Parameter Setting in EAs: a 30 Year Perspective

Kenneth De Jong

Department of Computer Science
George Mason University
4400 University Drive, MSN 4A5
Fairfax, VA 22030, USA
kdejong@gmu.edu

Summary. Parameterized evolutionary algorithms (EAs) have been a standard part of the Evolutionary Computation community from its inception. The widespread use and applicability of EAs is due in part to the ability to adapt an EA to a particular problem-solving context by tuning its parameters. However, tuning EA parameters can itself be a challenging task since EA parameters interact in highly non-linear ways. In this chapter we provide a historical overview of this issue, discussing both manual (human-in-the-loop) and automated approaches, and suggesting when a particular strategy might be appropriate.

1 Introduction

More than 30 years have passed since I completed my thesis [18], and more than 40 years since the pioneering work of Rechenberg [34], Schwefel [36], Foget et al. [17] and Holland [24]. Since then the field of Evolutionary Computation (EC) has grown dramatically and matured into a well-established discipline.

One of the characteristics of the field from the very beginning was the notion of a parameterized evolutionary algorithm (EA), the behavior of which could be modified by changing the value of one or more of its parameters. This is not too surprising since EAs consist of populations of individuals that produce offspring using reproductive mechanisms that introduce variation into the population. It was clear from the beginning that changing the population size, the type and amount of reproductive variation, etc. could significantly change the behavior of an EA on a particular fitness landscape and, conversely, appropriate parameter settings for one fitness landscape might be inappropriate for others.

It is not surprising, then, that from the beginning EA practitioners have wanted to know the answers to questions like:

K. De Jong: *Parameter Setting in EAs: a 30 Year Perspective*, Studies in Computational Intelligence (SCI) **54**, 1–18 (2007)
www.springerlink.com

- Are there optimal settings for the parameters of an EA in general?
- Are there optimal settings for the parameters of an EA for a particular class of fitness landscapes?
- Are there robust settings for the parameters of and EA than produce good performance over a broad range of fitness landscapes?
- Is it desirable to dynamically change parameter values during an EA run?
- How do changes in a parameter affect the performance of an EA?
- How do landscape properties affect parameter value choices?

A look at the table of contents of this book will convince you that there are no simple answers to these questions. In this chapter I will provide my perspective on these issues and describe a general framework that I use to help understand the answers to such questions.

2 No Free Lunch Theorems

It has been shown in a variety of contexts including the areas of search, optimization, and machine learning that, unless adequate restrictions are placed on the the class of problems one is attempting to solve, there is no single algorithm that will outperform all other algorithms (see, for example, [83]). While very general and abstract, No Free Lunch (NFL) theorems provide a starting point for answering some of the questions listed in the previous section. In particular, these theorems serve as both a cautionary note to EA practitioners to be careful about generalizing their results, and as a source of encouragement to use parameter tuning as a mechanism for adapting algorithms to particular classes of problems.

It is important at this point to clarify several frequently encountered misconceptions about NFL results. The first misconception is that NFL results are often interpreted as applying only to algorithms that do not self-adapt during a problem solving run. So, for example, I might imagine a two-stage algorithm that begins by first figuring out what kind of problem it is being asked to solve and then choosing the appropriate algorithm (or parameter settings) from its repertoire, thus avoiding NFL constraints. While these approaches can extend the range of problems for which an algorithm is effective, it is clear that the ability of the first stage to identify problem types and select appropriate algorithms is itself an algorithm that is subject to NFL results.

A second misconception is the interpretation of NFL results as leaving us in a hopeless state unable to say anything general about the algorithms we develop. A more constructive interpretation of NFL results is that they present us with a challenge to define more carefully the classes of problems we are trying to solve and the algorithms we are developing to solve them. In particular, NFL results provide the context for a meaningful discussion parameter setting in EAs, including:

- What EA parameters are useful for improving performance?

- How does one choose appropriate parameter values?
- Should parameter values be fixed during a run or be modified dynamically?

These issues are explored in some detail in the following sections.

3 Parameter Setting in Simple EAs

Before we tackle more complicated EAs, it is helpful to ascertain what we know about parameter setting for simple EAs. I use the term "simple EA" to refer to algorithms in which:

- A population of individuals is evolved over time.
- The current population is used as a source of parents to produce some offspring.
- A subset of the parents and offspring are selected to "survive" into the next generation.

This is the form that the early EAs took (Evolution Strategies (ESs), Evolutionary Programming (EP), and Genetic Algorithms (GAs)), and it represents the basic formulation of many EAs used today. With any algorithm, the easiest things to consider parameterizing are the numerical elements. In the case of the early EAs, the obvious candidate was the population size. For ESs, the parent population size μ was specified independently of the offspring population size λ, while early EP and GA versions defined both populations to be of the same size controlled by a single parameter m. The values of these parameters were specified at the beginning of a run and remained unchanged throughout a run.

