

Parameter Control Methods for Selection Operators in Genetic Algorithms

P. Vajda, A.E. Eiben*, and W. Hordijk

* Corresponding author: Vrije Universiteit Amsterdam
gusz@cs.vu.nl <http://www.cs.vu.nl>

Abstract. Parameter control is still one of the main challenges in evolutionary computation. This paper is concerned with controlling selection operators on-the-fly. We perform an experimental comparison of such methods on three groups of test functions and conclude that varying selection pressure during a GA run often yields performance benefits, and therefore is a recommended option for designers and users of evolutionary algorithms.

1 Introduction

Evolutionary computing (EC) has become a proven problem solving technology over the past few decades [6]. However, the performance of *evolutionary algorithms* (EAs) depends largely on their parameters, such as population size, selection pressure, crossover and mutation rates. Choosing good values for EA parameters before an EA run (parameter tuning) and/or appropriately varying parameter values during an EA run (parameter control) is still one of the main challenges of the field [4]. The ultimate goal is to develop methods that are capable of adjusting the parameter values to a given problem, and also to the different stages of the search process. Traditionally, most attention has been paid to parameters of the variation operators (in particular the crossover and mutation rates) and –to a much lesser extent– to population size [3, 13]. In contrast, on-the-fly control of selection operators has received little attention, and the few available results are scattered throughout the literature. Note that in EAs that use evolution strategy-like population management, that is, (μ, λ) or $(\mu + \lambda)$, the offspring population size λ determines the selection pressure, and can thus be used to control the selection operator [11]. However, in this study we focus on *genetic algorithms* (GAs), in particular steady-state GAs that inherently feature two selection mechanisms: one for parent selection and one for survivor selection (replacement). To keep things simple, we keep the replacement strategy fixed and focus on parent selection. The objectives of this paper are twofold: To present an overview of known parameter control methods for (parent) selection operators, and to perform an experimental comparison of these methods with each other and a benchmark GA (a simple GA with K -tournament selection). To broaden the support base of our empirical findings we use three groups of test functions: (1) a collection of popular test functions for EA research, (2) known

“GA teasers”, and (3) a set of abstract fitness landscapes created by a randomized generator. The results show that the winning policy¹ largely depends on the group of test functions. Consequently, the overall winner depends on the relative weights these groups are given when calculating the final score for each policy. However, the data provides sufficient support for the superiority of on-the-fly control of selection pressure (as opposed to keeping it constant during a GA run) and an indication of a control mechanism that is capable of regulating selection pressure by itself.

2 Parameter control for selection operators

In this section we present the contestants of our experimental comparison, including K -tournament selection with a fixed K (benchmark), K -tournament selection with changing values of K (new method, introduced here), and three control methods from the literature. Despite an extensive literature study we could not find any other existing control methods, except variations on the Boltzmann selection, e.g., [14].

Fixed tournament size: As a benchmark for the parameter control methods, we use a *simple genetic algorithm* (SGA) with tournament selection with fixed tournament size K . We consider 10 values, $K = 1, 2, 4, 6, 8, 12, 16, 24, 32, 64$, for a comparison with the control methods, where $K = 1$ amounts to uniform random selection (compensated by the replacement strategy).

Deterministic tournament-size control: This method (DTC) is the most straightforward way of adjusting the selection pressure during the search. The tournament size K is a deterministic function of the time step (generation) t :

$$K(t) = \begin{cases} \frac{t(p_2-p_1)}{1000} + p_1 & \text{if } t \in [0, 1000] \\ p_2 & \text{otherwise} \end{cases} \quad (1)$$

where p_1 and p_2 are parameters of the method. In words, the tournament size increases linearly from p_1 to p_2 (or decreases if $p_1 > p_2$) for the first 1000 generations, after which it stays fixed at p_2 .

