Non-Static Parameter Choices in Evolutionary Computation

Carola Doerr

CNRS and Université Pierre et Marie Curie, Paris, France

http://gecco-2017.sigevo.org/

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author. Copyright is held by the owner/author(s).
GECCO '17 Companion, July 15-19, 2017, Berlin, Germany
ACM 978-1-4503-4939-0/17/07.
http://dx.doi.org/10.1145/3067695.3067707

Link to the Latest Version, BibTeX Entry

- I will almost surely update these slides before the tutorial in July
- You can find the latest version of this tutorial online at http://www-ia.lip6.fr/~doerr/GECCO17tutorial.pdf
- You can cite this tutorial (thanks in advance ;)) as follows
  - Carola Doerr: Non-Static Parameter Choices in Evolutionary Computation. GECCO (Companion). ACM, 2017
  - BibTeX entry:

```
@inproceedings{Doerr17tutorial,
author    = {Carola Doerr},
title     = {Non-Static Parameter Choices in Evolutionary Computation},
booktitle = {Genetic and Evolutionary Computation Conference, {GECCO} 2017, Berlin, Germany, July 15-19, 2017, Companion Material Proceedings},
pages     = {tbd},
year      = {2017},
url       = {http://doi.acm.org/10.1145/3067695.3067707},
doi       = {10.1145/3067695.3067707}
publisher = {ACM}}
```


Instructor: Carola Doerr

- **Carola Doerr, née Winzen,** is a permanent researcher with the CNRS and Pierre et Marie Curie University (Paris 6).
- Carola's main research interest is in the *theory of randomized search heuristics*, both in the design of efficient algorithms as well as in lower bounds for such general-purpose optimization techniques. After contributing to the revival of black-box complexity, a theory-guided approach to explore the limitations of heuristic search algorithms, she recently started a series of works aimed at exploiting insights from the theory of evolutionary computation to design more efficient EAs, in particular such with a non-static choice of parameters.
- Carola has been co-chair of the theory track at GECCO 2015 and 2017 and served as tutorial chair at PPSN 2016. She is editor of two special issues in Algorithmica. From 2014 to 2016 she has been a co-organizer of the women@GECCO workshop series.
- She studied mathematics at Kiel University (Diploma in 2007) and computer science at the Max Planck Institute for Informatics and Saarland University (PhD in 2011). Her PhD studies were supported by a Google Europe Fellowship in Randomized Algorithms. Carola's thesis has been awarded the Otto Hahn Medal of the Max Planck Society. From Dec. 2007 to Nov. 2009, Carola Doerr has worked as a business consultant for McKinsey & Company. She was a post-doc at the Université 7 in Paris and the Max Planck Institute for Informatics in Saarbrücken.

What We Will Discuss Today

1. (Almost) all EAs depend on a number of parameters
2. Choosing the right parameters of an EA is an important task
3. Choosing the right parameters of an EA is a difficult task
4. Parameter Tuning vs. Parameter Control:
   why it is (almost) always better to use non-static parameters
5. Parameter Control Mechanisms
   1. Which parameters should be updated?
   2. Which effects should trigger an update?
   3. How should we update the parameters and how do we classify the different update schemes in EC?

A Word of Warning: Scope (1/2)

- The performance of an EA depends on
  - the components of the EA
  - the operators in use
  - the representation of the problem/ the model of the fitness function

  and the interactions among these!
- We won't discuss how to chose
  - the "best" algorithm for your problem
  - nor how to find a good fitness function to model your problem

  → we will take the algorithm(s) and problem(s) as given and ask ourselves how to find, for this given setting, good parameter values

A Word of Warning: Scope (2/2)

- In 110 minutes, I cannot offer you an exhaustive literature review. You can find very good surveys here (see reference list for details)
  - Eiben, Hinterding, Michalewicz, IEEE TEC, 1999 [EHM99]
  - Eiben, Michalewicz, Schoenauer, Smith, 2007 [EMSS07]
  - Karafotias, Hoogendoorn, Eiben, IEEE TEC, 2015 [KHE15]
  - Those of you who are interested in *runtime analysis works* on non-static parameter choices can take the survey in Section 6.1 in Doerr, Doerr: Optimal Static and Self-Adjusting Parameter Choices for the (1+(λ, λ)) Genetic Algorithm [DD17] as a starting point for further investigations

→ I have decided to discuss in-depth a few mechanisms that should give you a flavor of what has been done and what is possible (and how to get there!)

- We will mostly focus on discrete optimization
  - in continuous optimization, adaptive parameter choices are standard
  - similar mechanisms are used in continuous optimization, often (but not always) originating from a similar source of inspiration

Main Goal of this Tutorial

To inspire (and enable) you to use and to experiment with non-static parameter choices

Modus Operandi

- I have prepared a slide presentation but this should not stop you from starting discussions or commenting on the content of this tutorial → the more you get involved, the more we can learn from each other (and the more fun it is for all of us ☺)
- There will be a short learning control at the end of the tutorial
  → no worries, I won't collect exam sheets ;)
  → enables you to check if you have understood the main messages and ideas presented in this tutorial
- During the tutorial, you will be asked to share in your experience with non-static parameter choices
- Don't hesitate to ask questions during the tutorial !!!
  - if I am using a phrase that you don't know, it is likely that someone else in the room does not know it either
  - Same holds if I am unable to get my message across

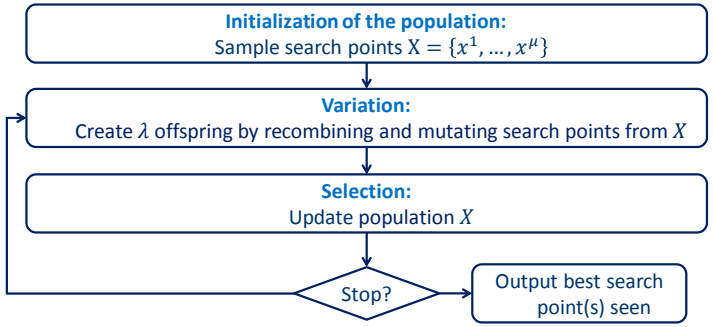
Feedback

- This is the first time that I am presenting this tutorial at GECCO
  - any feedback (positive and negative!) is highly welcome !
    → kindly take notes if you want to comment on or suggest something
    - which parts did you (not) like?
    - was the speed accurate?
    - is there anything that you would like to see changed?
  - I am also very interested to hear more about related literature. So if you know of any works that should be cited in this tutorial, please let me know!

Part 1: Choosing the Right Parameters is an Important but Difficult Task

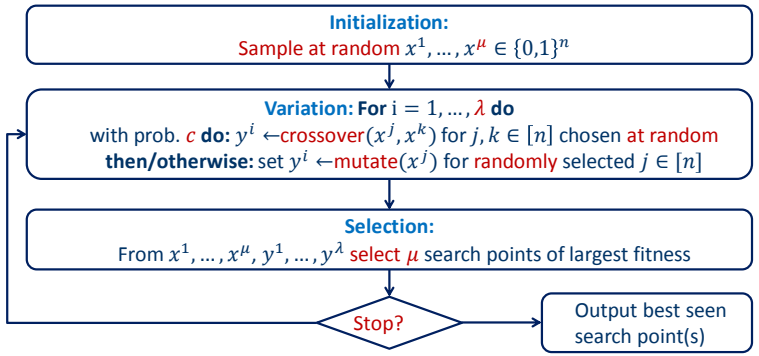
Simplified EA Blueprint

- To simplify our discussions, we will use the following blueprint to model evolutionary algorithms
- All content applies, nevertheless, also to more complex or other black-box algorithms



(Almost) All EAs are Parametrized

- Here is a "typical" evolutionary algorithm, a $(\mu + \lambda)$ EA with crossover



- There are quite a few parameters that need to be decided upon
- One of the most important questions in EC: how to choose these parameters???