Somewhat more subtle were the parameters defined to control reproductive variation (the degree to which offspring resemble their parents). Early ES and EP algorithms produced offspring via *asexual* reproduction, i.e., by cloning and mutation. Hence, the amount of reproductive variation is controlled by varying the parameters of the mutation operator, one of which specifies the (expected) number of "genes" undergoing mutation. A second parameter controls how different (on average) a mutated gene is from its original form.

Since many ES and EP algorithms were evolving populations of individuals whose genes were real-valued parameters of an objective fitness function, a Gaussian mutation operator was used with a mean of zero and a variance of σ^2. It became clear early on that leaving σ fixed for the duration of an entire evolutionary run was suboptimal, and mechanisms were developed for dynamically adapting its value were developed. The earliest of these mechanisms was the "1/5th rule"' developed by Rechenberg and Schwefel [36] in which the ratio of "successful" mutations (improvements in fitness) to unsuccessful ones was monitored during the course of a run. If the ratio fell below 1/5, σ was decreased. If the ratio exceeded 1/5, σ was increased. The amount

that σ was increased or decreased was a parameter as well, but set to a fixed value at the beginning of a run.

By contrast, early GA versions used an internal binary string representation, and produced reproductive variation via two-parent (sexual) recombination and a "bit-flipping" mutation operator. In this case the degree of reproductive variation was controlled by the amount of recombination and the number of bits flipped. The amount of recombination was controlled in two ways: by specifying the percentage of offspring to be produced using recombination, and the number of crossover!points to be used when performing recombination (see, for example, [18]). Interestingly, there was less concern about dynamically adapting these GA parameters, in part because the variation due to recombination decreases automatically as the population becomes more homogeneous.

So, already with the early canonical EAs we see examples of parameters whose values remained fixed through out an EA run and parameters whose values are dynamically modified during a run. Both of these approaches are examined in more detail in the following sections.

4 Static Parameter Setting Strategies

The No Free Lunch results place an obligation on the EA practitioner to understand something about the particular properties of the problems that (s)he is trying to solve that relate to particular choices of EA parameter settings. This poses a bit of a dilemma in that often an EA is being used precisely because of a lack of knowledge about the fitness landscapes being explored. As a consequence, the parameter setting strategy frequently adopted is a multiple run, human-in-the-loop approach in which various parameter settings are tried in an attempt to fine-tune an EA to a particular problem.

Since this can be a somewhat tedious process, it is not uncommon to automate this process with a simple, top-level "parameter sweep" procedure that systematically adjusts selected parameter values. However, unless care is taken, this can lead to a combinatorial explosion of parameter value combinations and require large amounts of computation time. One possible escape from this is to replace the parameter sweep procedure with a parameter optimization procedure. Which optimization procedure to use is an interesting question. The fitness landscapes defined by EA parameter tuning generally have the same unknown properties as the underlying problems that suggested the use of EAs in the first place! So, a natural consequence is the notion of a two-level EA, the top level of which is evolving the parameters of the lower level EA. The best example of this approach is the "nested" Evolution Strategy in which a top-level ES evolves the parameters for a second-level ES [35]. An interesting (and open question) is how to choose the parameters for top-level EAs!

Since exploring EA parameter space is generally very time consuming and computationally expensive, it is done in a more general "offline" setting. The idea here is to find EA parameter settings that are optimal for a class of problems using a sweep or optimization procedure on a selected "test suite" of sample problems, and then use the resulting parameter settings for all of the problems of that class when encountered in practice (see, for example, [18] or [34]). The key insight from such studies is the robustness of EAs with respect to their parameter settings. Getting "in the ball park" is generally sufficient for good EA performance. Stated another way, the EA parameter "sweet spot" is reasonably large and easy to find [18]. As a consequence most EAs today come with a default set of static parameter values that have been found to be quite robust in practice.

5 Dynamic Parameter Setting Strategies

A more difficult thing to ascertain is whether there is any advantage to be gained by dynamically changing the value of a parameter during an EA run and, if so, how to change it. The intuitive motivation is that, as evolution proceeds, it should be possible to use the accumulating information about the fitness landscape to improve future performance. The accumulating information may relate to global properties of the fitness landscape such as noise or ruggedness, or it may relate to the local properties of a particular region of the landscape. A second intuition is a sense of the need to change EA parameters as it evolves from a more diffuse global search process to a more focused converging local search process.

One way to accomplish this is to set up an *a priori* schedule of parameter changes in a fashion similar to those used in simulating annealing [16]. This is difficult to accomplish in general since it is not obvious how to predict the number of generations an EA will take to converge, and set a parameter adjustment schedule appropriately. A more successful approach is to monitor particular properties of an evolutionary run, and use changes in these properties as a signal to change parameter values. A good example of this approach is the 1/5 rule described in the previous section. More recently, the use of a covariance matrix for mutation step size adaptation has been proposed [23]. Another example is the adaptive reproductive operator selection procedure developed by Davis [17] in which reproductive operators are dynamically chosen based on their recent performance.