Boltzmann selection with annealing: A more sophisticated deterministic parameter control method is Boltzmann selection with a Riemann-Zeta annealing schedule (BSR). The probability $p(x, t)$ of selecting an individual x from the population P_t at time step t is calculated as:

$$p(x, t) = \frac{e^{\gamma_t \cdot f(x)}}{\sum_{y \in P_t} e^{\gamma_t \cdot f(y)}}, \quad \gamma_t = \gamma_0 \sum_{k=1}^t \frac{1}{k^\alpha} \quad (2)$$

where γ_t is the annealing temperature. γ_0 and α are parameters of the method. **Self-adaptive tournament size:** It is also possible to let the parameter value

¹ By policy we mean either a control mechanism, or the value of K in the SGA using K -tournament selection.

be controlled by the evolutionary process itself. In [5], a self-adaptive tournament size (SAT) method was introduced, where an extra parameter $k \in (0, 1)$ is added to each individual's chromosome, which represents this individual's contribution to the overall tournament size parameter $K = \left\lceil \sum_{i=1}^N k_i \right\rceil$, where N is the population size. Note that $K \in [1, N]$. The self-adaptive mechanism for mutation rates in GAs as described in [2] is then used to mutate the individual values of k :

$$k' = \left(1 + \frac{1-k}{k} \cdot e^{-\gamma \cdot N(0,1)} \right)^{-1} \tag{3}$$

where γ is a learning rate which allows for control of the adaptation speed. This mutation mechanism has the desirable property that if $k \in (0, 1)$, then also $k' \in (0, 1)$. A variant of this, called *hybrid* self-adaptive tournament size (HSAT; see [5] for details) adjusts an individual's parameter value k according to whether its fitness value is better or worse than that of its parent:

$$k' = \begin{cases} (1 + \frac{1-k}{k} e^{-|\gamma N(0,1)|})^{-1} & \text{if } f(x') \geq f(x) \\ (1 + \frac{1-k}{k} e^{|\gamma N(0,1)|})^{-1} & \text{otherwise} \end{cases} \tag{4}$$

where x represents the parent and x' the offspring. Of course the \geq sign will be a \leq for minimization problems. Again, γ is a learning rate. This method is self-adaptive as in the original SAT method, but it also uses feedback from the search (in particular the fitness values of parents and offspring).

Fuzzy tournament selection: This method (FTS) is based on the adaptation of selection in [8]. The concept of fuzzy logic (FL) itself was conceived by Zadeh [7]. Here, we use two inputs, genotypic and phenotypic diversity, and one output, μ , the modification parameter. *Genotypic diversity* (GD) is calculated as follows:

$$GD = \frac{\bar{d} - d_{min}}{d_{max} - d_{min}} = \frac{\frac{1}{N} \sum_{i=1}^N D(C_{best}, C_i) - \min \{D(C_{best}, C_i)\}}{\max \{D(C_{best}, C_i)\} - \min \{D(C_{best}, C_i)\}} \tag{5}$$

where N is the population size, $D(x, y) \in [0, 1]$ is the (normalized) Hamming distance of genomes x and y , and C_{best} is the chromosome of the best individual in the population. *Phenotypic diversity* (PD) is calculated as follows:

$$PD = \begin{cases} \frac{f_{best}}{f} & \downarrow \text{ for minimization problems} \\ \frac{1-f_{best}}{1-f} & \uparrow \text{ for maximization problems} \end{cases} \tag{6}$$

The modification parameter μ is calculated using the fuzzy logic rules shown in figure 1. The tournament size K in generation t is set to $K_t = (\mu + 0.5) \cdot K_{t-1}$. The label set for the output is {Low, Medium, High} (figure 1), calculated as follows:

GD \ PD	Low	Medium	High
Low	Low	Medium	High
Medium	Low	High	High
High	Low	Low	High

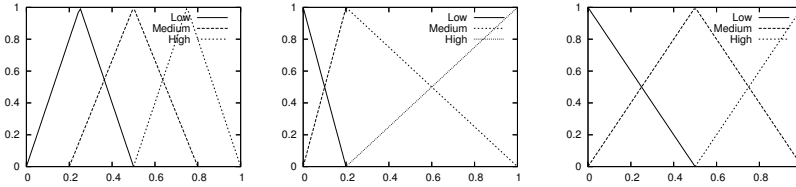


Fig. 1. Fuzzy sets of GD (left), PD (center), and μ (right).

3 Test suite

To compare the different parameter control methods for selection operators, we have composed an extensive test suite that includes (1) some popular (mostly difficult) functions such as Ackley’s function and Rastrigin’s function, (2) well known “GA teasers” such as the long path problem and the royal road function, and (3) a set of instances from the multimodal problem generator of Spears. In this section, a brief review of the used test functions is presented.

