were realised quite early in EC history. For instance, evolution strategies changed mutation parameters on-the-fly by Rechenberg's 1/5 success rule [352] using information about the ratio of successful mutations. Davis experimented with changing the crossover rate in GAs based on the progress realised by particular crossover operators [97]. The common feature of such methods is the presence of a human-designed feedback mechanism that utilises actual information about the search to determine new parameter values.

A third approach is based on the observation that finding good parameter values for an EA is a poorly structured, ill-defined, complex problem. This is exactly the kind of problem on which EAs are often considered to perform better than other methods. It is thus a natural idea to use an EA for tuning an EA to a particular problem. This could be done using a meta-EA or by using only one EA that tunes itself to a given problem *while* solving that problem. Self-adaptation, as introduced in evolution strategies for varying the mutation parameters, falls within this category. In the next section we discuss various options for changing parameters, illustrated by an example.

8.2 Examples of Changing Parameters

Consider a numerical optimisation problem of minimising

$$f(\overline{x}) = f(x_1, \ldots, x_n),$$

subject to some inequality and equality constraints

$$g_i(\overline{x}) \leq 0, i = 1, \ldots, q,$$

and

$$h_j(\overline{x}) = 0, j = q + 1, \ldots, m,$$

where the domains of the variables are given by lower and upper bounds $l_i \leq x_i \leq u_i$ for $1 \leq i \leq n$. For such a problem we might design an EA based on a floating-point representation, where each individual $\overline{x}$ in the population is represented as a vector of floating-point numbers $\overline{x} = \langle x_1, \ldots, x_n \rangle$.

8.2.1 Changing the Mutation Step Size

Let us assume that offspring are produced by arithmetic crossover followed by Gaussian mutation that replaces components of the vector $\overline{x}$ by

$$x_i' = x_i + N(0, \sigma).$$

To adjust σ over time we use a function $\sigma(t)$ defined by some heuristic rule and a given measure of time t. For example, the mutation step size may be defined by the current generation number t as:

$$\sigma(t) = 1 - 0.9 \cdot \frac{t}{T},$$

where t varies from 0 to T, the maximum generation number. Here, the mutation step size $\sigma(t)$, which is used for all for vectors in the population and for all variables of each vector, decreases slowly from 1 at the beginning of the run ($t = 0$) to 0.1 as the number of generations t approaches T. Such decreases may assist the fine-tuning capabilities of the algorithm. In this approach, the value of the given parameter changes according to a fully deterministic scheme. The user thus has full control of the parameter, and its value at a given time t is completely determined and predictable.

Second, it is possible to incorporate feedback from the search process, still using the same σ for all vectors in the population and for all variables of each vector. For example, Rechenberg's 1/5 success rule [352] states that the ratio of successful mutations to all mutations should be 1/5. Hence if the ratio is greater than 1/5 the step size should be increased, and if the ratio is less than 1/5 it should be decreased. The rule is executed at periodic intervals, for instance, after k iterations each σ is reset by

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

where p_s is the relative frequency of successful mutations, measured over a number of trials, and the parameter c should be $0.817 \leq c \leq 1$ [372]. Using this mechanism, changes in the parameter values are now based on feedback from the search. The influence of the user is much less direct here than in the deterministic scheme above. Of course, the mechanism that embodies the link between the search process and parameter values is still a heuristic rule indicating how the changes should be made, but the values of $\sigma(t)$ are not deterministic.

Third, we can assign an individual mutation step size to each solution and make these co-evolve with the values encoding the candidate solutions. To this end we extend the representation of individuals to length $n + 1$ as $\langle x_1, \ldots, x_n, \sigma \rangle$ and apply some variation operators (e.g., Gaussian mutation and arithmetic crossover) to the values of x_i as well as to the σ value of an

individual. In this way, not only the solution vector values (x_i) but also the mutation step size of an individual undergo evolution. A solution introduced in Sect. 4.4.2 is:

$$\sigma' = \sigma \cdot e^{\tau \cdot N(0,1)}, \tag{8.1}$$

$$x_i' = x_i + \sigma' \cdot N_i(0,1). \tag{8.2}$$

Observe that within this self-adaptive scheme the heuristic character of the mechanism resetting the parameter values is eliminated, and a certain value of σ acts on all values of a single individual.