The purist might argue that inventing feedback control procedures for EAs is a good example of over-engineering an already sophisticated adaptive system. A better strategy is to take our inspiration from nature and design our EAs to be self-regulating. For example, individuals in the population might contain "regulatory genes" that control mutation and recombination mechanisms, and these regulatory genes would be subject to the same evolutionary processes as the rest of the genome. Historically, the ES community has used

this approach as a way of independently controlling the mutation step size σ_i of each objective parameter value x_i [3]. Similar ideas have been used to control the probability of choosing a particular mutation operator for finite state machines [16]. Within the GA community Spears [76] has used a binary control gene to determine which of two crossover operators to use.

These two approaches to setting parameters dynamically, using either an *ad hoc* or a self-regulating control mechanism, have been dubbed "adaptive" and "self-adaptive" in the literature [11].

Perhaps the most interesting thing to note today, after more than 30 years of experimenting with dynamic parameter setting strategies, is that, with one exception, none of them are used routinely in every day practice. The one exception are the strategies used by the ES community for mutation step size adaptation. In my opinion this is due is the fact that it is difficult to say anything definitive and general about the performance improvements obtained through dynamic parameter setting strategies. There are two reasons for this difficulty. First, our EAs are stochastic, non-linear algorithms. This means that formal proofs are extremely difficult to obtain, and experimental studies must be carefully designed to provide statistically significant results. The second reason is that a legitimate argument can be made that comparing the performance of an EA with static parameter settings to one with dynamic settings is unfair since it is likely that the static settings were established via some preliminary parameter tuning runs that have not been included in the comparison.

My own view is that there is not much to be gained in dynamically adapting EA parameter settings when solving static optimization problems. The real payoff for dynamic parameter setting strategies is when the fitness landscapes are themselves dynamic (see, for example, [4], [5], or [32]).

6 Choosing the EA Parameters to Set

An important aspect to EA parameter setting is a deeper understanding of how changes in EA parameters affect EA performance. Without that understanding it is difficult to ascertain which parameters, if properly set, would improve performance on a particular problem. This deeper understanding can be obtained in several ways. The simplest approach is to study parameter setting in the context of a particular family of EAs (e.g., ESs, GAs, etc.). The advantage here is that they are well-studied and well-understood. For example, there is a large body of literature in the ES community regarding the role and the effects of the parent population size μ, the offspring population size λ, and mutation step size σ. Similarly, the GA community has studied population size m, various crossover operators, and mutation rates.

The disadvantage to this approach is that the EAs in use today seldom fit precisely into one of these canonical forms. Rather, they have evolved via recombinations and mutations of these original ideas. It would be helpful, then,

to understand the role of various parameters at a higher level of abstraction. This is something that I have been interested in for several years. The basis for this is a unified view of simple EAs [9]. From this perspective an EA practitioner makes a variety of design choices, perhaps the most important of which is representation. Having made that choice, there are still a number of additional decisions to be made that affect EA performance, the most important of which are:

- The size of the parent population m.
- The size of the offspring population n.
- The procedure for selecting parents *p_select*.
- The procedure for producing offspring *reproduction*.
- The procedure for selecting survivors *s_select*.

Most modern EA toolkits parameterize these decisions allowing one to choose traditional parameter settings or create new and novel variations. So, for example, a canonical $(\mu + \lambda)$-ES would be specified as:

- $m = \mu$
- $n = \lambda$
- *p_select* = deterministic and uniform
- *reproduction* = clone and mutate
- *s_select* = deterministic truncation

and a canonical GA would be specified as:

- $m = n$
- *p_select* = probabilistic and fitness-proportional
- *reproduction* = clone, recombine, and mutate
- *s_select* = offspring only

It is at this level of generality that we would like to understand the effects of parameter settings. We explore this possibility in the following subsections.

6.1 Parent Population Size *m*

From my perspective the parent population size m can be viewed as a measure of the degree of parallel search that an EA supports, since the parent population is the basis for generating new search points. For simple landscapes like the 2-dimensional (inverted) parabola illustrated in Figure 1, little, if any, parallelism is required since any sort of simple hill climbing technique will provide reasonable performance.

By contrast, more complex, multi-peaked landscapes may require populations of 100s or even 1000s of parents in order to have some reasonable chance of finding globally optimal solutions. As a simple illustration of this, consider the 2-dimensional landscape defined by the fitness function $f(x_1, x_2) = x_1^2 + x_2^2$ in which the variables x_1 and x_2 are constrained to the interval $[-10, 5]$.