Popular test functions

These functions are used very often in experimental EC research. Therefore, we restrict ourselves to simply listing them: the Sphere, Generalized Rosenbrock, Generalized Ackley, Rastrigin, and Griewank functions [16, 1]. These are all defined as minimization problems on real numbers in L dimensions.

GA teasers

These functions were either specifically constructed to be deceptive or to test certain assumptions about how the genetic algorithm performs its search. They are all defined as maximization problems on bit strings of length L .

Long path: This function [9] is difficult mainly because (at least from a hill-climbing point of view) the only way to the global optimum is very long and narrow, with the length of this path increasing exponentially with the problem size L . So, even though this problem is unimodal, it may take an exponentially long time to reach the optimum.

Ugly: This function [18] presents a deceptive function. To calculate an individual’s overall fitness value F , first its chromosome is cut into 3-bit substrings $b_i b_{i+1} b_{i+2}$, and each of these substrings is assigned a fitness value f . The overall fitness of an individual is the average over all the substrings:

$$F(\mathbf{b}) = \frac{\sum_{i=0}^{L/3-1} f(b_{3i+1}b_{3i+2}b_{3i+3})}{L}, \quad f(\mathbf{b}) = \begin{cases} 2 & \text{if } \mathbf{b} \in \{0 * *\} \\ 3 & \text{if } \mathbf{b} = 111 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

Royal road: This function [15] was originally introduced to investigate the validity of the building block hypothesis [10]. The main idea of the function is to compare a given individual with a set of schemata with a varied number of defined bits and containing various pre-defined building blocks. For each of the

schemata that the individual is an instance of, it receives one point. The overall fitness value of the individual is then a weighted sum of the received points.

Random landscapes by the multimodal problem generator

A useful and tunable test suite for testing GAs can be created with the *multimodal problem generator* (MPG) of Spears [12, 17]. Here, we generate problem instances with $P = 1, 2, 5, 10, 25, 50, 100, 250, 500$, and 1000 peaks whose heights are linearly distributed and where the lowest peak has height 0.5. The fitness of an individual is measured by the Hamming distance (HD) between the individual and the nearest peak, scaled by the height of that peak. The nearest peak is determined by

$$Peak_{near}(\mathbf{b}) = \min_{i=1}^P (HD(\mathbf{b}, Peak_i)) \quad (8)$$

In case of multiple peaks at the same (minimum) distance, the highest of these peaks is chosen. The evaluation function of an individual is then:

$$F(\mathbf{b}) = \frac{L - HD(\mathbf{b}, Peak_{near}(\mathbf{b}))}{L} \cdot height(Peak_{near}(\mathbf{b})) \quad (9)$$

4 Experimental setup

We used all parameter control methods for selection described in section 2 on all test functions in the test suite described in section 3. For each method, 100 runs are performed on each problem instance (with different random seeds for each run), for a maximum of 5000 fitness evaluations per run. Three standard performance measures are used: *Success Rate* (SR; fraction of times the global optimum was found), *Average number of Evaluations to Solution* (AES; only calculated over those runs where the global optimum was found), and *Mean Best Fitness* (MBF). These performance measures are calculated over the 100 runs for each combination of control method and problem instance. We used two variants of the simple genetic algorithm, one with a floating point representation and one with a bit string representation.

For the popular test functions we use a floating point GA, cf. table 1. Note that all problems in this group are minimization problems in $L = 30$ dimensions/variables. Float vector mutation works as follows. Individuals are represented as $\mathbf{x} = \langle x_1, x_2, \dots, x_L \rangle$. The mutation rate $p \in [0, 1]$ determines whether a particular value x_i will be mutated. If so, then x'_i is drawn from a uniform random distribution from the interval $I = [x_i - r, x_i + r]$, where r is defined as:

$$r = \begin{cases} p \frac{b-a}{2} & \text{if } a \leq x_i - p \frac{b-a}{2} \text{ and } b \geq x_i + p \frac{b-a}{2} \\ x_i - a & \text{if } a > x_i - p \frac{b-a}{2} \\ b - x_i & \text{if } b < x_i + p \frac{b-a}{2} \end{cases} \quad (10)$$

where the domain of the chromosome is $[a, b]^L$. The used domain values for the different functions are $[-5.12, 5.12]$ for the Sphere, Rosenbrock, and Rastrigin functions, $[-32.768, 32.768]$ for the Ackley function, and $[-600, 600]$ for the Griewank function.

