Parameter Control in Evolutionary Algorithms

Jiří Kubalík
Department of Cybernetics, CTU Prague

http://cw.felk.cvut.cz/doku.php/courses/a0m33eoa/start

Parameter Control in Evolutionary Algorithms

An EA is a metaheuristic whose components need to be instantiated and properly tuned in order to yield a fully functioning algorithm:

- components – representation, selection and replacement strategy, recombination and mutation operators, ...
- strategy parameters – population size, probability of crossover and mutation, parameter of selection, etc.

The values of these algorithms greatly determine whether the algorithm will find an optimal (or near-optimal) solution, and whether it will find such a solution effectively.

Two major forms of setting the parameter values

- **Parameter tuning** – finding good values for the parameters before the run of the algorithm, and then running the algorithm with these values, which remain fixed during the run.
- **Parameter control** – starts a run with initial parameter values that change during the run.

Parameter Tuning

Typically done by experimenting with different values and selecting the ones that give the best results on the test problems at hand.

Technical drawbacks to parameter tuning:

- Parameters are not independent, but trying all different combinations systematically is practically impossible.
- It is heavily based on personal experience and is guided by a mixture of rules of thumb.
- The process of parameter tuning is time consuming, even if parameters are optimised one by one, regardless of their interactions.
- For a given problem the selected parameter values are not necessarily optimal, even if the effort made for setting them was significant.
- There are no generally good parameter settings since specific problems require specific setups for satisfactory performance.

Parameter Tuning

Typically done by experimenting with different values and selecting the ones that give the best results on the test problems at hand.

Technical drawbacks to parameter tuning:

- Parameters are not independent, but trying all different combinations systematically is practically impossible.
- It is heavily based on personal experience and is guided by a mixture of rules of thumb.
- The process of parameter tuning is time consuming, even if parameters are optimised one by one, regardless of their interactions.
- For a given problem the selected parameter values are not necessarily optimal, even if the effort made for setting them was significant.
- There are no generally good parameter settings since specific problems require specific setups for satisfactory performance.
- A run of an EA is an intrinsically dynamic, adaptive process. Thus, different values of parameters might be optimal at different stages of the evolutionary process.

  Ex.: Large mutation steps can be good in the early generations, helping the exploration.

  Small mutation steps do better in the late generations fine-tuning the suboptimal solution.

Parameter Tuning: F-Race

F-Race [Birattari02] – procedure that empirically evaluates a set of candidate configurations by discarding bad ones as soon as statistically sufficient evidence is gathered against them.

- The process starts from a given finite pool of candidate configurations.
- If sufficient evidence is gathered that some candidate is inferior to at least another one, such a candidate is dropped from the pool and the procedure is iterated over the remaining ones.

The methodology can be applied to repetitive problems – problems where many similar instances appear over time.

F-Race: Formal Definition of the Configuration Problem

- Θ is the finite set of candidate configurations.
- I is the possibly infinite set of instances.
- P_I is a probability measure over the set I of instances – indicates the probability that the instance i is selected for being solved.
- $t : I \rightarrow \mathfrak{R}$ is a function associating to every instance the comput. time that is allocated to it.
- $\mathbf{c}(\theta, i) = \mathbf{c}(\theta, i, t(i))$ is a random variable representing the cost of the best solution found by running configuration θ on instance i for $t(i)$ seconds.
- $C \subset \mathfrak{R}$ is the range of $\mathbf{c}$, that is, the possible values for the cost of the best solution found in a run of a configuration $\theta \in \Theta$ on an instance $i \in I$.
- P_C is a probability measure over the set C: $P_C(c|\theta, i)$ indicates the probability that c is the cost of the best solution found by running for $t(i)$ seconds configuration θ on instance i.
- $\mathcal{C}(\theta) = \mathcal{C}(\theta|\Theta, I, P_I, P_C, t)$ is the criterion that needs to be optimized with respect to θ.

F-Race: Formal Definition of the Configuration Problem

The configuration problem is formally described by the 6-tuple $< \Theta, I, P_I, P_C, t, \mathcal{C} >$.