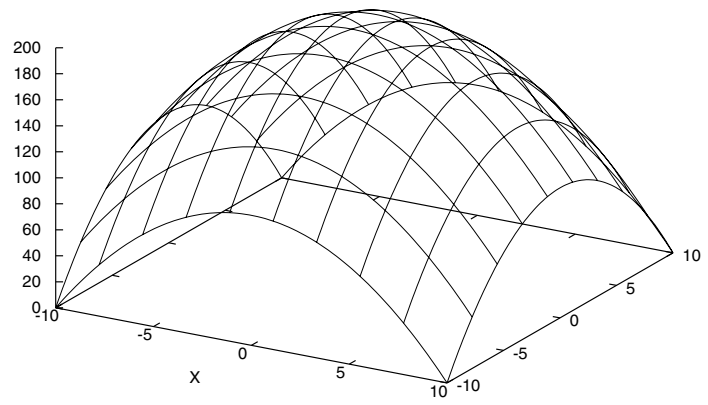

Fig. 1. A simple 2-dimensional (inverted) parabolic fitness landscape.

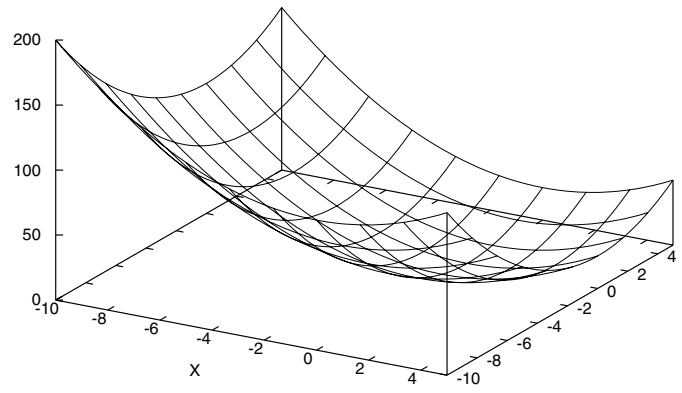

Fig. 2. A 2-dimensional parabolic objective fitness landscape with multiple peaks and a unique maximum.

This landscape has four peaks and a unique optimal fitness value of 200 at $\langle -10, -10 \rangle$ as shown in Figure 2.

Regardless of which simple EA you choose, increasing m increases the probability of finding the global optimum. Figure 3 illustrates this for a standard ES-style EA. What is plotted are best-so-far curves averaged over 100 runs for increasing values of m while keeping n fixed at 1 and using a non-adaptive Gaussian mutation operator with an average step size of 1.0.

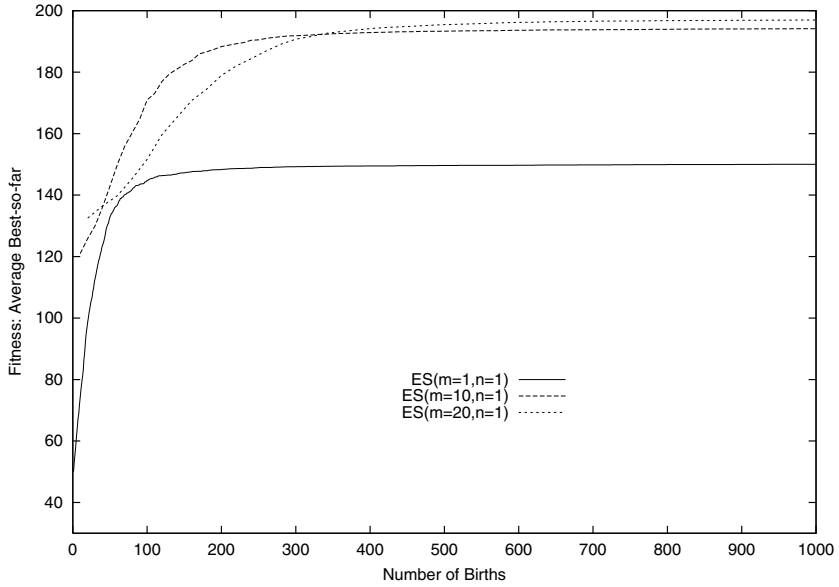

Fig. 3. The effects of parent population size on average best-so-far curves on a multi-peaked landscape.

In this particular case, we see that the *average* behavior of $ES(m, n = 1)$ improves with increasing m, but at a decreasing rate. The same thing can be shown for GA-like EAs and other non-traditional simple EAs.

For me, this is the simplest and clearest example of a relationship between fitness landscape properties and EA parameters, and provides useful insight into possible strategies for choosing parent population size. At the beginning of an EA run it is important to have sufficient parallelism to handle possible multi-modalities, while at the end of a run an EA is likely to have converged to a local area of the fitness landscape that is no more complex than Figure 1. That, in turn, suggests some sort of "annealing" strategy for parent population size that starts with a large value that decreases over time. The difficulty is in defining a computational procedure to do so effectively since it is difficult to predict *a priori* the time to convergence. One possibility is to set a fixed

time limit to an EA run and define an annealing schedule for it *a priori* (e.g., [30]).