Finally, we can use a separate σ_i for each x_i, extend the representation to

$$\langle x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n \rangle,$$

and use the mutation mechanism described in Eq. (4.4). The resulting system is the same as the previous one, except the granularity, here we are co-evolving n parameters of the EA instead of 1.

8.2.2 Changing the Penalty Coefficients

In this section we illustrate that the evaluation function (and consequently the fitness function) can also be parameterised and varied over time. While this is a less common option than tuning variation operators, it can provide a useful mechanism for increasing the performance of an EA.

When dealing with constrained optimisation problems, penalty functions are often used (see Chap. 13 for more details). A common technique is the method of static penalties [302], which requires penalty parameters within the evaluation function as follows:

$$eval(\overline{x}) = f(\overline{x}) + W \cdot penalty(\overline{x}),$$

where f is the objective function, $penalty(\overline{x})$ is zero if no violation occurs and is positive[1] otherwise, and W is a user-defined weight prescribing how severely constraint violations are weighted. For instance, a set of functions f_j $(1 \leq j \leq m)$ can be used to construct the penalty, where the function f_j measures the violation of the jth constraint:

$$f_j(\overline{x}) = \begin{cases} \max\{0, g_j(\overline{x})\} & \text{if} \quad 1 \leq j \leq q, \\ |h_j(\overline{x})| & \text{if} \quad q+1 \leq j \leq m. \end{cases} \tag{8.3}$$

To adjust the evaluation function over time, we can replace the static parameter W by a function $W(t)$. For example, the method in [237] uses

$$W(t) = (C \cdot t)^{\alpha},$$

[1] For minimisation problems.

where C and α are constants. Note that the penalty pressure grows with the evolution time provided $1 \leq C$ and $1 \leq \alpha$.

A second option is to utilise feedback from the search process. In one example, the method decreases the penalty component $W(t+1)$ for the generation $t+1$ if all best individuals in the last k generations were feasible, and increases penalties if all best individuals in the last k generations were infeasible. If there are some feasible and infeasible individuals as best individuals in the last k generations, $W(t+1)$ remains without change, cf. [45]. Technically, $W(t)$ is updated in every generation t in the following way:

$$W(t+1) = \begin{cases} (1/\beta_1) \cdot W(t) & \text{if } \overline{b}^i \in \mathcal{F} \quad \text{for all } t-k+1 \leq i \leq t, \\ \beta_2 \cdot W(t) & \text{if } \overline{b}^i \in \mathcal{S}-\mathcal{F} \text{ for all } t-k+1 \leq i \leq t, \\ W(t) & \text{otherwise.} \end{cases}$$

In this formula, $\mathcal{S}$ is the set of all search points (solutions), $\mathcal{F} \subseteq \mathcal{S}$ is a set of all *feasible* solutions, $\overline{b}^i$ denotes the best individual in terms of the function *eval* in generation i, $\beta_1, \beta_2 > 1$, and $\beta_1 \neq \beta_2$ (to avoid cycling).

Third, we could allow self-adaptation of the weight parameter, similarly to the mutation step sizes in the previous section. For example, it is possible to extend the representation of individuals into $\langle x_1, \ldots, x_n, W \rangle$, where W is the weight that undergoes the same mutation and recombination as any other variable x_i. Furthermore, we can introduce a separate penalty for each constraint as per Eq. (8.3). Hereby we obtain a vector of weights and can extend the representation to $\langle x_1, \ldots, x_n, w_1, \ldots, w_m \rangle$. Then define

$$eval(\overline{x}) = f(\overline{x}) + \sum_{j=1}^{m} w_j f_j(\overline{x}),$$

as the function to be minimised. Variation operators can then be applied to both the $\overline{x}$ and the $\overline{w}$ part of these chromosomes, realising a self-adaptation of the penalties, and thereby the fitness function.

It is important to note the crucial difference between self-adapting mutation step sizes and constraint weights. Even if the mutation step sizes are encoded in the chromosomes, the evaluation of a chromosome is *independent* from the actual σ values. That is,

$$eval(\langle \overline{x}, \overline{\sigma} \rangle) = f(\overline{x}),$$

for any chromosome $\langle \overline{x}, \overline{\sigma} \rangle$. In contrast, if constraint weights are encoded in the chromosomes, then we have

$$eval(\langle \overline{x}, \overline{w} \rangle) = f_{\overline{w}}(\overline{x}),$$

for any chromosome $\langle \overline{x}, \overline{w} \rangle$. This could enable the evolution to 'cheat' in the sense of making improvements by minimising the weights instead of optimising f and satisfying the constraints. Eiben et al. investigated this issue in [134] and found that using a specific tournament selection mechanism neatly solves this problem and enables the EA to solve constraints.