The solution of this problem is the configuration θ^* such that:

$$\theta^* = \text{argmin} \mathcal{C}_\theta(\theta)$$

Here, the optimization of the expected value of the cost $\mathbf{c}(\theta, i)$ is considered:

$$\mathcal{C}(\theta) = E_{I,C}[\mathbf{c}(\theta, i)] = \int_I \int_C c(\theta, i) \mathrm{d}P_C(c|\theta, i) \mathrm{d}P_I(i)$$

where the expectation is considered with respect to both P_I and P_C.

F-Race: Formal Definition of the Configuration Problem

The configuration problem is formally described by the 6-tuple $< \Theta, I, P_I, P_C, t, \mathcal{C} >$.

The solution of this problem is the configuration θ^* such that:

$$\theta^* = \text{argmin} \mathcal{C}_\theta(\theta)$$

Here, the optimization of the expected value of the cost $\mathbf{c}(\theta, i)$ is considered:

$$\mathcal{C}(\theta) = E_{I,C}[\mathbf{c}(\theta, i)] = \int_I \int_C c(\theta, i) \mathrm{d}P_C(c|\theta, i) \mathrm{d}P_I(i)$$

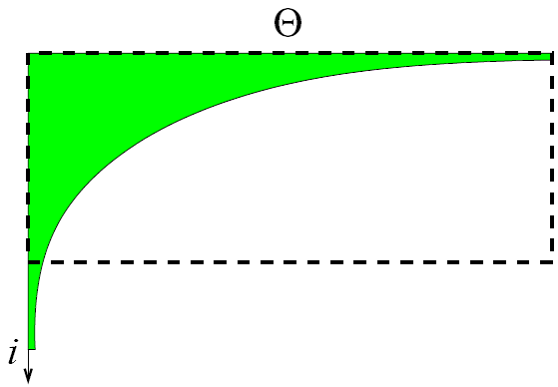
where the expectation is considered with respect to both P_I and P_C.

- The analytical solution of the integrals is not possible since the measures of P_I and P_C are not explicitly available.
- The integrals will be estimated in a Monte Carlo fashion on the basis of a training set of instances.

Idea of Racing Algorithms

Brute force approach – estimate the quantities P_C and P_I by means of a sufficiently large number of runs of each candidate on a sufficiently large set of training instances.

- The training set must be defined prior the computation – how large?
- How many runs of each configuration on each instance should be performed?
- The same computational resources are allocated to each configuration – wasting time on poor configs!



Racing algorithm – provides a better allocation of computational resources among candidate configurations and allows for a clean solution to the problems with fixing the number of instances and the number of runs to be considered.

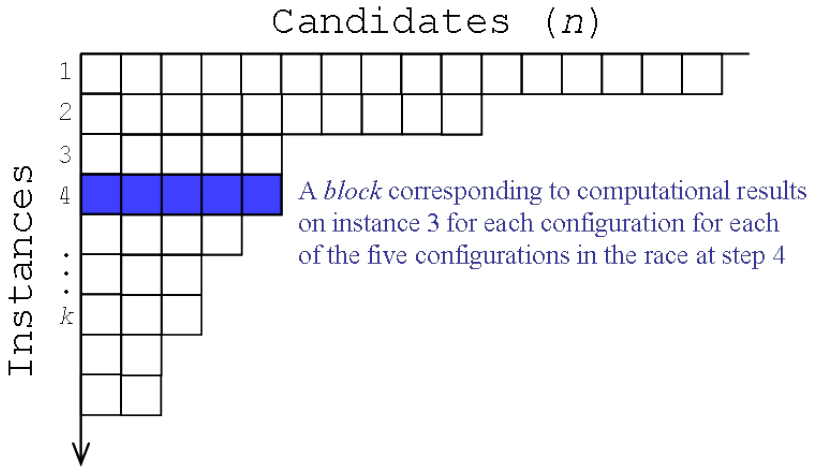
- Sequentially evaluates candidate configs and discards poor ones as soon as statistically sufficient evidence is gathered against them.
- Elimination of the inferior candidates speeds up the procedure and allows to evaluate the promising ones on more instances.
- As the evaluation proceeds, the race focuses more and more on the promising configurations.

F-Race: Algorithm

- k is the current step of the race process and $n = |\Theta_{k-1}|$ configurations are still in the race.
- $\underline{i}$ is a random sequence of training instances; $\underline{i}_k$ is drawn from I according to P_I, independently for each k.
- $\underline{c}^k(\theta, \underline{i})$ is an array of k terms; $c(\theta, \underline{i}_l)$ is the cost of the best solution found by configuration θ on instance $\underline{i}_l$.