Studies of this sort confirm our intuition about the usefulness of dynamically adapting population size, but they achieve it by making (possibly) suboptimal *a priori* decisions. Ideally, one would like parent population size controlled by the current state of an EA. This has proved to be quite difficult to achieve (see, for example, [39], [2], or [12]). Part of the reason for this is that there are other factors that affect parent population size choices as well, including the interacting effects of offspring population size (discussed in the next section), the effects of noisy fitness landscapes [20], selection pressure [13], and whether generations overlap or not [37]. As a consequence, most EAs used in practice today run with a fixed population size, the value of which may be based on existing off-line studies (e.g., [34]) or more likely via manual tuning over multiple runs.

6.2 Offspring Population Size n

By contrast the offspring population size n plays a quite different role in a simple EA. The current parent population reflects where in the solution space an EA is focusing its search. The number of offspring n generated reflects the amount of exploration performed using the current parent population before integrating the newly generated offspring back into the parent population. Stated in another way, this is the classic exploration-exploitation tradeoff that all search techniques face.

This effect can be seen quite clearly if we keep the parent population size m constant and increase the offspring population size n. Figure 4 illustrates this for the same ES-like algorithm used previously with $m = 1$. Notice how increasing n on this multi-peaked landscape results in a decrease in average performance unlike what we saw in the previous section with increases in m.

This raises an interesting EA question: should the parameters m and n be coupled as they are in GAs and EP, or is it better to decouple them as they are in ESs? Having fewer parameters to worry about is a good thing, so is there anything to be gained by having two population size parameters? There is considerable evidence to support an answer of "yes", including a variety of studies involving "steady-state" GAs (e.g., [42]) that have large m and $n = 1$, and other simple EAs (e.g., [27]). However, just as we saw with parent population size, defining an effective strategy for adapting the offspring population size during an EA run is quite difficult (see, for example, [26]). As a consequence, most EAs in use today run with a fixed offspring population size, the default value of which is based on some off-line studies and possibly manually tuned over several preliminary runs.

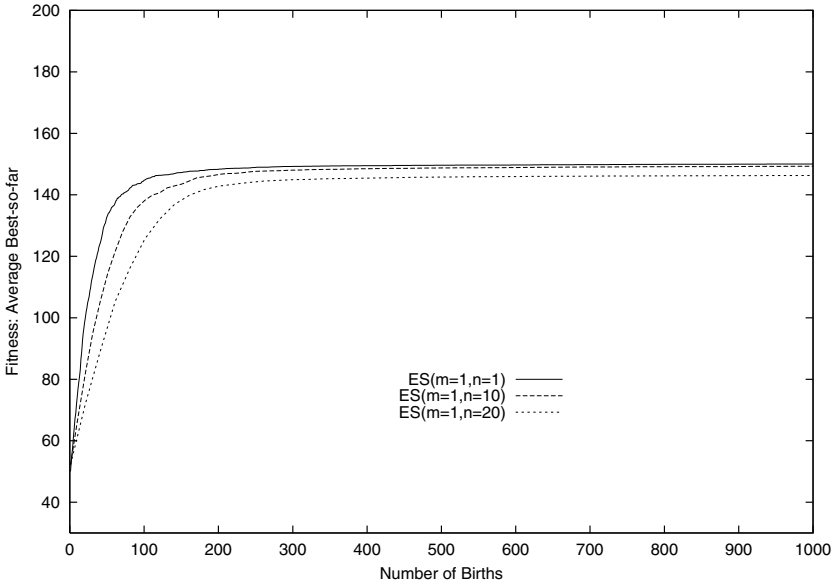

Fig. 4. The effects of offspring population size on average best-so-far curves on a multi-peaked landscape.

7 Selection

Selection procedures are used in EAs in two different contexts: as a procedure for selecting parents to produce offspring and as a procedure for deciding which individuals "survive" into the next generation. It is quite clear that the more "elitist" a selection algorithm is, the more an EA behaves like a local search procedure (i.e, a hill-climber, a "greedy" algorithm) and is less likely to converge to a global optimum. Just as we saw with parent population size m, this suggests that selection!pressure should be adapted during an EA run, weak at first to allow for more exploration and then stronger towards the end as an EA converges. However, the same difficulty arises here in developing an effective mechanism for modifying selection pressure as a function of the current state of an EA. For example, although fitness-proportional selection does vary selection pressure over time, it does so by inducing stronger pressure at the beginning of a run and very little at the end.