Relevance of Good Parameter Choices

- The very early days of EC:
  "*EAs are robust problem solvers*"
  → no need to tune parameters!
- However, it was soon realized that this hope does not (and, in fact, cannot, as the "no free lunch" theorems tell us) materialize. It is today widely acknowledged that the parameter values have a decisive influence on the performance of an EA.
- Big open question (to date!): *How to find good parameter values?*

How to Find Good Parameter Values? (1/3)

- "Sports" of the 70s/80s in EC: Finding good parameter values
  - good = "globally good", i.e., for a broad range of problems
  - Examples: De Jong [DJ75], Grefenstette [Gre86] give recommendations for parameters such as population size, mutation and crossover probabilities, selection strategies, etc.
    → these recommendations are *independent* of problem class, problem size, … (*absolute values*)
  - Mühlenbein [Müh92] and others suggest $1/n$ as mutation rate for problems of lengths n (*relative values*)
    - Note: we know today that this choice indeed works well for a broad range of problems, cf. discussion below. However, it is widely acknowledged today, that problem size is not the only feature that matters.

How to Find Good Parameter Values? (2/3)

- "Modern view" of parameter selection: no globally optimal parameters exist
  → the right choice of parameter values crucially depends on the problem that we face and the algorithm that we employ
  - Result: whenever we face a new problem, or employ a new algorithm, we need to ask ourselves how to set the parameters
  - Very often, some (often many !) preliminary experiments are conducted to find reasonable parameter values
  - This *parameter tuning* quite often is a difficult task (see next slide)
  - In recent years, there is a substantial amount of EA literature on how to find good (static) parameter values, so-called parameter tuning mechanisms (see below for discussion and references)
  - Similar situation in the theory of EC: typical research question concerns the performance of a given algorithm with respect to some fixed set of parameters.
  - The bulk of EC papers (with a focus on discrete optimization problems) falls into this category of analyzing performance with respect to some fixed set of parameters! *(How about your latest GECCO paper?)*

Difficulty of Finding Good Parameter Choices

1. Even if we find "optimal" parameter values for one problem, these may (!, don't have to) be much different for similarly-looking problems (which is the basis for so-called *parameter choice by analogy*)
2. Small changes in one parameter can (!, don't have to) cause huge performance gaps
   - Many empirical works on this matter exist (again, check this year's GECCO talks to see if/how much effort has been put into finding the right parameters ☺)
   - Those of you interested in theoretical results can find in [DoerrJS+13] [Doerr, Jansen, Sudholt, Winzen, Zarges: Mutation Rate Matters Even When Optimizing Monotonic Functions. Evolutionary Computation, 2013] or [LS16] [Lengler, Steger: Drift analysis and evolutionary algorithms revisited, arXiv] examples where changing the mutation rate by a small constant factor changes the expected running time from a small polynomial (e.g., $O(n \log n)$) to super-polynomial/exponential

How to Find Good Parameter Values? (3/3)

- What I am trying to advocate with this tutorial:
  "New (?) modern view": the best parameter choices are non-static !

→ Instead of (or at least in addition to) parameter tuning, we should learn

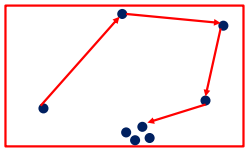
1. that the best parameter values depend on the current state of the optimization process and thus change over time
2. that there are some simple ways to enable our algorithms to figure out good parameter values themselves

Part 2: Parameter Tuning vs. Parameter Control: why it is (almost) always better to use non-static parameters

Why Use Non-Static Parameters?

VERY intuitive motivation:

1. Different stages of the optimization process require different parameter values!
   - Example:
     - beginning = "exploration phase"
       → large mutation rate/small selective pressure to make large jumps and discover different areas of the search space
     - end = "exploitation phase"
       → small mutation rates/high selective pressure to focus the search



2. No need for the user to identify good parameter values
   - Parameter tuning takes time and is quite complex (tuning 1 parameter is difficult already, but the operators also interact with each other, tuning 2 or more parameters typically requires non-sequential optimization, which is a difficult task)
   - Hope is that the algorithm identifies good parameter values itself

Your Experience with Non-Static Parameter Values

1. Have you already experimented with non-static parameter values?
   1. What was your motivation?
   2. How did you do this?
   3. What did go well, what did not go well?
2. If you haven't experimented with this idea, why not?
   (e.g., *never crossed my mind, not convinced by the motivation, no time to work on this, my algorithm doesn't have any parameters,…)*

Part 3:
Parameter Control – Introduction

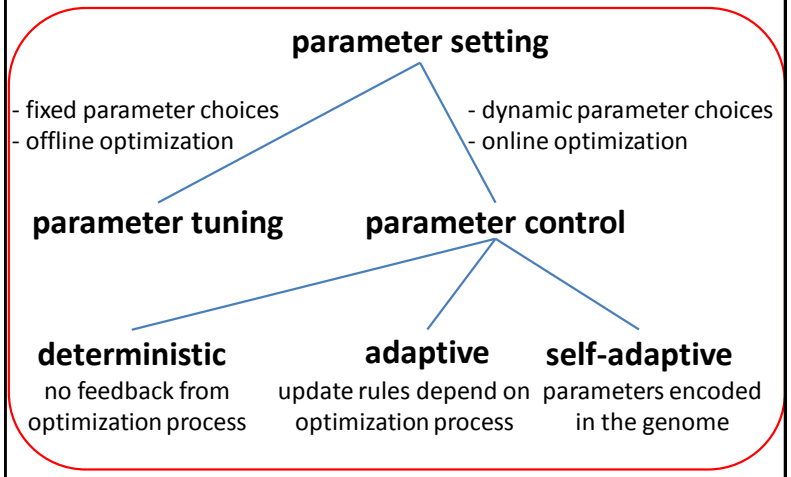
Main Questions in Parameter Control

1. *What* is adapted? (and who is affected: 1 individual vs. whole population)
   1. Population size
   2. Mutation rate, Crossover probability
   3. Selection pressure
   4. Fitness function (e.g., penalty terms for constraints)
   5. Representation
   6. …
2. What is the *basis/evidence for the update*?
   1. number of fitness evaluations performed or time elapsed
   2. progress, e.g., in terms of absolute or relative fitness values
   3. diversity measures
   4. …
3. *How* do we update the parameter(s):
   1. deterministic rule: time-dependent, feedback-free
   2. self-adaptive rule: treat parameter optimization as a complex optimization problem, use EAs to find good values
   3. adaptive: use feedback from the optimization process and plug it into some update mechanism determining the direction and/or the magnitude of change
   4. ….