  For a given θ, the array $\underline{c}^k$ of length k can be obtained from $\underline{c}^{k-1}$ by appending the cost concerning the k-th instance in $\underline{i}$.

- A *block* is n-variate random variable $(\underline{c}^k(\theta_1, \underline{i}_l), \underline{c}^k(\theta_2, \underline{i}_l), \ldots, \underline{c}^k(\theta_n, \underline{i}_l))$ that corresponds to the computational results on instance $\underline{i}_l$ for each configuration in the race at step k.



- Null hypothesis – all possible rankings of the candidates within each block are equally likely.

F-Race: Algorithm

The optimization problem is tackled by generating a sequence $\Theta_0 = \Theta \supseteq \Theta_1 \supseteq \Theta_2 \supseteq \ldots$

The step from a set Θ_{k-1} to Θ_k is realized as follows

1. At step k, a new instance $\underline{i}_k$ is considered; each candidate $\theta \in \Theta_{k-1}$ still in the race is executed on $\underline{i}_k$ and each observed cost $c(\theta, \underline{i}_k)$ is appended to its $\underline{c}^{k-1}(\theta, \underline{i})$.
2. An aggregate comparison of the arrays $\underline{c}^k(\theta, \underline{i})$ for all $\theta \in \Theta_{k-1}$ is carried out by a statistical test – non-parametric Friedman 2-way analysis of variance by ranks.

   The null hypothesis being that all possible rankings of the candidates within each block are equally likely.
3. If the null hypothesis is rejected, pairwise comparisons between the best candidate and each other one are carried out by means of the t-test. All candidates that result significantly worse than the best one are discarded.

   Otherwise, all candidates in Θ_{k-1} pass to Θ_k.

Classification of Control Techniques

The main criteria for classifying methods controlling the EA's strategy parameters are

- What component/parameter is changed – representation, evaluation function, variation operators, selection, replacement, etc.
- How is the change made
  - deterministic heuristic – the strategy parameter is modified in a fixed way without using any feedback from the search. Typically, a time-varying rule is used that is activated at predefined generations.
  - feedback-based heuristic – some form of feedback from the search is used to trigger the change of the strategy parameter and to specify the direction and magnitude of the change. The updating mechanism is externally supplied. Example is the covariance matrix adaptation in CMA-ES.
  - self-adaptive – based on the idea of the *evolution of evolution*. The parameters to be adapted are encoded in the chromosomes and are subject to crossover and mutation. Example is the self-adaptation of mutation parameters in Evolution Strategies.
- Which evidence is used to make the change – monitoring performance of operators, diversity of the population, etc.

What Component/Parameter is Changed: Representation

Variable representation scheme in **Delta coding GA** – the idea is to maintain a good balance between fast search and sustaining diversity. Based on multiple restarts.

- The first run is used to find an **interim solution**.
- Subsequent runs decode the genes as distances (**delta values**) from the last interim solution.
- Each restart forms a new hypercube with the interim solution as its origin.
- The search expands or contracts by altering the resolution of the delta values (by changing the number of coding bits).
- The restarts are triggered when the population converges (convergence measured by the Hamming distance).

Delta Coding GA

```
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
```

©Eiben, A.E. et al.: Parameter Control in Evolutionary Algorithms

What Component/Parameter is Changed: Mutation

Evolution strategies

- Real-valued search spaces,
- **1/5 success rule** – adaptive mutation step size control
  - The ratio of successful mutations to all mutations should be $1/5$.
  - Mutation operator m modifies all components of the object parameter x^t according to

$$x'^t = x^t + \mathbf{N}_0(\sigma^t)$$

while the σ^t variances are dynamically adjusted (all at the same time) according to

$$\sigma^{t+n} = \begin{cases} c_d \cdot \sigma^t & \text{, if } p_s^t < 1/5 \\ c_i \cdot \sigma^t & \text{, if } p_s^t > 1/5 \\ \sigma^t & \text{, if } p_s^t = 1/5 \end{cases}$$

where p_s^t is the frequency of successful mutations, measured over $10n$ trials and recommended values of the increase/decrease step sizes are $c_d = 0.82$ and $c_i = 1/c_d$.