One source of difficulty here is that selection pressure is not as easy to "parameterize" as population size. We have a number of families of selection procedures (e.g, tournament selection, truncation selection, fitness-proportional selection, etc.) to choose from and a considerable body of literature analyzing their differences (see, for example, [19] or [15]), but deciding which family to choose or even which member of a parameterized family is still quite difficult, particularly because of the interacting effects with population size [13].

Another interesting question is whether the selection algorithm chosen for parent selection is in any way coupled with the choice made for survival selection. The answer is a qualified "yes" in the following sense: the overall selection pressure of an EA is due to the combined effects of both selection procedures. Hence, strong selection pressure for one of these (e.g, truncation selection) is generally combined with weak selection pressure (e.g., uniform selection) for the other. For example, standard ES and EP algorithms use uniform parent selection and truncation survival selection, while standard GAs use fitness-proportional parent selection and uniform survival selection.

One additional "parameterization" of selection is often made available: the choice between overlapping and non-overlapping generations. With non-overlapping models, the entire parent population dies off each generation and the offspring only compete with each other for survival. Historical examples of non-overlapping EAs include "generational GAs" and the "," version of ESs.

The alternative is an overlapping-generation model such as a steady-state GA, a $(\mu+\lambda)$-ES, or any EP algorithm. In this case, parents and offspring compete with each other for survival. The effects of this choice are quite clear. An overlapping-generation EA behaves more like a local search algorithm, while a non-overlapping-generation EA exhibits better global search properties [9]. Although an intriguing idea, I am unaware of any attempts to change this parameter dynamically.

8 Reproductive Operators

Most EAs produce offspring using two basic classes of reproductive mechanisms: an asexual (single parent) mutation operator and a sexual (more than one parent) recombination operator. Deciding if and how to parameterize this aspect of an EA is a complex issue because the effects of reproductive operators invariably interact with all of the other parameterized elements discussed so far.

At a high level of abstraction reproductive operators introduce variation into the population, counterbalancing the reduction in diversity due to selection. Intuitively, we sense that population diversity should start out high and decrease over time, reflecting a shift from a more global search to a local one. The difficulty is in finding an appropriate "annealing schedule" for population diversity. If selection is too strong, convergence to a local optimum is highly likely. If reproductive variation is too strong, the result is undirected random search. Finding a balance between exploration and exploitation has been a difficult-to-achieve goal from the beginning [18].

What is clear is that there is no one single answer. ES and EP algorithms typically match up strong selection pressure with strong reproductive variation, while GAs match up weaker selection pressure with weaker reproductive variation. Finding a balance usually involves holding one of these fixed (say, selection pressure) and tuning the other (say, reproductive variation).

This is complicated by the fact that what is really needed from reproductive operators is *useful* diversity. This was made clear early on in [33] via the notion of fitness correlation between parents and offspring, and has been shown empirically in a variety of settings (e.g., [31], [1], [28]). This in turn is a function of the properties of the fitness landscapes being explored which makes it clear that the ability to dynamically improve fitness correlation will improve EA performance. Precisely how to achieve this is less clear. One approach is to maintain a collection of plausible reproductive operators and dynamically select the ones that seem to be helping most (see, for example, [17] or [16]). Alternatively, one can focus on tuning specific reproductive operators to improve performance. This is a somewhat easier task about which considerable work has been done, and is explored in more detail in the following subsections.

8.1 Adapting Mutation

The classic one-parent reproductive mechanism is mutation that operates by cloning a parent and then providing some variation by modifying one or more genes in the offspring's genome. The amount of variation is controlled by specifying how many genes are to be modified and the manner in which genes are to be modified. These two aspects together determine both the amount and usefulness of the resulting variation.

Although easy to parameterize, the expected number of modified genes is seldom adapted dynamically. Rather, there are a number of studies that suggest a fixed value of 1 is quite robust, and is the default value for traditional GAs. By contrast, traditional ES algorithms mutate *every* gene, typically by a small amount. This seeming contradiction is clarified by noting that GAs make small changes in genotype space while the ES approach makes small changes in phenotype space.

Both of these approaches allow for dynamic adaptation. The "1/5" rule discussed earlier is a standard component of many ES algorithms. Adapting GA mutation rates is not nearly as common, and is typically done via a self-adaptive mechanism in which the mutation rate is encoded as a control gene on individual genomes (see, for example, [3]).

Mutating gene values independently can be suboptimal when their effects on fitness are coupled (i.e., epistatic non-linear interactions). Since these interactions are generally not known *a priori*, but difficult to determine dynamically during an EA run. The most successful example of this is the pair-wise covariance matrix approach used by the ES community [23]. Attempting to detect higher order interactions is computational expensive and seldom done.