Classification Scheme of [EHM99]

- Many attempts to find unifying taxonomy for parameter choices exist (cf. page 168 in [KHE15] for a survey)
- To date, the most popular classification scheme is that of Eiben, Hinterding, Michalewicz [EHM99] (we will discuss it in detail below)
- Important open problem:
  - to establish a more fine-grained classification scheme, based on a solid mathematical framework
    - → ideally, same terminology
      - for *discrete* and *continuous* optimization
      - in *theoretical* and *empirical* research streams
    - →it should reflect the main differences in the recent approaches on parameter control mechanisms

Classification Scheme of [EHM99]



Sketching the Classification Scheme of [EHM99] Parameter Tuning

- "Oldschool" (yes, I know that this is provocative and not 100% correct…) parameter selection: *parameter tuning*
  - → do some preliminary experiments to find good values for the parameters in question
    - Note: a lot of research exists on analyzing ways to tune parameters in a smart, efficient, and effective way. Albeit being a highly relevant topic, we won't touch this question here. The interested reader can find a survey on parameter tuning mechanisms for evolutionary algorithms in [ES11,Smi12]
  - → the bulk of EA literature falls into this category! (hopefully, this tutorial contributes to changing this ☺)
  - → Is the situation better in theoretical EC research?
    - Not so much better, see next slide

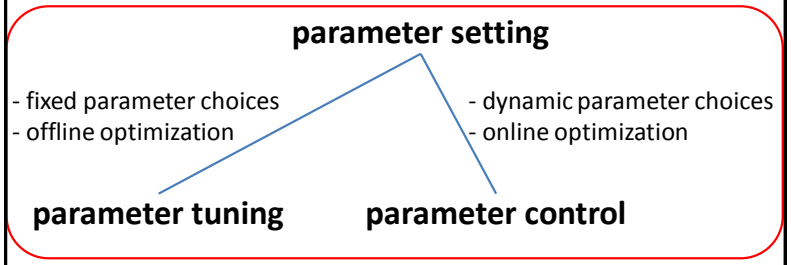
Sketching the Classification Scheme of [EHM99] Parameter Tuning

Similar situation in theoretical EA research: most research in the EA theory community is dedicated to EAs with static parameter settings

- Example: long series of work to find the optimal mutation rate for the (1+1) EA optimizing linear functions, (temporarily?) cumulating in the work of Witt [Wit13] confirming that the often-recommended choice of $1/n$ is indeed optimal among all static choices
- Quote from the abstract: "*As a consequence, the standard mutation probability $p = 1/n$ is optimal for all linear functions, and the (1+1) EA is found to be an optimal mutation-based algorithm.*" [Wit13]
- After this tutorial, you will agree that such statements have to be taken with care as they often (as in this and many similar cases) refer only to optimality among all static parameter choices !
- Main reasons for this situation:
  - ignorance of non-static parameter choices
  - difficulty of analyzing EAs with non-static parameters, cf. also [DDY16b] for difficulties of identifying optimal mutation strengths for the simple OneMax function

Sketching the Classification Scheme of [EHM99]

- First level of differentiation: discriminate between *parameter tuning* and *parameter control*



Sketching the Classification Scheme of [EHM99] Deterministic Parameter Control

Parameter Control Idea 1: Updates should follow a similar pattern

- Most popular example: exploration first, then exploitation
  - mutation rate: large in the beginning, smaller towards the end
  - selection strength: more generous in the beginning, higher selection pressure in the end
- to stimulate or enforce this behavior, we can change the parameters based on the time elapsed (number of generations, fitness evaluations, wall-clock time, etc.)
  - Simple examples:
    - cooling schedule of the selective pressure ("temperature") in Boltzmann selection of Simulated Annealing
    - start with mutation rate $p = 1/2$, decrease p after 10,000 fitness evaluations
    - after each 1,000 iterations, draw a random mutation probability

Sketching the Classification Scheme of [EHM99] Deterministic Parameter Control

Parameter Control Idea 1: Updates should follow a similar pattern

- Most popular example: exploration first, then exploitation
  - mutation rate: large in the beginning, smaller towards the end
  - selection strength: more generous in the beginning, higher selection pressure in the end
- to stimulate or enforce this behavior, we can change the parameters based on the time elapsed (number of generations, fitness evaluations, wall-clock time, etc.)
  - Simple examples:
    - cooling schedule of the selective pressure ("temperature") in Boltzmann selection of Simulated Annealing
    - start with mutation rate $p = 1/2$, decrease p after 10 evaluations
    - after each 1,000 [illegible] mutation probability

"Deterministic parameter choices"

Remarks on "Deterministic" Update Schemes

- Already Eiben, Hinterding, and Michalewicz noted in their work [EHM99] that the term "deterministic" is sub-optimal (update rules may be random as in the third example on the previous slide)
- More suitable terms could be
  - "*time-dependent*", "*scheduled*" update scheme, or
  - "*feedback-free*", "*progress-independent*" update scheme

  but in lack of a widely acknowledged alternative, "deterministic update rule" is still the predominantly used term
- The crucial feature here is that there is no feedback from the optimization process, so the update rule is determined in advance, before the actual run of the algorithm
  (Note: when random decisions are involved, then it would be possible to run the experiments determining the parameter value before the EA is started)
- Note that finding the optimal deterministic update rules requires tuning, i.e., while they bypass the disadvantage of the non-flexible static parameter values, they do not allow the algorithm to identify the good parameter values by itself

Examples for Deterministic Parameter Choices (1/2)

- Some selected theory works:
  - Hesser and Männer (PPSN'90) [HM90] suggested the following rule for the mutation strength of a GA with population size λ for OneMax:
    $p_m(t) := \frac{\sqrt{\frac{\alpha}{\beta}}\exp\left(-\frac{\gamma t}{2}\right)}{\lambda\sqrt{n}}$ where α, β, γ are constants
  - Jansen Wegener [JW06]: mutation rate changes in every iteration
    - $p_t(n) := 2^i/n$ where $i \equiv (t-1) \bmod \log(n) - 1$

    +/- very frequent changes → non-stable algorithm

    - worse performance on simple functions like OneMax, linear functions, LeadingOnes, etc.

    + examples where better performance than any static choice can be proven
  - Doerr, Doerr, Kötzing (GECCO 2016) [DDK16b]: in every iteration, a random step size is used for a multi-valued OneMax-type problem (problem will be discussed in more detail in the next section, along with a self-adjusting parameter choice)
    - Note: non-static parameter values, but static probability distribution used here!

Examples for Deterministic Parameter Choices (2/2)

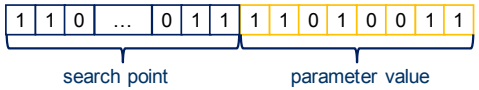
- Random Variation of the Population Size GA (RVPS) by Costa, Tavares, and Rosa [CTR99]
  - size of the actual population is changed every *N* fitness evaluations, for a given *N* (according to some monotonous rule)
  - Both shrinking and increasing the population size are considered
- Saw-tooth like population size growth considered by
  - Koumousis and Katsaras in [KK06] (TEC 2006): linear decrease of population size with eventual re-initialization of the population size by adding randomly selected individuals
  - Hu, Harding, Banzaf [HHB10]: inverse saw-tooth like population sizes

Sketching the Classification Scheme of [EHM99]
Self-Adjusting Parameter Control

- Parameter Control Idea 2:

	Finding good parameter values is difficult
+	EAs are good problem solvers
=	Use an EA to determine parameter values

- Many different ways to do this. Examples (sketched, much room for creativity here !):
  1. Create a new population of parameter values, choose from this parameter values, possibly apply variation to them, and employ them in your EA, select based on progress made
  2. append to the solution candidates a string which encodes the parameter value, first mutate the parameter value part, then use this parameter to change the search point, selection as usual