8.3 Classification of Control Techniques

In classifying parameter control techniques of an evolutionary algorithm, many aspects can be taken into account. For example:

1. *what* is changed (e.g., representation, evaluation function, operators, selection process, mutation rate, population size, and so on)
2. *how* the change is made (i.e., deterministic heuristic, feedback-based heuristic, or self-adaptive)
3. *the evidence* upon which the change is carried out (e.g., monitoring performance of operators, diversity of the population, and so on)
4. *the scope/level* of change (e.g., population-level, individual-level, and so forth)

In the following we discuss these items in more detail.

8.3.1 *What* Is Changed?

To classify parameter control techniques from the perspective of what component or parameter is changed, it is necessary to agree on a list of all major components of an evolutionary algorithm, which is a difficult task in itself. For that purpose, let us assume the following components of an EA:

- representation of individuals
- evaluation function
- variation operators and their probabilities
- selection operator (parent selection or mating selection)
- replacement operator (survival selection or environmental selection)
- population (size, topology, etc.)

Note that each component can be parameterised, and that the number of parameters is not clearly defined. Despite the somewhat arbitrary character of this list of components and of the list of parameters of each component, we will maintain the 'what-aspect' as one of the main classification features, since this allows us to locate where a specific mechanism has its effect.

8.3.2 *How* Are Changes Made?

As illustrated in Sect. 8.2, methods for changing the value of a parameter (i.e., the 'how-aspect') can be classified into one of three categories.

- **Deterministic parameter control**
  Here the value of a parameter is altered by some deterministic in rule predetermined (i.e., user-specified) manner without using any feedback from the search. Usually, a time-varying schedule is used, i.e., the rule is activated at specified intervals.

- **Adaptive parameter control**
  Here some form of feedback from the search serves as input to a mechanism that determines the change. Updating the parameter values may involve credit assignment, based on the quality of solutions discovered by different operators/parameters, so that the updating mechanism can distinguish between the merits of competing strategies. The important point to note here is that the updating mechanism used to control parameter values is externally supplied, rather than being part of the usual evolutionary cycle.
- **Self-adaptive parameter control**
  Here the evolution of evolution is used to implement the self-adaptation of parameters [257]. The parameters to be adapted are encoded into the chromosomes and undergo mutation and recombination. The better values of these lead to better individuals, which in turn are more likely to survive, produce offspring and hence propagate these better parameter values. This is an important distinction between adaptive and self-adaptive schemes: in the latter the mechanisms for the credit assignment and updating of different strategy parameters are entirely implicit, i.e., they are the selection and variation operators of the evolutionary cycle itself.

This terminology leads to the taxonomy illustrated in Fig. 8.1.

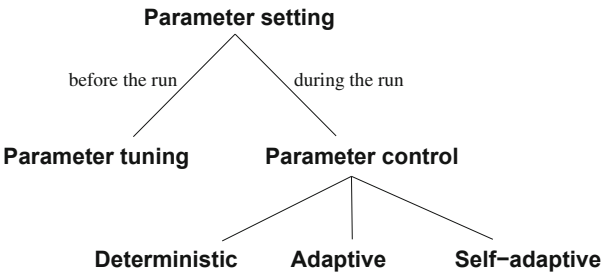

Fig. 8.1. Global taxonomy of parameter setting in EAs

Some authors have introduced a different terminology, cf. [8] or [410], but after the publication of [133] the one in Fig. 8.1 became the most widely accepted one. However, we acknowledge that the terminology proposed here is not perfect. For instance, the term "deterministic" control might not be the most appropriate, as it is not determinism that matters, but the fact that the parameter-altering mechanism is "uninformed", i.e., takes no input related to the progress of the search process. For example, one might *randomly* change the mutation probability after every 100 generations, which is not a deterministic process. Also, the terms "adaptive" and "self-adaptive" could be replaced by the equally meaningful "explicitly adaptive" and "implicitly adaptive", respectively. We have chosen to use "adaptive" and "self-adaptive" because of the widely accepted usage of the latter term.