Covariance Matrix Adaptation Evolution Strategy – the covariance matrix is adapted based on the current population and the past adaptation steps.

What Component/Parameter is Changed: Mutation

Self-adaptive mutation rate control – for GAs with binary representation.

- Extra bits for encoding the individual's own mutation rate p_m
- **Fixed learning rate** α – a probability of applying bitwise mutation to the encoded mutation rate.
- The mutation works as follows:
  1. Mutate the bits that encode p_m with mutation probability α
  2. Decode these bits to $p_m^{'}$
  3. Mutate the bits that encode the solution with mutation probability $p_m^{'}$

What Component/Parameter is Changed: Evaluation Function

Penalty function constraint-handling approach in EAs – the penalty fitness function of the following form

$$\psi(x) = f(x) + r_g \times \sum_{i=1}^{m+p} G_i(x)^2$$

- Ideally, the value(s) r_g is adapted based on the current stage of the search process.

Stepwise adaptation of weights mechanism (SAW) – the idea is that constraints that are not satisfied after a certain number of fitness evaluations (steps) are considered difficult and as such must be given a high weight.

- The best individual in the population is periodically checked and the weights of those constraints that it violates are raised.

Facets of SAW – the weights reflect the difficulty of constraints

- for the given algorithm,
- given problem instance in
- the given stage of the search.

What Component/Parameter is Changed: Crossover

Adaptive operator selection – EAs that uses multiple crossover operators simultaneously within the same generation.

- Credit assignment mechanism – associates to each operator a reward, modeling its impact on the progress of evolution.

  The credit of an operator is calculated based on the improvement of the fitness of the newborn offspring over some reference fitness value – that of the individual parents, of the current best or median individuals.

- Selection rule – determines the operator to be used at each time step, depending on the operator rewards.

  The selection rules attach a probability to each operator and use a roulette wheel-like process to select the operator to be applied, based on these probabilities.

- Exploration/exploitation tradeoff in the operator landscape must be ensured – typically, if the reward provides an instant feedback, modeling the immediate benefits of applying the operator, then the selection rule must ensure that operators with low current benefits can still be explored at a later stage of evolution.

What Component/Parameter is Changed: Davis's Adaptive Operator Fitness

Credit assignment

- k crossover operators are used.
- Each operator has its local delta value d_i – represents the strength of the operator measured by the advantage of the child with respect to the best individual in the population.
- The values d_i are updated after every use of operator i.

Selection rule

- Each crossover operator has its own crossover rate $p_c(i)$.
- The crossover rates are recalculated every K generations. The idea is to redistribute 15% of the probabilities biased by the accumulated operator strengths (d_i).
  1. d_i values are normalised to d_i^{norm} so that $\sum_{i=1}^{k} d_i^{norm} = 15$.
  2. The new value for each $p_c(i)$ is calculated according to

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

  Note: The crossover rates of all of the operators sums up to 1, thus the shift up in the crossover rate of one operator is at the cost of other operators.

What Component/Parameter is Changed: Extreme Value Based AOS

Credit assignment – The idea is that attention should be payed to extreme, rather than average, events. If the average fitness reward was considered then an operator bringing frequent small improvements would dominate over an operator bringing rare large improvements.

1. When operator o is applied on the individual x, the fitness of the offspring is evaluated and the current improvement
$$(f(o(x)) - f(x))_+$$
is calculated and added to the window (FIFO structure) of size W.
2. The operator reward $\hat{p}_t$ at time t is set to the maximal fitness improvement in this window
$$\hat{p}_t = argmax\{\delta(t_i), i = 1 \dots W\}$$
where $\delta(t_i)$ denotes the fitness improvement observed at time t.

What Component/Parameter is Changed: Extreme Value Based AOS

Let k denote the number of variation operators, $(s_{i,t})_{i=1,k}$ a probability vector and $\hat{p}_{i,t}$ an estimate of the current operator reward.