Sketching the Classification Scheme of [EHM99]
Self-Adjusting Parameter Control

- Parameter Control Idea 2:

	Finding good parameter values is difficult
+	EAs are good problem solvers
=	Use an EA to determine parameter values

"Self-adaptive parameter choices"

Examples for Self-Adaptive Parameter Choices

- We won't discuss this in much detail, but if you are interested in such mechanisms, you can start your investigations with the following works
  - Bäck PPSN'92 [Bäc92] and follow-up works: extends the chromosome by 20 bits. Mutation works as follows:
    1. Decoding the 20 bits to the individual's own mut. rate p_m
    2. Mutating the bits encoding p_m with mutation probability p_m
    3. Decoding these changed bits to p'_m
    4. Mutating the bits that encode the solution with mutation probability p'_m
  - Dang, Lehre (PPSN'16) [DL16]: theoretical work on a self-adaptive choice of the mutation strength in a non-elitist population

Sketching the Classification Scheme of [EHM99]
Adaptive Parameter Control

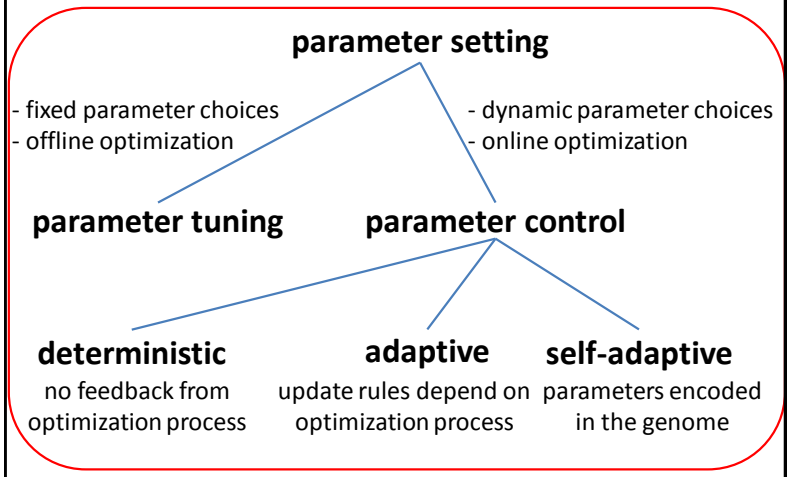
- Parameter Control Idea 3:
  - use feedback from the optimization process to change the parameters according to some pre-described rule
- Relevant feedback includes:
  - *success-based rules:* presence/absence of progress
    (Example: "if the iteration was successful, increase mutation rate, and decrease it otherwise")
  - *fitness-based update rules:* magnitude of parameter change depends on the magnitude of the progress or the fitness of the current-best search point(s)
    (Example: fitness- or ranking-based mutation rates)

"Adaptive parameter choices"

Sketching the Classification Scheme of [EHM99] Adaptive Parameter Control

- Parameter Control Idea 3:
  - use feedback from the optimization process to change the parameters according to some pre-described rule
- In my opinion, adaptive control mechanisms are the most promising direction for future work. The remainder of this tutorial therefore focusses on such update mechanisms
- Note: no formal definition exists to distinguish between self-adaptive and adaptive parameter control mechanisms. As a rule of thumb, self-adaptive mechanisms are those in which the parameter value is encoded in the genome and undergo variation and selection

Classification Scheme of [EHM99]



Discussion of the Classification Scheme

- The classification scheme of [EHM99] is the most widely accepted one
- While historically there has been quite some work on deterministic and self-adaptive update rules, today the most commonly applied rules are adaptive
- These adaptive rules can be much different in flavor, as we shall see below
- As mentioned earlier, after 18 years of existence, it may be the right time for a new classification scheme
- In [DD15a] (GECCO'15) we suggested to distinguish between
  - *functionally-dependent adaptive schemes:* parameters are in functional dependence of current population. That is, only the current state matters, not the process to achieve this state
  - *self-adjusting adaptive schemes:* parameters depend on success of previous iterations
  - Other discriminations are much needed (see discussions below) but haven't been established to date

Part 4: Examples for Parameter Control Mechanisms

Reminder: Main Questions in Parameter Control

1. *What* is adapted? (and who is affected: 1 individual vs. whole population)
   1. Population size
   2. Mutation rate, Crossover probability
   3. Selection pressure
   4. Fitness function (e.g., penalty terms for constraints)
   5. Representation
   6. …

   Focus of this tutorial
2. What is the *basis/evidence for the update*?
   1. number of fitness evaluations performed or time elapsed
   2. progress, e.g., in terms of absolute or relative fitness values
   3. diversity measures
   4. …
3. *How* do we update the parameter(s):
   1. deterministic rule: time-dependent, feedback-free
   2. self-adaptive rule: treat parameter optimization as a complex optimization problem, use EAs to find good values
   3. adaptive: use feedback from the optimization process and plug it into some update mechanism determining the direction and/or the magnitude of change
   4. ….

Part 4a: Simple Success-Based Update Rules

Main Message of this Section

- Non-static parameter update schemes don't have to be complicated!
- Quite simple ideas and mechanisms can do the job

Simple Success-Based Rules (1/2)

- Rechenberg's 1/5th success rule [Rec73]:
  - main idea: the optimal success rate of an ES should be around 1/5 (based on considerations wrt sphere function and corridor landscape)
  - If (observed success rate > 1/5) → increase mutation rate
    If (observed success rate < 1/5) → decrease mutation rate
  - similar rules have been proposed by Schumer, Steiglitz 68 [SS68] and Devroye [Dev72]
- Intuition:
  - when success is too likely to happen, we seem to be in an easy part of the optimization problem
    → let's try to make more progress per step
    → corresponds to making larger jumps, i.e., larger mutation rates
  - when success is happening too seldom, we could be approaching the optimum and should focus our search → smaller jumps are sought → decrease mutation rate

Simple Success-Based Rules (2/2)

- Rechenberg's 1/5th success rule:
  If (observed success rate > 1/5) → increase mutation rate
  If (observed success rate < 1/5) → decrease mutation rate
- Main question: *how* should we update the mutation rate?
- One quite simple idea:
  Simple success-based (multiplicative) parameter update mechanism:
  If (iteration successful) → multiply mutation rate by a constant C>1
  If (iteration not successful) → multiply mutation rate by a constant c<1
- Note 1: there is also justification to do this the other way around, i.e.,
  If (iteration successful) → multiply mutation rate by a constant c<1
  If (iteration not successful) → multiply mutation rate by a constant C>1
  (think of jump functions or other functions with a local optimum from which the algorithm needs to escape)
- Note 2: the same idea can also be used to control other parameters, such as the population size, crossover probabilities, etc.