8.3.3 What *Evidence* Informs the Change?

The third criterion for classification concerns the evidence used for determining the change of parameter value [382, 397]. Most commonly, the progress of the search is monitored by looking at the performance of operators, the diversity of the population, and so on, and the information gathered is used as feedback for adjusting the parameters. From this perspective, we can further distinguish between the following two cases:

- **Absolute evidence**
  We speak of absolute evidence when the rule to change the value of a parameter is applied when a predefined event occurs. For instance, one could increase the mutation rate when the population diversity drops under a given value, or resize the population based on estimates of schemata fitness and variance. As opposed to deterministic parameter control, where a rule fires by a deterministic trigger (e.g., time elapsed), here feedback from the search is used. Such mechanisms require that the user has a clear intuition about how to steer the given parameter into a certain direction in cases that can be specified in advance (e.g., they determine the threshold values for triggering rule activation). This intuition relies on the implicit assumption that changes that were appropriate in *another* run on *another* problem are applicable to *this* run on *this* problem.
- **Relative evidence**
  In the case of using relative evidence, parameter values within the *same* run are compared according to the positive/negative effects they produce, and the better values get rewarded. The direction and/or magnitude of the change of the parameter is not specified deterministically, but relative to the performance of other values, i.e., it is necessary to have more than one value present at any given time. As an example, consider an EA using several crossovers with crossover rates adding up to 1.0 and being reset based on the crossover's performance measured by the quality of offspring they create.

8.3.4 What Is the *Scope* of the Change?

As discussed earlier, any change within any component of an EA may affect a gene (parameter), whole chromosomes (individuals), the entire population, another component (e.g., selection), or even the evaluation function. This is the aspect of the scope or level of adaptation [8, 214, 397]. Note, however, that the scope or level is not an independent dimension, as it usually depends on the component of the EA where the change takes place. For example, a change of the mutation step size may affect a gene, a chromosome, or the entire population, depending on the particular implementation (i.e., scheme used), but a change in the penalty coefficients typically affects the whole population. In this respect the scope feature is a secondary one, usually depending on the given component and its actual implementation.

8.3.5 Summary

In conclusion, the main criteria for classifying methods that change the values of the strategy parameters of an algorithm during its execution are:

1. What component/parameter is changed?
2. How is the change made?
3. What evidence is used to make the change?

Our classification is thus three-dimensional. The *component* dimension consists of six categories: representation, evaluation function, variation operators (mutation and recombination), selection, replacement, and population. The other dimensions have respectively three (deterministic, adaptive, self-adaptive) and two categories (absolute, relative). Their possible combinations are given in Table 8.1. As the table indicates, deterministic parameter control with relative evidence is impossible by definition, and so is self-adaptive parameter control with absolute evidence. Within the adaptive scheme both options are possible and are indeed used in practice.

	Deterministic	Adaptive	Self-adaptive
Absolute	+	+	–
Relative	–	+	+

Table 8.1. Refined taxonomy of parameter setting in EAs: types of parameter control along the type and evidence dimensions. The '–' entries represent meaningless (nonexistent) combinations

8.4 Examples of Varying EA Parameters

Here we discuss some illustrative examples concerning all major EA components. For a more comprehensive overview the reader is referred to the classic survey from 1999 [133] and its recent successor [241].

8.4.1 Representation

We illustrate variable representations with the delta coding algorithm of Mathias and Whitley [461], which effectively modifies the encoding of the function parameters. The motivation behind this algorithm is to maintain a good balance between fast search and sustaining diversity. In our taxonomy it can be categorised as an adaptive adjustment of the representation based on absolute evidence. The GA is used with multiple restarts; the first run is used to find an *interim solution*, and subsequent runs decode the genes as distances (*delta values*) from the last interim solution. This way each restart

forms a new hypercube with the interim solution at its origin. The resolution of the delta values can also be altered at the restarts to expand or contract the search space. The restarts are triggered when population diversity (measured by the Hamming distance between the best and worst strings of the current population) is not greater than 1. The sketch of the algorithm showing the main idea is given in Fig. 8.2. Note that the number of bits for δ can be increased if the same solution `INTERIM` is found.

```
BEGIN
  /* given a starting population and genotype-phenotype encoding */
  WHILE (  HD > 1 ) DO
    RUN_GA with k bits per object variable;
  OD
  REPEAT UNTIL (  global termination is satisfied ) DO
    save best solution as INTERIM;
    reinitialise population with new coding;
    /*  k-1 bits as the distance δ to the object value in  */
    /*  INTERIM and one sign bit */
    WHILE (  HD > 1 ) DO
      RUN_GA with this encoding;
    OD
  OD
END
```