Representation	floating point	bit string
GA model	steady-state	steady-state
Optimization	minimization	maximization
Chromosome length (L)	30	29, 30, 64, 100
Population size	100	100
Selection	all from section 2	all from section 2
Crossover	uniform ($p_c = 0.7$)	uniform ($p_c = 0.7$)
Mutation	float vector ($p_m = 0.1$)	bit flip ($p_m = 0.01$)
Replacement	delete worst	delete worst

Table 1. Details of the GAs used in the experiments.

Since the GA teasers and the multimodal problems are defined in terms of bit strings, we use a GA with a bit-string representation here, see Table 1, right column. The chromosome lengths depend on the individual problem, and here we use $L = 29$ for the long path, $L = 30$ for the ugly problems, $L = 64$ for the royal road problem (with the "building blocks" being of length 8), and $L = 100$ for the multimodal problem instances. The parameter values come from the definition of the functions and most relevant articles. If there were no sufficient and accepted parameter values in the literature (BSR, SGA and DTC), we tried to cover the parameter space by using multiple values.

Finally, the parameter values for the various control methods need to be specified. For the deterministic tournament-size control (DTC) method, we use $p_1, p_2 \in \{2, 7, 17\}$, i.e., six combinations of p_1 and p_2 values such that $p_1 \neq p_2$. For the Boltzmann selection (BSR) method, we use the following values for the (γ_0, α) parameter combination: (40.89, 1.0001), (60.57, 1.1), (154.30, 1.5), and (243.2, 2). And for the self-adaptive tournament size (SAT) and the hybrid version (HSAT), we use $\gamma = 0.22$.

5 Results

For a direct comparison, we calculated a normalized score for each method. On each test function, the worst method gets a score of 0, the best one gets 1, and the remaining ones get a score that is linearly dependent on their result ($RES_{A,T}$):

$$score(A, T) = \begin{cases} \frac{RES_{A,T} - MIN(RES_{.,T})}{MAX(RES_{.,T}) - MIN(RES_{.,T})} & \text{if } \uparrow \\ \frac{MAX(RES_{.,T}) - RES_{A,T}}{MAX(RES_{.,T}) - MIN(RES_{.,T})} & \text{if } \downarrow \end{cases} \quad (11)$$

where A is a selection method and T a test function. Recall that there are three groups of test functions (*Popular*, *Teasers*, and *MPG*). Each method gets a score between 0 and 1 for each group by averaging the scores over all test functions in the group. The total score of a method is the average of its scores for all three groups. This total score is calculated for each of the performance measures (*SR*, *AES*, *MBF*). The average of these three scores is then the final score of a method (Table 2, last column). Of course we can also calculate a total score for

each performance measure first, and then calculate the final score as an average over these three scores. Table 2 shows the results of both these calculations, i.e., either by performance measure (columns 2–4) or by test function group (columns 5–7), both leading to the same final score (last column). As the table shows, the best overall score is achieved by the HSAT method.

	SR	AES	MBF	Teasers	Popular	MPG	Final score
HSAT	0.832	0.764	0.828	0.780	0.977	0.667	0.808
SGA 1	0.949	0.568	0.840	0.889	0.869	0.599	0.786
DTC 2 7	0.720	0.863	0.737	0.682	0.958	0.679	0.773
SGA 2	0.822	0.695	0.748	0.715	0.918	0.632	0.755
SAT	0.752	0.810	0.684	0.692	0.927	0.627	0.749
DTC 7 17	0.712	0.875	0.631	0.548	0.956	0.713	0.739
SGA 4	0.734	0.824	0.657	0.625	0.922	0.668	0.739
DTC 2 17	0.752	0.751	0.689	0.501	0.967	0.724	0.731
FTS	0.714	0.870	0.609	0.594	0.919	0.682	0.731
SGA 8	0.755	0.808	0.619	0.498	0.934	0.751	0.727
DTC 7 2	0.760	0.852	0.549	0.531	0.874	0.756	0.720
SGA 6	0.667	0.852	0.590	0.541	0.937	0.630	0.703
SGA 16	0.648	0.860	0.589	0.507	0.929	0.661	0.699
SGA 12	0.681	0.860	0.541	0.470	0.930	0.682	0.694
SGA 24	0.630	0.886	0.458	0.409	0.909	0.655	0.658
BRS 2	0.653	0.880	0.441	0.402	0.901	0.671	0.658
SGA 64	0.661	0.886	0.417	0.437	0.838	0.690	0.655
DTC 17 2	0.748	0.769	0.434	0.504	0.737	0.710	0.650
SGA 32	0.618	0.879	0.395	0.413	0.813	0.665	0.631
DTC 17 7	0.666	0.759	0.380	0.370	0.784	0.653	0.602

Table 2. Scores of selection methods by the three performance measures and by the three groups of test functions. The final score is the average of these.