Examples for Simple Success-Based Rules

- Simple success-based (multiplicative) parameter update mechanisms have been experimented with in different research streams of EC
- In what follows, I give a few examples, mostly stemming from the theory of EC
  → the simple algorithms and problems regarded there allow us to concentrate on the main ideas
  → for some of the considered ("toy") problems, it is sometimes possible to formally prove that the adaptive parameter choices outperform any (!) static one
  → while all the mentioned works serve as a showcase that self-adjustment is feasible and brings performance gains, for none of them has there been a thorough investigation of how much one can gain by tuning the adjustment rules (see discussion below), so there is a lot of room for us to experiment and to learn !
- Empirical works which can serve as a starting point for further investigations on the simple success-based (multiplicative) parameter update mechanisms include [Aug09,KMH+04]

Simple Success-Based Rules: Example 1

- Lässig, Sudholt: Adaptive Population Models for Offspring Populations and Parallel Evolutionary Algorithms, FOGA 2011 [LS11]:
  - regard the $(1+\lambda)$ EA
  - an iteration is called *successful* if it produces an offspring of better than previous best fitness value
  - **Scheme A:**
    - If (iteration not successful) → double λ
      If (iteration successful) → reduce λ to 1
  - **Scheme B:**
    - If (iteration not successful) → double λ
      If (iteration successful) → halve λ
  - Main results: decreased expected parallel optimization times without increasing the expected sequential runtime for problems like OneMax, LeadingOnes, Jump, unimodal functions

Simple Success-Based Rules: Example 1

- Similar mechanism has been proposed by Jansen, De Jong, Wegener ECJ 2005 [JDW05]:
  - **Scheme C:**
    - If (iteration not successful) → double λ
      If (iteration successful) → replace λ by λ/s where s is the nbr of better offspring
  - Jansen, De Jong, Wegener showed that this principle works well in practice, but did not analyze it theoretically

Simple Success-Based Rules: Example 2

- In [DDK16a] (PPSN 2016) we regard a multi-valued version of OneMax
  - Reminder: OneMax function
    - traditionally, OM is the counting-ones function $\mathrm{OM}(x) = |\{i \mid x_i = 1\}|$
    - generalization:
      - unknown target string $z \in \{0,1\}^n$
      - fitness $\mathrm{OM}_z(x) = |\{i \mid x_i = z_i\}| = n - H(x,z)$ = number of bits in which x and z agree.
        (For $z = (1,\dots,1)$, $\mathrm{OM}_z = \mathrm{OM}$ = counting-ones function)
    - Maximization of OM_z = find z = minimize the Hamming distance to z
  - Multi-valued version $z \in \{0,1,\dots,r-1\}^n$
    - $f_z(x) = \sum_{i=1,\dots,n} d(x_i, z_i)$ where $d(.,.)$ is some distance function, e.g., $d(a,b) = |b-a|$ (interval metric) or $d(a,b) = \min\{|b-a|, |b-a+r|, |b-a-r|\}$ (ring metric)
  - Algorithm: RLS-type algorithm with *component-wise step sizes* (blackboard, or see next slide)

Simple Success-Based Rules: Example 2

Algorithm 1: $\mathrm{RLS}_{a,b}$ with self-adjusting step sizes maximizing a function $f : [r]^n \to \mathbb{R}$

Initialization: Let $v \in [1, \lfloor r/4 \rfloor]^n$ uniformly at random;
Sample $x \in [0..r-1]^n$ uniformly at random and query $f(x)$;
Optimization: for $t = 1, 2, 3, \dots$ **do**
  Choose $i \le n$ uniformly at random;
  for $j = 1, \dots, n$ **do**
    if $j = i$ **then** with probability $1/2$ let $y_j \leftarrow x_j - \lfloor v_j \rfloor$ and let $y_j \leftarrow x_j + \lfloor v_j \rfloor$ otherwise;
    else $y_j \leftarrow x_j$;
  Query $f(y)$;
  if $f(y) < f(x)$ **then** $v_i \leftarrow \min\{av_i, \lfloor r/4 \rfloor\}$ **else** $v_i \leftarrow \max\{1, bv_i\}$;
  if $f(y) \le f(x)$ **then** $x \leftarrow y$;

- For suitable $a > 1$ and $b < 1$ (e.g., $a \in [1.7, 2]$ and $b \in [0.8, 0.9]$) this algorithm achieves an expected optimization time of $\Theta\big(n(\log n + \log r)\big)$, which is best possible among all (static and non-static) parameter choices
- We do not know if any static parameter choice can achieve this performance

Simple Success-Based Rules: Example 3

- The following example requires a bit of time
- I decided to invest this time because
  - I think that this algorithm is worth it
  - this is an example where we can formally prove that the simple success-based rule is better than any static parameter choice
  - I want to discuss with you how we came up with the main ideas
  - there are quite a few open questions, interesting for both empirically- and theory-oriented researchers
- References for this part:
  - [DDE13] (GECCO 2013) and [DDE15] (TCS 2015, journal version of [DDE13])
    suggested the (1+(λ,λ)) GA
  - [DD15b] (GECCO'15): tight bound for static parameter setting
  - [DD15a] (GECCO'15): analysis of self-adjusting mechanism
  - [Doe16] (GECCO'16): lower bound for 3-dimensional parameter space

The (1+(λ,λ)) GA

1. **Initialization:** Sample $x \in \{0,1\}^n$ u.a.r.
2. **Optimization: for** $t = 1,2,3,\dots$ **do**
3. **Mutation phase:**
4. Sample ℓ from $B(n,p)$;
5. **for** $i = 1,\dots,\lambda$ **do** Sample $x^{(i)} \leftarrow \mathrm{mut}_\ell(x)$;
6. Choose $x' \in \{x^{(1)},\dots,x^{(\lambda)}\}$ with $f(x') = \max\{f(x^{(1)}),\dots,f(x^{(\lambda)})\}$;
7. **Crossover phase:**
8. **for** $i = 1,\dots,\lambda$ **do** Sample $y^{(i)} \leftarrow \mathrm{cross}_c(x,x')$;
9. Choose $y \in \{y^{(1)},\dots,y^{(\lambda)}\}$ with $f(y) = \max\{f(y^{(1)}),\dots,f(y^{(\lambda)})\}$;
10. **Selection step: if** $f(y) \ge f(x)$ **then** replace x by y;

0	1	0	1	0	0	0	0	x
0	0	1	1	0	1	0	0	$x^{(i)}$

0	1	0	1	0	0	0	0	x
0	0	1	1	0	1	0	0	x'
0	1	1	1	0	0	0	0	$y^{(i)}$

The (1+(λ, λ)) GA

1. **Initialization:** Sample $x \in \{0,1\}^n$ u.a.r.
2. **Optimization: for** $t = 1,2,3,\dots$ **do**
3. **Mutation phase:**
4. Sample ℓ from $B(n, p)$;
5. **for** $i = 1, \dots, \lambda$ **do** Sample $x^{(i)} \leftarrow \text{mut}_\ell(x)$;
6. Choose $x' \in \{x^{(1)}, \dots, x^{(\lambda)}\}$ with $f(x') = \max\{f(x^{(1)}), \dots, f(x^{(\lambda)})\}$;
7. **Crossover phase:**
8. **for** $i = 1, \dots, \lambda$ **do** Sample $y^{(i)} \leftarrow \text{cross}_c(x, x')$;
9. Choose $y \in \{y^{(1)}, \dots, y^{(\lambda)}\}$ with $f(y) = \max\{f(y^{(1)}), \dots, f(y^{(\lambda)})\}$;
10. **Selection step: if** $f(y) \geq f(x)$ **then** replace x by y;

- Quite a few parameters that need to be chosen
- Analyzing the performance of the algorithm on OneMax, we observed that $p = \lambda/n$ and $c = 1/\lambda$ are good choices, reducing the 3-dimensional parameter space to a 1-dimensional one

How to Chose λ in the (1+(λ, λ)) GA?