At each time t:

1. operator i is selected with probability $s_{i,t}$
2. the corresponding reward r_t is computed using the credit assignment at hand
3. the reward estimate $\hat{p}_{i,t}$ of the selected operator is updated after r_t, using an additive relaxation mechanism with learning rate α $(0 < \alpha = 1)$

$$\hat{p}_{i,t+1} = (1 - \alpha) \cdot \hat{p}_{i,t} + \alpha \cdot r_t$$

Probability matching – aims to making $s_{i,t}$ proportional to $\hat{p}_{i,t}$, while enforcing a minimal amount of *exploration*.

- Let p_{min} denote the minimal probability of selection of nay operator, then

$$s_{i,t+1} = p_{min} + (1 - k \cdot p_{min}) \frac{\hat{p}_{i,t+1}}{\sum_{j=1}^{k} \hat{p}_{j,t+1}}$$

- If some operator gets no reward (respectively the maximal reward) for some time, its expected reward will go very slowly to p_{min} (resp. $1 - k \cdot p_{min}$).

  Even irrelevant operators keep being selected.

What Component/Parameter is Changed: Extreme Value Based AOS

Adaptive Pursuit – follows a winner-take-all strategy, selecting at each time step the operator i_t^* with maximal reward, and accordingly increasing its selection probability:

1. $i^* = argmax\{\hat{p}_{i,t}, i = 1, \ldots, k\}$
2. $s_{i^*,t+1} = s_{i^*,t} + \beta(1 - (k-1)p_{min} - s_{i^*,t})$, $\beta > 0$
3. $s_{i,t+1} = s_{i,t} + \beta(p_{min} - s_{i,t})$, for $i \neq i^*$

where learning rate β controls the greediness of the winner-take-all strategy.

What Component/Parameter is Changed: Selection

Adapting the selection pressure based on the so-called **Boltzmann selection** mechanism that changes the selection pressure according to a predefined *cooling schedule*.

Inspiration taken from condensed matted physics, where a minimal energy level is sought by state transitions. Being in a state i the chance of accepting state j is

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

where E_i, E_j are the energy levels, K_b is the Boltzmann constant, and T is the temperature.

- The more is the solution j inferior to solution i the smaller is the probability of accepting the inferior solution j.
- The smaller is the temperature T the smaller is the probability of accepting inferior solutions.

Memetic algorithm – Boltzmann acceptance used in the local search part, with the temperature inversely related to the fitness diversity of the population:

- Highly diversified population – the temperature is low, so only fitter solutions found by local search are likely to be accepted.
- Converged population – the temperature is high, making it more likely that an inferior solution will be accepted. This way a diversity is reintroduced into the population.

Parameter-less Genetic Algorithm: Motivation

Motivation – to make the EA an algorithm that is robust, efficient and easy-to-use.

- Typically, the EAs require quite a bit of expertise in order to make them work well for a particular application.
- The user is not interested in tuning and fiddling the EA's parameters for each single application. He would be happy if he could get around somehow.

Parameter-less GA [Harik99] eliminates the following parameters when applying the algorithm to a particular problem:

- **population size**,
- **selection rate** s – the amount of bias towards better individuals; usually expressed by a ratio of sampling rates of individuals with the best and average fitness in the population.
- **crossover probability** p_c – the amount of mixing.

Parameter-less GA: Getting Rid of Selection Rate and Crossover Prob.

- **Schema** S – a template, which defines set of solutions from the search space with certain specific similarities. The schema consists of 0s, 1s and wildcard symbols * (any value).

  Schema properties – defining length, order, and fitness.

  Example: schema $S =$\{11*0*\} covers strings 11000, 11001, 11100, and 11101

- A **simplified growth ratio of schema** S **at generation** t, considering only $\phi(S,t)$ the effect of the selection operator on schema S at generation t and $\epsilon(S,t)$ the disruption factor on schema S due to the crossover operator is

$$\phi(S,t) \cdot [1 - \epsilon(S,t)]$$

  Under the conservative hypothesis that a schema is destroyed during the crossover we get

$$s(1 - p_c)$$

- **Schema theorem**: Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.

We just need to ensure that the GA will obey the schema theorem and the growth ratio of building blocks will be greater than 1.