Assigning equal weights to each group of test functions or performance measures is rather arbitrary, and obviously the overall winner could change if we use different weights (which would reflect different relative importances for the different types of test functions or performance measures). In Figure 2 we present the overall winner for all possible combinations of weights. In the three corners of each graph either the three test function groups (left) or the three performance measures (right) are shown, representing a weight of 1 for the group belonging to this corner and 0 for the other two. The equally-weighted average (as given in table 2) is the point in the center. As the figure shows, HSAT remains the best method for a wide range of relative weights (in both cases). The SGA 1 method also performs quite well, but mostly on the GA teasers group, which represents the most difficult test functions. On these functions, a low selection pressure works best (allowing for more exploration), and with the SGA 1 method there is no selection pressure at all (parents are chosen at random with uniform probability). As far as we know, Figure 2 introduces a new way of presenting EA performance results which provides useful insights for algorithm comparisons relative to importance given to different test functions or performance measures.

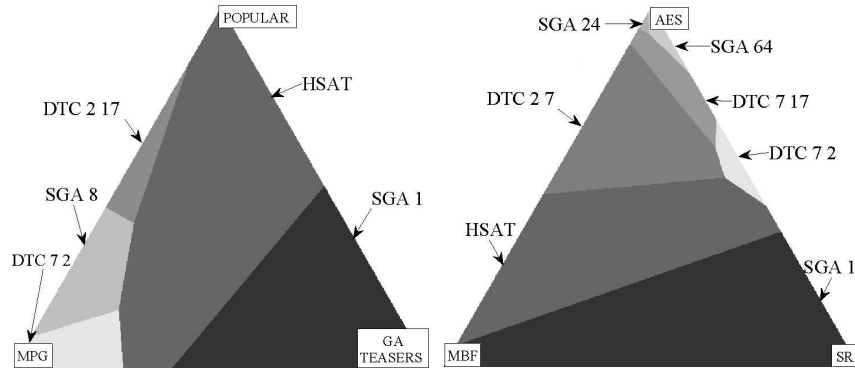


Fig. 2. The best selection methods for different weights of the test function groups (left) and of the performance measures (right). The equally-weighted average is the point in the center.

Given these results, we are now able to answer the question on which test functions does each selection method perform well, and why?

SGA: This is the benchmark for the parameter control methods. Our results show that the optimal tournament size depends strongly on the test function. There are some easy landscapes where the GAs with higher selection pressure perform better (*MPG*), and there are difficult landscapes where the SGA behaves in the opposite way (*GA teasers*). However, the optimal tournament size can be anywhere in between, so we can conclude that there is no perfect (constant) tournament size.

DTC: The results show clearly that there exist landscapes where DTC performs better than the SGA. On the *Ackley* function, e.g., the gradual increase in tournament size leads to a much better performance than all of the SGA variants. This confirms that the optimal tournament size indeed requires adaptation during the runtime of the GA.

BRS: This proves to be a good method on *MPG* problems. However, since it uses a deterministic parameter control, it does not work well on more difficult problems. Due to the parameter settings, the BRS method reaches a very high selection pressure when the population is close to the optimum, which makes the method very fast. Nevertheless, if the landscape does not need a high selection pressure, this method does not give satisfactory results.

HSAT: Overall, this method performs well on all test functions. For the Spears functions, the control parameter (K) increases very much at first, and then drops down again when the population is very close to a peak. This is due to the fact that recombination can initially create better offspring quite easily, so a higher selection pressure converges more quickly to a good area in the landscape, whereas once it becomes more difficult to find better offspring, a lower selection pressure could be beneficial (allowing for more exploration). On more difficult problems, like the *GA teasers*, the parameter value stays low all throughout, because it is always difficult to find fitter offspring, and a low selection pressure

(more exploration) is better – as the good score of SGA 1 indicates. Whenever a landscape requires a high selection pressure, the control parameter always reaches the upper limit (e.g., on most popular functions).