8.2 Adapting Recombination

The classic two-parent reproductive mechanism is a recombination operator in which subcomponents of the parents' genomes are cloned and reassembled

to create an offspring genome. For simple fixed-length linear genome representations, the recombination operators have traditionally taken the form of "crossover" operators, in which the crossover points mark the linear subsegments on the parents' genomes to be copied and reassembled. For these kinds of recombination operators, the amount of variation introduced is dependent on two factors: how many crossover points there are and how similar the parents are to each other. The interesting implication of this is that, unlike mutation, the amount of variation introduced by crossover diminishes over time as selection makes the population more homogeneous.

This dependency on the contents of the population makes it much more difficult to estimate the level of crossover-induced variation, but has the virtue of self-adapting along the lines of our intuition: more variation early in an EA run and less variation as evolution proceeds. Providing *useful* variation is more problematic in that, intuitively, one would like offspring to inherit important combinations of gene values from their parents. However, just as we saw for mutation, which combinations of genes should be inherited is seldom known *a priori*. For example, it is well-known that the 1-point crossover operator used in traditional GAs introduces a distance bias in the sense that the values of genes that are far apart on a chromosome are much less likely to be inherited together than those of genes that are close together [25]. Simply increasing the number of crossover points reduces that bias but increases the amount of reproductive variation at the same time [10]. One elegant solution to this is to switch to a parameterized version of uniform crossover. This simultaneously removes the distance bias and provides a single parameter for controlling diversity [40]. Although an intriguing possibility, I am unaware of any current EAs that dynamically adjust parameterized uniform crossover.

Alternatively, one might consider keeping a recombination operator fixed and adapting the representation in ways that improve the production of useful diversity. This possibility is discussed in the next section.

9 Adapting the Representation

Perhaps the most difficult and least understood area of EA design is that of adapting its internal representation. It is clear that choices of representation play an extremely important role in the performance of an EA, but are difficult to automate. As a field we have developed a large body of literature that helps us select representations *a priori* for particular classes of problems. However, there are only a few examples of strategies for dynamically adapting the representation during an EA run. One example is the notion of a "messy GA" [21] in which the position of genes on the chromosome are adapted over time to improve the effectiveness 1-point crossover. Other examples focus on adapting the range and/or resolution of the values of a gene (see, for example, [38] or [82]). Both of these approaches have been shown to be useful in specific contexts, but are not general enough to be included in today's EA toolkits.

10 Parameterless EAs

Perhaps the ultimate goal of these efforts is to produce an effective and general problem-solving EA with *no externally visible* parameters. The No Free Lunch discussion at the beginning of this chapter makes it clear that this will only be achieved if there are effective ways to dynamically adapt various *internal* parameters. However, as we have seen throughout this chapter, there are very few techniques that do so effectively in a general setting for even one internal parameter, much less for more than one simultaneously. The few examples of parameterless EAs that exist in the literature involve simplified EAs in particular contexts (e.g., [29]). To me this is a clear indication of the difficulty of such a task.

An approach that appears more promising is to design an EA to perform internal restarts, i.e. multiple runs, and use information from previous runs to improve performance on subsequent (internal) runs. The most notable success in this area is the CHC algorithm developed by [14]. Nested ESs are also quite effective and based on a similar ideas but still require a few external parameter settings [35].

Clearly, we still have a long way to go in achieving the goal of effective parameterless EAs.

11 Summary

The focus in this chapter has be primarily of parameter setting in simple EAs for two reasons. First, this is where most of the efforts have been over the past 30 or more years. Second, although many new and more complex EAs have been developed (e.g., spatially distributed EAs, multi-population island models, EAs with speciation and niching, etc.), these new EAs do little to resolve existing parameter tuning issues. Rather, they generally exacerbate the problem by creating new parameters that need to be set.

My sense is that, for static optimization problems, it will to continue to be the case that particular types of EAs that have been pre-tuned for particular classes of problems will continue to outperform EAs that try to adapt too many things dynamically. However, if we switch our focus to time-varying fitness landscapes, dynamic parameter adaptation will have a much stronger impact.

References

1. L. Altenberg. The schema theorem and Price's theorem. In M. Vose and D. Whitley, editors, *Foundations of Genetic Algorithms 3*, pages 23–49. Morgan Kaufmann, 1994.