Fig. 8.2. Outline of the delta coding algorithm

8.4.2 Evaluation Function

Evaluation functions are typically not varied in an EA because they are often considered as part of the problem to be solved and not as part of the problem-solving algorithm. In fact, an evaluation function forms the bridge between the two, so both views are at least partially true. In many EAs the evaluation function is derived from the (optimisation) problem at hand with a simple transformation of the objective function. In the class of constraint satisfaction problems, however, there is no objective function in the problem definition, cf. Chap. 13. One possible approach here is based on penalties. Let us assume that we have m constraints c_i ($i \in \{1, \ldots, m\}$) and n variables v_j ($j \in \{1, \ldots, n\}$) with the same domain S. Then the penalties can be defined as follows:

$$f(\bar{s}) = \sum_{i=1}^{m} w_i \times \chi(\bar{s}, c_i),$$

where w_i is the weight associated with violating c_i, and

$$\chi(\bar{s}, c_i) = \begin{cases} 1 \text{ if } \bar{s} \text{ violates } c_i, \\ 0 \text{ otherwise.} \end{cases}$$

Obviously, the setting of these weights has a large impact on the EA performance, and ideally w_i should reflect how hard c_i is to satisfy. The problem is that finding the appropriate weights requires much insight into the given problem instance, and therefore it might not be practicable.

The stepwise adaptation of weights (SAW) mechanism, introduced by Eiben and van der Hauw [149], provides a simple and effective way to set these weights. The basic idea behind the SAW mechanism is that constraints that are not satisfied after a certain number of steps (fitness evaluations) must be difficult, and thus must be given a high weight (penalty). SAW-ing changes the evaluation function adaptively in an EA by periodically checking the best individual in the population and raising the weights of those constraints this individual violates. Then the run continues with the new evaluation function. A nice feature of SAW-ing is that it liberates the user from seeking good weight settings, thereby eliminating a possible source of error. Furthermore, the used weights reflect the difficulty of constraints for *the given algorithm* on the *given problem instance* in *the given stage of the search* [151]. This property is also valuable since, in principle, different weights could be appropriate for different algorithms.

8.4.3 Mutation

A large majority of work on adapting or self-adapting EA parameters concerns variation operators: mutation and recombination (crossover). The 1/5 rule of Rechenberg we discussed earlier constitutes a classic example for adaptive mutation step size control in ES. Furthermore, self-adaptive control of mutation step sizes is traditional in ES [257].

8.4.4 Crossover

The classic example for adapting crossover rates in GAs is Davis's adaptive operator fitness. The method adapts the rates of crossover operators by rewarding those that are successful in creating better offspring. This reward is diminishingly propagated back to operators of a few generations back, who helped set it all up; the reward is an increase in probability at the cost of other operators [98]. The GA using this method applies several crossover operators simultaneously within the same generation, each having its own crossover rate $p_c(op_i)$. Additionally, each operator has its local delta value d_i that represents the strength of the operator measured by the advantage of a child created by using that operator with respect to the best individual in the population. The

local deltas are updated after every use of operator i. The adaptation mechanism recalculates the crossover rates periodically redistributing 15% of the probabilities biased by the accumulated operator strengths, that is, the local deltas. To this end, these d_i values are normalised so that their sum equals 15, yielding d_i^{norm} for each i. Then the new value for each $p_c(op_i)$ is 85% of its old value and its normalised strength:

$$p_c(op_i) = 0.85 \cdot p_c(op_i) + d_i^{norm}.$$

Clearly, this method is adaptive based on relative evidence.

8.4.5 Selection

Most existing mechanisms for varying the selection pressure are based on the so-called **Boltzmann** selection mechanism, which changes the selection pressure during evolution according to a predefined cooling schedule [279]. The name originates from the Boltzmann trial from condensed matter physics, where a minimal energy level is sought by state transitions. Being in a state i the chance of accepting state j is

$$P[\text{accept } j] = \begin{cases} 1 & \text{if} \quad E_i \geq E_j, \\ \exp\left(\frac{E_i - E_j}{K_b \cdot T}\right) & \text{if} \quad E_i < E_j, \end{cases}$$

where E_i, E_j are the energy levels, K_b is a parameter called the Boltzmann constant, and T is the temperature. This acceptance rule is called the Metropolis criterion.