- We analyzed the performance of the (1+(λ, λ)) GA on OneMax
- First "quick&dirty" result: for $\lambda = \Theta(\sqrt{\log n})$ the expected runtime of the (1+(λ, λ)) GA on OneMax is $O(n\sqrt{\log n})$ [DDE13]
- This bound has later been slightly improved in [DD15b]: for $\lambda = \Theta(\sqrt{\log(n) \log\log(n) / \log\log\log(n)})$ the expected runtime of the (1+(λ, λ)) GA on OneMax is $O(n\sqrt{\log(n) \log\log\log(n) / \log\log(n)})$
- No other (static!) combination of p, c, λ can yield a better runtime

How to Chose λ in the (1+(λ, λ)) GA?

- In [DDE13] we also observed (and this was in fact no so difficult to spot!) that a fitness-based choice of λ gives a better result:

  for $\lambda = \sqrt{\frac{n}{n-f(x)}}$, the runtime is $O(n)$

  - This linear runtime is better than what any (!) static parameter value can achieve (by the results presented in [Doe16])

  → one of the few examples where a non-static choice can be proven (with mathematical rigor) to outperform static parameter setting

  - Linear runtime can also be shown to be the best possible achievable runtime
- Disadvantage of this non-static, fitness-dependent choice: hard to guess such a functional relationship!
- Main question: is there a way to achieve similar performance in an automated way?

Self-Adjusting λ in the (1+(λ, λ)) GA (1/2)

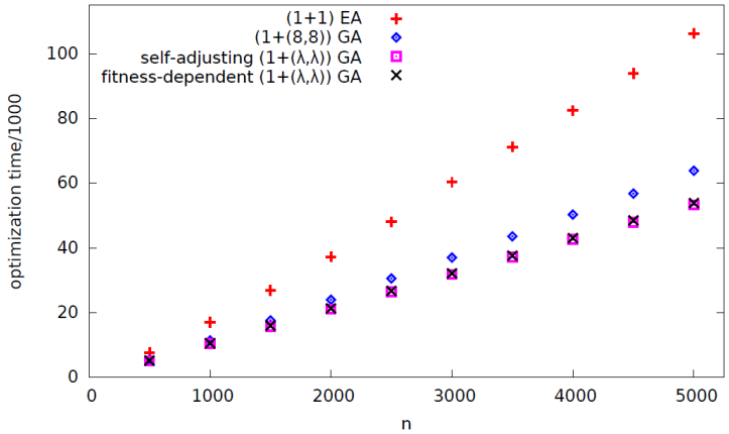
- Can the algorithm find good (or optimal) values for λ by itself?
- Idea: simple success-based rule.
  - If at the end of an iteration
    - we have an improvement ($f(y) > f(x)$) then $\lambda \leftarrow \lambda/F$;
    - No improvement ($f(y) \leq f(x)$) then $\lambda \leftarrow \lambda F^{1/4}$;
- We did not invent this update scheme. We took it from work by Auger (GECCO'09) [Aug09], who herself had taken it from work by Kern, Müller, Hansen, Büche, Ocenasek , Koumoutsakos, Natural Computing, 2004 [KMH+04]
- Why is this called "*discrete 1/5th success rule*"?
  - Assume that at time t you have a parameter value of x^t
  - We run the algorithm for s iterations
  - If $s/5$ of these iterations were successful, then $x^{t+s} = x^t$

  → Note tough that we do not enforce or induce a success rate of 1/5, nor is this sought

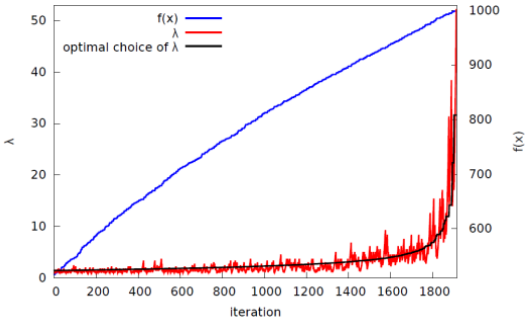
Self-Adjusting λ in the (1+(λ, λ)) GA (2/2)

- Can the algorithm find good (or optimal) values for λ by itself?
- Idea: simple success-based rule.
  - If at the end of an iteration
    - we have an improvement ($f(y) > f(x)$) then $\lambda \leftarrow \lambda/F$;
    - No improvement ($f(y) \leq f(x)$) then $\lambda \leftarrow \lambda F^{1/4}$;
- Why did we try this discrete 1/5th success rule?
  - By chance… We knew about it, we tried it, and it worked…
  - We actually did not (not yet…) experiment with this rule, and it is not unlikely that other update mechanisms yield even better performance. For the time being, we were happy with the results presented next ☺
  - Anyone in this room interested in investigating these choices further, please talk to me !

Experimental Results for Self-Adjusting (1+(λ, λ)) GA on OneMax



Self-Adjusting Choice Imitates the Optimal Fitness-Dependent Parameter Choice



- Plot shows one representative run of the self-adjusting (1+(λ,λ)) GA on ONEMAX for $n = 1{,}000$
- In [DD15a] we could prove, with mathematical rigor, that the simple success-based rule suggested above indeed yields linear (and thus asymptotically optimal !) performance

Good Performance Also for Other Test Functions

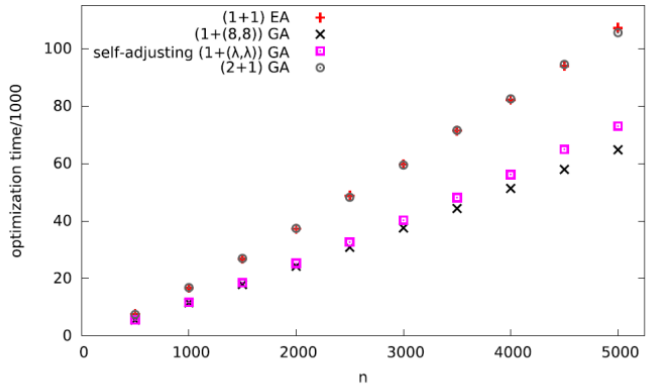
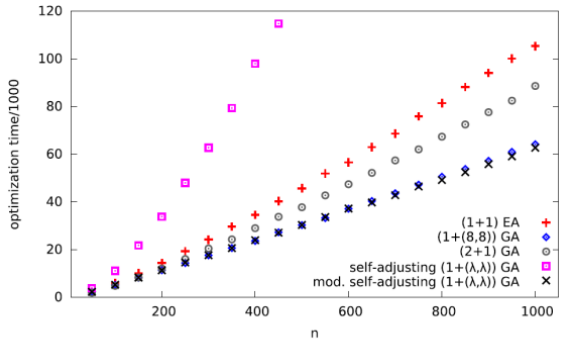


Fig. 6. Average optimization time of the $(1 + (\lambda, \lambda))$ GA on linear functions with random weights $w_i \in [1, 2]$.