2. J. Arabas, Z. Michalewitz, and J. Mulawka. Gavaps - a genetic algorithm with varying population size. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 73–78. IEEE Press, 1994.
3. T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.
4. T. Bäck. On the behavior of evolutionary algorithms in dynamic fitness landscapes. In *IEEE International Conference on Evolutionary Computation*, pages 446–451. IEEE Press, 1998.
5. J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer, Boston, 2002.
6. L. Davis. Adapting operator probabilities in genetic algorithms. In *Third International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufmann, 1989.
7. L. Davis. *The Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.
8. K. De Jong. *Analysis of Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, 1975.
9. K. De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, Cambridge, MA, 2006.
10. K. De Jong and W. Spears. A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 5(1): 1–26, 1992.
11. A. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.
12. A. Eiben, E. Marchiori, and V. Valko. Evolutionary algorithms with on-the-fly population size adjustment. In X. Yao et al., editor, *Proceedings of PPSN VIII*, pages 41–50. Springer-Verlag, 2004.
13. A. Eiben, M. Schut, and A. de Wilde. Is self-adaptation of selection pressure and population size possible? In T. Runarsson et al., editor, *Proceedings of PPSN VIII*, pages 900–909. Springer-Verlag, 2006.
14. L. Eshelman. The CHC adaptive search algorithm. In G. Rawlins, editor, *Foundations of Genetic Algorithms 1*, pages 265–283. Morgan Kaufmann, 1990.
15. S. Ficici and J. Pollack. Game–theoretic investigation of selection methods used in evolutionary algorithms. In D. Whitley, editor, *Proceedings of CEC 2000*, pages 880–887. IEEE Press, 2000.
16. L. Fogel, P. Angeline, and D. Fogel. An evolutionary programming approach to self-adaptation on finite state machines. In J. McDonnell, R. Reynolds, and D. Fogel, editors, *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 355–365. MIT Press, 1995.
17. L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, 1966.
18. D. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer, Boston, 2002.
19. D. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In G. Rawlins, editor, *Proceedings of the First Workshop on Foundations of Genetic Algorithms*, pages 69–92. Morgan Kaufmann, 1990.
20. D. Goldberg, K. Deb, and J. Clark. Accounting for noise in sizing of populations. In D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 127–140. Morgan Kaufmann, 1992.

21. D. Goldberg, K. Deb, and B. Korb. Don't worry, be messy. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 24–30. Morgan Kaufmann, 1991.
22. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.
23. N. Hansen and A. Ostermeier. Completely derandomized step-size adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.
24. J.H. Holland. Outline for a logical theory of adaptive systems. *JACM*, 9:297–314, 1962.
25. J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.
26. T. Jansen, K. De Jong, and I. Wegener. On the choice of offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13(4):413–440, 2005.
27. Thomas Jansen and Kenneth De Jong. An analysis of the role of offspring population size in EAs. In W. B. Langdon et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference* (*GECCO 2002*), pages 238–246. Morgan Kaufman, 2002.
28. T. Jones. *Evolutionary algorithms, fitness landscapes, and search*. PhD thesis, University of New Mexico, 1995.
29. C. Lima and F. Lobo. Parameter-less optimization with the extended compact genetic algorithm and iterated local search. In K. Deb et al, editor, *Proceedings of GECCO-2004*, pages 1328–1339. Springer-Verlag, 2004.
30. S. Luke, G. Balan, and L. Panait. Population implosion in genetic programming. In *Proceedings of GECCO-2003*, volume 2724, pages 1729–1739. Springer LNCS Series, 2003.
31. B. Manderick, M. de Weger, and P. Spiessens. The genetic algorithm and the structure of the fitness landscape. In R. K. Belew and L. B. Booker, editors, *The Fourth International Conference on Genetic Algorithms and Their Applications*, pages 143–150. Morgan Kaufmann, 1991.
32. R. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*. Springer-Verlag, Berlin, 2004.
33. George Price. Selection and covariance. *Nature*, 227:520–521, 1970.
34. I. Rechenberg. Cybernatic solution path of an experimental problem. In *Library Translation 1122*. Royal Aircraft Establishment, Farnborough, 1965.
35. I. Rechenberg. *Evolutionsstrategie '94*. Frommann-Holzboog, Stuttgart, 1994.
36. H.-P. Schwefel. *Evolutionsstrategie und numerishe Optimierung*. PhD thesis, Technical University of Berlin, Berlin, Germany, 1975.
37. H.-P. Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, New York, 1995.
38. C. Shaefer. The ARGOT strategy: adaptive representation genetic optimizer technique. In J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 50–58. Lawrence Erlbaum, 1987.
39. R. Smith. Adaptively resizing populations: an algorithm and analysis. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms and their Applications*, page 653. Morgan Kaufmann, 1993.
40. W. Spears and K. De Jong. On the virtues of parameterized uniform crossover. In R. K. Belew and L. B. Booker, editors, *International Conference on Genetic Algorithms*, volume 4, pages 230–236. Morgan Kaufmann, 1991.

41. W.M. Spears. Adapting crossover in evolutionary algorithms. In J. McDonnell, R. Reynolds, and D.B. Fogel, editors, *Proceedings of the Fourth Conference on Evolutionary Programming*, pages 367–384. MIT Press, 1995.
42. D. Whitley. The Genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann, 1989.
43. D. Whitley, K. Mathias, and P. Fitzhorn. Delta coding: an iterative search strategy for genetic algorithms. In R. K. Belew and L. B. Booker, editors, *Proceeding of the Fourth International Conference on Genetic Algorithms*, pages 77–84. Morgan Kaufmann, 1991.
44. D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.