- **Setting** $s = 4$ **and** $p_c = 0.5$ **gives a net growth ratio of 2.**

Standard GA: Population Sizing

Whether the building blocks will mix efficiently in a single optimal solution is now a matter of having an adequate population size

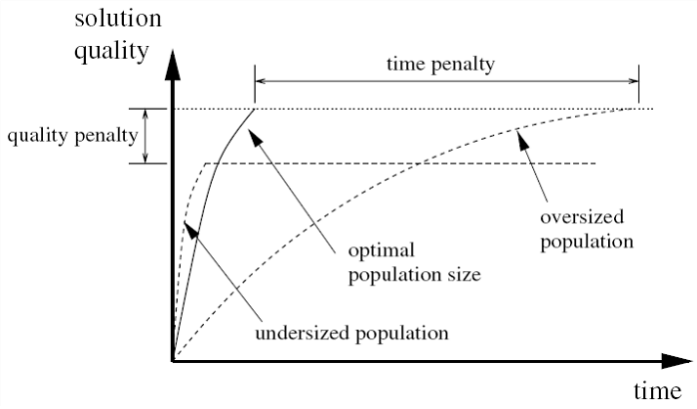
There are theoretical models for population sizing concluding that the **population size** should be **proportional to problem length and building blocks's signal-to-noise ratios**.

- For compact building blocks the required population size is reasonable.
- If the building blocks are not compact, then the population sizing requirements can be extremely large.

The models are difficult to apply in practice because they rely on parameters that are usually unknown and are hard to estimate for real world problems.

Effects of improperly set population size

- Too small population size $\Rightarrow$ quality penalty: The GA will converge to sub-optimal solutions.
- Too large population size $\Rightarrow$ time penalty: The GA will spend unnecessary computational resources.



Parameter-less GA: Getting Rid of Population Sizing

The idea is to let the algorithm do the experimentation with population sizes automatically by establishing a race among multiple populations of various sizes in a single GA's run:

- Each population $k > 1$ is twice as large as the population $k - 1$.
- The smaller populations are given more function evaluations, thus the different populations are at different stages of evolution.
- As time goes on, The smaller populations are eliminated and larger populations are created automatically based on observed average average fitness of the populations.

  If at any point in time, a larger population has an average fitness greater than that of a smaller population, then the smaller population is destroyed.

  - The rational for doing this is that in this situation it is very unlikely that the smaller population will produce a fitter individual than the larger one.
- The coordination of the array of populations is implemented with a counter of base m, which determines the proportion of fitness evaluations given to each of the simulated runs.

Parameter-less GA: Coordination of Populations

- At each generation, the counter of base $m = 4$ is incremented, and the position of the most significant digit changed during the increment operation is noted. That position indicates which population should be run.
- Since each population k is on the one hand half as large as the population $k+1$ and on the other hand is allowed 4 times more generations than population $k+1$ the population k is allowed to spend twice the number of fitness evaluations of population $k+1$.
- When some population converges or its average fitness is less than the average fitness of a larger population (due to a genetic drift – a population does not converges due to an insufficient selection pressure), it is removed together with all of the smaller populations.

  The counter is reset.

Counter base 4	Action
0	run 1 generation of population 1
1	run 1 generation of population 1
2	run 1 generation of population 1
3	run 1 generation of population 1
10	run 1 generation of population 2
11	run 1 generation of population 1
12	run 1 generation of population 1
13	run 1 generation of population 1
20	run 1 generation of population 2
21	run 1 generation of population 1
22	run 1 generation of population 1
23	run 1 generation of population 1
30	run 1 generation of population 2
31	run 1 generation of population 1
32	run 1 generation of population 1
33	run 1 generation of population 1
100	run 1 generation of population 3
101	run 1 generation of population 1
⋮	⋮

Recommended Reading

[Eiben07] Eiben, A.E., Michalewicz, Z., Schoenauer, M., Smith, J.: Parameter Control in Evolutionary Algorithms, In: Parameter Setting in Evolutionary Algorithms Studies in Computational Intelligence (54), Springer Verlag , pp. 19–46, 2007.

http://www.cs.vu.nl/ gusz/papers/2007-eib-mich-schoen-smit-chap.pdf

[Birattari02] Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A Racing Algorithm for Configuring Metaheuristics, In GECCO 2002, pp. 11–18, 2002

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.71.3223&rep=rep1&type=pdf

[Harik99] Harik, G.R., Lobo. F.G.: A Parameter-Less Genetic Algorithm, *IEEE Transactions on Evolutionary Computation*, pp. 523–528, 1999

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.5188