- Performance on linear functions with random weights in [1,2]
- Average over $1{,}000$ runs

Good Performance Also for Other Test Functions

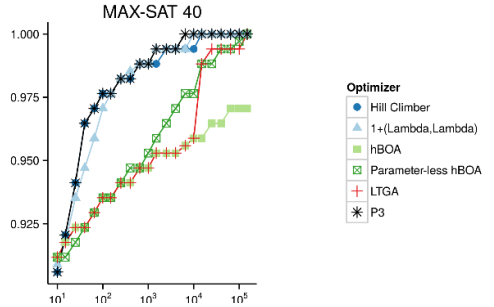


- Performance on royal road functions with block size 5

$$\mathrm{RR}_k(x) := |\{i \mid x_{(i-1)k+1} = \ldots = x_{\min\{ik,n\}} = 1\}|$$

- Average over 1,000 runs
- Modified self-adjusting parameter choice: no update if fitness does not change

Surprise: Performance on MaxSAT (and a number of other combinatorial problems)



- Graph taken from [GP15]: Goldman, Punch ECJ 2015. It shows number of satisfied clauses as a function of time for a MaxSAT instance (median values across 100 independent runs)
- First theoretical results for the self-adjusting (1+(λ,λ)) GA on MaxSAT available, see talk of Buzdalov and Doerr in the best paper session of the theory track here at GECCO 2017 [BD17]

Example 4: The $(1+\lambda)$ EA on OneMax

- Series of works analyzing how the mutation rate in the $(1+\lambda)$ EA, for fixed (!) λ, influences the expected number T^{gen} of generations (!) until, for OneMax, an optimum is evaluated for the first time
  - For static mutation rate $p = r/n$, Giessen and Witt (GECCO'15) [GW15] and [GW17] have shown that T^{gen} equals

$$(1 \pm o(1)) \left(\frac{1}{2} \cdot \frac{n \ln \ln \lambda}{\ln \lambda} + \frac{e^r}{r} \cdot \frac{n \ln n}{\lambda} \right)$$

  - This bound is minimized for $r = 1$ (i.e., $p = 1/n$)
    (note that in [GW16] (GECCO'16) they showed that even for moderate n and not too small λ mutation rates up to 10% larger than $1/n$ minimize the expected runtime)
  - For a fitness-dependent mutation rate, Badkobeh, Lehre, Sudholt (PPSN'14) [BLS14] showed a $T^{\mathrm{gen}} = \Theta\left(\frac{n}{\log \lambda} + \frac{n \log n}{\log \lambda}\right)$ runtime bound
    ☺ optimal among all possible and better than any static parameter setting
    ☹ requires the non-trivial setting $p = \max\left\{\frac{1}{n}, \frac{\ln \lambda}{n \ln\left(\frac{en}{d}\right)}\right\}$
    →main question is again how to achieve such a behavior without having to guess such a complicated relationship

Example 4: The $(1+\lambda)$ EA on OneMax

Doerr, Giessen, Witt, Yang (GECCO'17, theory track) [DGWY17] suggest the following mechanism:

- let p be the current mutation rate
  - in each iteration do:
    - create $\lambda/2$ offspring with mutation rate $2p$
    - create $\lambda/2$ offspring with mutation rate $p/2$
  - update p as follows (capping at $2/n$ and 1/4, respectively)
    - with probability 1/2 set it to the value for which the best offspring has been found
    - with probability 1/2, independently of the last iteration, randomly decide whether to replace p by either $p/2$ or by $2p$
  - Main result: this simple mechanism achieves the asymptotically optimal $T^{\mathrm{gen}} = \Theta\left(\frac{n}{\log \lambda} + \frac{n \log n}{\log \lambda}\right)$ performance
  - Note: this is very recent work. It is likely to be better to work with 3 subpopulations, created by mutation rate cp, p, and Cp for some constants $c < 1, C > 1$ (feel free to be creative ;))

Part 4b: Learning- or Reward-Type Mechanisms ("Operator Selection")

Main Ideas of Learning-Type Updates

- The main idea for learning-/reward-type adjustment rules is
  - have a set S of possible parameter values
  - according to some rule, test all or some of these values
  - update the likelihood to employ the tested value based on the feedback from the optimization process
- Picture to have in mind:
  - n experts
  - in each round, you have to chose one of them and you follow his advice
  - you update your confidence in this expert depending on the quality of his forecast
- **Main difficulty: exploitation vs. exploration trade off**
  - exploitation: we want to test each parameter value sufficiently often, to make sure that we select the "optimal" one (in particular when the quality of its "advice" changes, which is the typical situation that we face in evolutionary optimization)
  - exploration: we want, of course, to use an optimal parameter value as often as possible

Learning-Type Updates, Remarks

- Frequently found feature: *time-discounted methods*. That is, a good advice in the past is worth less than a good advice now
  - different update mechanisms and "forgetting rates" have been experimented with, see discussion below
  - note that such mechanisms are in particular useful when the quality of advice (in our setting, this could be the expected fitness gain, the expected decrease in distance to the optimum, or some other quantity) changes over time
- Note: such learning mechanisms are referred to as "*operator selection*" in [KHE15]. Another keywords to search for is "*credit assignment*". It may also be worth to look into literature from learning, in particular on *multi-armed bandit* algorithms (main goal: maximize reward "on the go", i.e., while learning) and on *reinforcement learning* (possibly have dedicated "learning" iterations, a notion of state is introduced and the hope is to learn for each state which operator maximizes expected progress)
- Again I will have to focus on a few selected works here. Much more work has been done, cf. Section IV.C.4 in [KHE15] for a survey. There is still much room for further creativity and much research is needed to understand which mechanisms are most useful in which situations

Example 1: Davis's adaptive operator fitness (1/2)

Davis (ICGA'89) [Dav89] suggests to adapt rates of crossover operators based on rewards

- Several crossover operators are used simultaneously in every iteration, each having its own crossover rate $p_c(\text{operator}_i)$
- the strength of an operator is measured by the fitness value d_i gained over the best so-far individual in the population. These values are updated after every use of operator i
- every K iterations, the crossover rates are being updated as follows:
  $$p_c^{\text{new}}(\text{operator}_i) = 0.85\, p_c^{\text{old}}(\text{operator}_i) + d_i^{\text{normalized}}$$
  with $d_i^{\text{normalized}}$ are normalized d_i values (summing up to 15)
  that is, 15% of the probability mass is re-allocated based on the experience from the last K iterations

Example 1: Davis's adaptive operator fitness (2/2)

- Julstrom (ICGA'95) [Jul95] revisited this mechanism and proposed the following changes:
  - simpler update mechanism
  - an operator is considered successful if its offspring is better than its parents, i.e., it does not necessarily have to be better than the current-best individual (*local reward*) or if it better than the median fitness of the individuals in the population
    - *local reward:* offspring better than parents
    - *global reward:* offspring better than current-best individual (used by Davis)
- Combinations of local and global rewards can also be considered, cf. work by Barbosa and e Sa [BeS00] and follow-up works

Example 2: COBRA

Cost Operator Based Rate Adaption (COBRA), suggested by Tuson and Ross (ECJ 1998) [TR98]

- Set of possible values for operator probabilities
- Operators are evaluated periodically, but information does not transfer to the next cycle, i.e., the rates are based only on the "productivity" of the operators in the last cycle
- "Productivity" = average fitness gain over parents during the time period divided by the cost of evaluating an offspring
- the rank of an operator determines the operator probability