We illustrate variable selection pressure in the survivor selection (replacement) step by **simulated annealing** (SA). SA is a generate-and-test search technique based on a physical, rather than a biological, analogy [2, 250]. Formally, however, SA can be envisioned as an evolutionary process with population size of 1, undefined (problem-dependent) representation and mutation, and a specific survivor selection mechanism. The selective pressure changes during the course of the algorithm in the Boltzmann style. The main cycle in SA is given in Fig. 8.3.

In this mechanism the parameter c_k, the temperature, decreases according to a predefined scheme as a function of time, making the probability of accepting inferior solutions smaller and smaller (for minimisation problems). From an evolutionary point of view, we have here a (1+1) EA with increasing selection pressure.

8.4.6 Population

An innovative way to control the population size is offered by Arabas et al. [11, 295] in their GA with variable population size (GAVaPS). In fact, the population size parameter is removed entirely from GAVaPS, rather than

```
BEGIN
  /* given a current solution i ∈ S */
  /* given a function to generate the set of neighbours N_i of i */
  generate j ∈ N_i;
  IF (f(i) < f(j)) THEN
    set i = j;
    ELSE
      IF ( exp((f(i)−f(j))/c_k) > random[0,1)) THEN
        set i = j;
      FI
    ESLE
  FI
END
```

Fig. 8.3. Outline of the simulated annealing algorithm

adjusted on-the-fly. Certainly, in an evolutionary algorithm the population always has a size, but in GAVaPS this size is a derived measure, not a controllable parameter. The main idea is to assign a lifetime to each individual when it is created, and then to reduce its remaining lifetime by one in each consecutive generation. When the remaining lifetime becomes zero, the individual is removed from the population. Two things must be noted here. First, the lifetime allocated to a newborn individual is biased by its fitness: fitter individuals are allowed to live longer. Second, the expected number of offspring of an individual is proportional to the number of generations it survives. Consequently, the resulting system favours the propagation of good genes.

Fitting this algorithm into our general classification scheme is not straightforward because it has no explicit mechanism that sets the value of the population size parameter. However, the procedure that implicitly determines how many individuals are alive works in an adaptive fashion using information about the status of the search. In particular, the fitness of a newborn individual is related to the fitness of the present generation, and its lifetime is allocated accordingly. This amounts to using relative evidence.

8.4.7 Varying Several Parameters Simultaneously

Mutation, crossover, and population size are all controlled on-the-fly in the GA "without parameters" of Bäck et al. in [25]. Here, the self-adaptive mutation from [17] (Sect. 8.4.3) is adopted without changes, a new self-adaptive technique is invented for regulating the crossover rates of the individuals, and the GAVaPS lifetime idea (Sect. 8.4.6) is adjusted for a steady-state GA model. The crossover rates are included in the chromosomes, much like the mutation rates. If a pair of individuals is selected for reproduction, then their

individual crossover rates are compared with a random number $r \in [0, 1]$, and an individual is seen as ready to mate if its $p_c > r$. Then there are three possibilities:

1. If both individuals are ready to mate then uniform crossover is applied, and the resulting offspring is mutated.
2. If neither is ready to mate then both create a child by mutation only.
3. If exactly one of them is ready to mate, then the one not ready creates a child by mutation only (which is inserted into the population immediately through the steady-state replacement), the other is put on hold, and the next parent selection round picks only one other parent.

This study differs from those discussed before in that it explicitly compares GA variants using only one of the (self-)adaptive mechanisms and the GA applying them all. The experiments show remarkable outcomes: the completely (self-)adaptive GA wins, closely followed by the one using only the adaptive population size control, and the GAs with self-adaptive mutation and crossover are significantly worse.

8.5 Discussion

Summarising this chapter, a number of things can be noted. First, parameter control in an EA can have two purposes. It can be done to find good parameter values for the EA at hand. Thus, it offers the same benefits as parameter tuning, but in an on-line fashion. From this perspective tuning and control are two different approaches to solving the same problem. Whether or not one is preferable over the other is an open question with very little empirical evidence to support an answer. Systematic investigations are particularly difficult here because of methodological problems. The essence of these problems is that a fair comparison of the extra computational costs (learning overhead) and the performance gains is hard to define in general.