FTS: Although this method is not performing as well as the others, its adaptation is handled quite easily. The fuzzy system does actually not directly control the selection parameter, but it merely multiplies it. Therefore, it usually becomes too high, and thus the selection parameter can not converge to an optimal value.

Finally, we can answer similar questions about the test functions: How “difficult” is a test function and which methods perform well on it?

Popular functions: This group of functions has a variety of difficulty levels, but are overall easier than the GA-teasers. Our results suggest that those methods which can adjust the selection pressure quickly work better. The HSAT method appears to perform the best overall.

GA-teasers: These are the most difficult test functions. Generally speaking, the methods that maintain a low selection pressure perform better, because the population can leave the (bad) local minima more easily. The SGA 1 method appears to be the best in this respect, with HSAT being second.

MPG: Because of the special properties of these functions, the greedy methods found the optimum faster. A high selection pressure gives an advantage on these functions. Here, the DTC (7, 2) method is the best.

6 Conclusions

The main conclusions that can be drawn from our comparison of the different selection methods are as follows:

- 1) On-the-fly adjustment of the parameter(s) regulating selection is superior to using a constant value. First, because this enables the GA to use different levels of selection pressure on different landscapes without human tuning. Second, because using different parameter values at different stages of the GA search can significantly improve the performance.
- 2) The best method to control selection depends on the problem (type), but our results indicate that HSAT is the generally best policy. It is capable of appropriately adjusting the selection pressure according to the search results so far *and* to the given landscape.
- 3) The general strategy of the HSAT method is that as long as improvements are found, the selection pressure will increase to exploit good solutions. However, when the population is stuck or has converged too much, the selection pressure will decrease to encourage more exploration. These parameter adjustment decisions can be made based on genotypic and/or phenotypic diversity, or from the fitness differences between parents and offspring. Here we only tested one possible implementation of the main idea.

In the future, more detailed investigations will have to be done, but our experiments have already provided useful results and a proof-of-principle. In general, the use of parameter control for selection operators is a recommendable design heuristic for EA users.

References

1. T. Bäck. *Evolutionary algorithms in theory and practice*. Oxford University Press, 1996.
2. T. Bäck and M. Schütz. Intelligent mutation rate control in canonical genetic algorithms. In Z. W. Ras and M. Michalewicz, editors, *ISMIS '96: Proceedings of the 9th International Symposium on Foundations of Intelligent Systems*, volume 1079 of *Lecture Notes in Computer Science*, pages 158–167, London, UK, 1996. Springer-Verlag.
3. Kenneth DeJong. Parameter setting in EAs: a 30 year perspective. In *Parameter Setting in Evolutionary Algorithms*, pages 1–18. Springer, 2007.
4. A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.
5. A.E. Eiben, M.C. Schut, and A.R. de Wilde. Boosting genetic algorithms with self-adaptive selection. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1584–1589, 2006.
6. A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing, Corrected reprint*. Springer-Verlag, 2007.
7. R.R. Yager et al. (eds.). *Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh*. John Wiley, New York, 1987.
8. F. Herrera and M. Lozano. Fuzzy genetic algorithms: issues and models. Technical report, No. 18071, Granada, Spain, 1999.
9. C. Hohn and C. Reeves. Are long path problems hard for genetic algorithms? In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the 4th Conference on Parallel Problem Solving from Nature*, number 1141 in *Lecture Notes in Computer Science*, pages 134–143. Springer, Berlin, Heidelberg, New York, 1996.
10. J.H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, 1975.
11. 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.
12. K.A. De Jong and W.M. Spears. A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, (5):1–26, 1992.
13. Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.
14. S. W. Mahfoud and D. E. Goldberg. Parallel recombinative simulated annealing: a genetic algorithm. *Parallel Computing*, 21(1):1–28, 1995.
15. M. Mitchell, S. Forrest, and J. H. Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. In Francisco J. Varela and Paul Bourguine, editors, *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, 1991*, pages 245–254, Paris, 11–13 1992. A Bradford book, The MIT Press.
16. H.-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, 1995.
17. W. Spears. *Evolutionary algorithms: the role of mutation and recombination*. Springer, 2000.
18. D. Whitley. Fundamental principles of deception. In Morgan Kaufmann, editor, *Foundations of Genetic Algorithms*, pages 221–241. Morgan Kaufmann, San Francisco, 1991.