The other motivation for controlling parameters on-the-fly is the assumption that the given parameter can have a different 'optimal' value in different stages of the search. If this holds, then there is simply no optimal static parameter value; for good EA performance one must vary this parameter. From this perspective tuning and control are not the same, control offers a benefit that tuning cannot.

The second thing to remark is that making a parameter (self-)adaptive does not necessarily mean that we obtain an EA with fewer parameters. For instance, in GAVaPS the population size parameter is eliminated at the cost of introducing two new ones: the minimum and maximum lifetime of newborn individuals. If the EA performance is sensitive to these new parameters then such a parameter replacement can make things worse. This problem also occurs on another level. One could say that the procedure that allocates lifetimes in GAVaPS, the probability redistribution mechanism for adaptive

crossover rates (Sect. 8.4.4), or the function specifying how the σ values are mutated in ES (Eq. (4.4)) are also (meta)parameters. It is in fact an assumption that these are intelligently designed and their effect is positive. In many cases there are more possibilities, that is, possibly well-working procedures one can design. Comparing these possibilities implies experimental (or theoretical) studies very much like comparing different parameter values in a classical setting. Here again, it can be the case that algorithm performance is not so sensitive to details of this (meta)parameter, which fully justifies this approach.

Third, it is important to note that the examples in the foregoing sections, while serving as good illustrations of various aspects of parameter control, do not represent the state of the art in 2014. There has been much research into parameter control during the last decade. It has been successfully applied in various domains of metaheuristics, including Evolution Strategies [257], Genetic Algorithms [162, 291], Differential Evolution [349, 280], and Particle Swarm Optimization [473]. Furthermore, there are several noteworthy contributions to the techniques behind parameter control. These range from inventive ideas that need further elaboration, like applying self-organised criticality [266], self-adaptation of population level parameters (e.g., population size) [144] or tuning the controllers to a problem instance [242], to generally applicable mechanisms including adaptive pursuit strategies for operator allocation [429], the Compass method [286] or ACROMUSE [291].

These and other valuable contributions to the field provide more and more evidence about the possible benefits and accumulate the knowhow of successful parameter control. Although the field is still in development, we can identify some trends and challenges. The research community seems to converge on the idea that successful parameter control must take into account two types of information regarding the evolutionary search: data about fitness and population diversity. However, there is a wide variety of approaches to how exactly we should define these types of information; for instance, there are many different ways to define diversity. A very promising approach was put forward recently by McGinley et al. based on the idea of considering the diversity of the fittest part of the population (the 'healthy' individuals) instead of the whole population's diversity [291]. Another agreement on a conceptual level is that a control mechanism is only successful if it appropriately balances exploration and exploitation. But here again, there is no generally accepted definition of these notions, indicating the need for more research [142, 91]. Perhaps one of the biggest obstacles that hinders widespread adoption of existing parameter control techniques is the 'patchwork problem'. The problem here is the lack of generally applicable methods for controlling EA parameters. There are numerous techniques to control mutation, quite a lot for controlling recombination, several ways to adjust population size and a handful for changing selection pressure on-the fly. To build an EA with all parameters controlled, one needs to pick some method for each parameter thus creating a

potpourri or patchwork with no solid evidence indicating how it all will work together.

Finally, let us place the issue in a larger perspective of parameter setting in EAs [273]. Over recent decades the EC community shifted from believing that EA performance is to a large extent independent from the given problem instance to realising that it is. In other words, it is now acknowledged that EAs need more or less fine-tuning to specific problems and problem instances. Ideally, this should be automated and advanced (search) algorithms should determine the best EA setting, instead of conventions and the users' intuitions. For the case of doing this in advance, before the EA run there are several powerful algorithms developed over the last ten years, see Section 7.6 and [145]. To put it optimistically, the tuning problem is now solved, and the community of EA researchers and practitioners can adopt tuning as part of the regular workflow. However, the picture is completely different for parameter control. As outlined above, this field is still in its infancy, requiring fundamental research into the most essential concepts (diversity, exploration, etc.) as well as algorithmic development towards good control strategies and some unification (solution of the patchwork problem). To this end, we can recommend the recent overview of Karafotias et al. [241] that identifies current research trends and provides suggestions for important research directions.

For exercises and recommended reading for this chapter, please visit www.evolutionarycomputation.org.