Example 3: Dynamic Multi-Armed Bandits

- Thierens (GECCO'05) [Thi05] parametrizes the forgetting rate in a dynamic multi-armed bandit problem
  - Operator i is selected with some probability s_i^t
  - The update mechanism for the reward estimate of this i-th operator is
    $$p_i^t = (1-\alpha)\, p_i^t + \alpha r^t,$$
    where r^t is the reward in the t-th iteration
- ***Probability Matching* algorithms:**
  - s_i^t is proportional to p_i^t while maintaining a minimal amount of exploration (more precisely, $s_i^t \geq p_{\min}$ and $s_i^t - p_{\min}$ proportional to p_i^t)
- ***Adaptive Pursuit* algorithms:**
  - almost greedy selection, a small "*exploration* (aka "*learning*") *rate*" β controls how likely other than current-best operators are used
- **Dynamic Multi-Armed Bandits**: Da Costa, Fialho, Schoenauer, Sebag (GECCO'08) [DFSS08] and follow-up works suggest a parameter control mechanism that hybridizes a multi-armed bandit algorithm (Upper Confidence Bound *UCB*-type, see next slide) with the statistical Page-Hinkley test (which triggers a restart of the UCB mechanism if positive, indicating a change in the time series)

Comment on Multi-Armed Bandits

- **Upper Confidence Bound**, aka *UCB*-mechanisms are well known in learning theory, cf. work by Auer, Cesa-Bianchi, Fischer ML'02 [ACBF02]
- main ideas:
  - cUCB greedily selects the operator (the "arm") maximizing the following expression:
    $$\text{expected reward} + \sqrt{c \log \frac{\sum_k n_{k,t}}{n_{j,t}}},$$
    where
    - $n_{k,t}$ is the number of times the k-th arm has been pulled in the first t iterations and
    - c is a parameter that allows to control the exploration likelihood (vs. exploitation, which is controlled by the first summand)
  - tuned and other variants of this algorithm exist, cf. [ACBF02] for details and empirical evaluations
- These ideas can be used in operator selection, but note that in contrast to the classical setting in multi-armed bandit theory the rewards change over time (dynamic multi-armed bandit scenario)

Extreme Value-Based Adaptive Operator Selection (ExAOS)

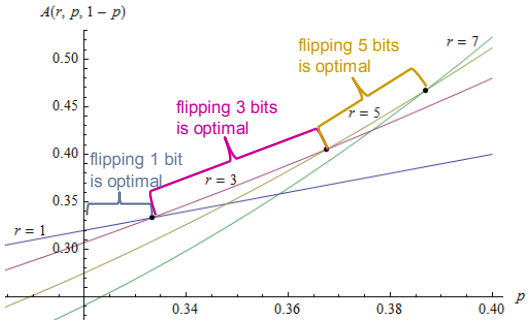
- In [FCSS08], Fialho, Da Costa, Schoenauer, and Sebag argue that, for many problems,
  - rare large fitness improvements are often better than
  - many small fitness improvements
- They suggest to distribute confidence based on the largest fitness improvement that an operator has produced in the last W iterations in which it has been used (sliding window of size W)
  - Sizing W is again non-obvious, too small W makes it difficult for an operator with rare but large fitness improvements to be chosen, while too large W makes it more difficult to adjust the search to the current state of the optimization process
- In [FCSS10] the authors suggest the following changes:
  - increase the reward with the time elapsed since the last application of the operator
  - decrease it with the number of times the operator has been used in the last iterations

Example 4: Self-Adjusting RLS on OneMax (1/4)

- An interesting (albeit not so easy to answer problem) is to determine, for a given search point x, how many random bits to flip in order to maximize the expected progress towards the target string z when $f = \mathrm{OM}_z$
- It is easy to convince oneself that the optimal number of bits that one should flip is large when $\mathrm{OM}_z(x)$ is small and is getting smaller when we approach the target string z (illustration on the blackboard)
- In [DDY16b] (GECCO'16) we analyzed this dependence and showed that an optimal mutation-based algorithm is the one employing such fitness-based step sizes, striving at any point in time for maximal drift towards the target string z
- As before, the question is how an algorithm designer should guess such a relationship (e.g., it turns out that the numbers should always be odd. It is not so easy to compute the cutoff-points from which on the optimal set size changes (see next slide), etc.)
- In [DDY16a] (PPSN'16) we showed how a learning-type mechanism automatically chooses parameter values that are close to optimal

Guessing the Optimal Mutation Strength is Non-Obvious

- Expected progress $A(r, p, 1-p)$ for different mutation strengths r=1,3,5,7 as a function of the distance pn to the target string
- As soon as the distance is less than $n/3$, it is optimal to flip 1 bit
- There is a complex monotonic relationship between distance and optimal mutation strength



Example 4: Self-Adjusting RLS on OneMax (2/4)

- Main idea: estimate the performance of different parameter values. Greedily choose the one which has the highest confidence score
  - Fix a small number of possible mutation strengths $[r] := \{1, 2, \dots, r\}$
  - Estimate the expected fitness gain $v_{t-1}[k]$ from using k-bit flips (using data from the past, see next slide)
  - In iteration t
    - with probability δ, use a random $k \in [r]$ "exploring mut. strengths"
    - with prob. $1-\delta$, use a k that maximized $v_{t-1}[k]$ "take the most efficient k"
  - Update the expected fitness gain estimations
- Result: RLS optimizing OneMax with this self-adjusting mutation strength in almost all iterations uses the (in this situation) optimal mutation strength.
- The iterations that do not operate with the optimal mutation rate account for an additive $o(n)$ contribution to the total runtime and are thus negligible
- This adaptive mechanism is provably faster than all static unbiased mutation operators!
- This algorithm with the same budget computes a solution that asymptotically is 13% closer to the optimum than RLS (given that the budget is at least $0.2675n$).

Example 4: Self-Adjusting RLS on OneMax (3/4)

- Expected fitness gain estimation for using a k-bit flip:
$$v_t[k] := \frac{\sum_{s=1}^{t} 1_{r_s=k} (1-\varepsilon)^{t-s} \big(f(x_s) - f(x_{s-1})\big)}{\sum_{s=1}^{t} 1_{r_s=k} (1-\varepsilon)^{t-s}}$$
- $1/\varepsilon$: "*forgetting rate*", determines the decrease of the importance of older information. $1/\varepsilon$ is (roughly) the information half-life
- The "velocity" can be computed iteratively in constant time by introducing a new parameter $w_t[r] := \sum_{s=1}^{t} \mathbf{1}_{r_s=r} (1-\epsilon)^{t-s}$
- This mechanism seems to work well also for other problems
  - So far, no other theoretical results available
  - A few experimental results for LeadingOnes and the Minimum Spanning Tree problem exist, see next 2 slides (these results were also presented in [DDY16a])
  - Again, much more work is needed to see how the algorithm performs on other problems and how to set the parameters δ and ε (see also discussion below)

Example 4: Self-Adjusting RLS on LeadingOnes

LeadingOnes

Theoretical runtime of other evolutionary algorithms:

- Classic RLS: $0.5n^2$
- (1+1) EA with mutation rate $p = 1/n$: $0.8591n^2$
- (1+1) EA with best-possible mutation rate: $0.6796n^2$ [2]

Experimental results (100 runs) :

- $n = 10{,}000$; $r_{\max} = 5$; $\delta = 0.1$; $\varepsilon = 1/5{,}000{,}000$

	Average complexity	Relative standard deviation
RLS	$0.500n^2$	1.74%
Our algorithm	$0.450n^2$	4.36%

- LeadingOnes(x)=number of initial 1s, e.g., LO(1110****)=3
- parameters above required some tuning, bit we did not invest much time for the tuning → it is likely that you can get better results by a more careful investigation

Example 4: Self-Adjusting RLS on MST

Minimum Spanning Trees

Experimental results for two settings of graph (100 runs on each setting) :

- Setting 1(S1): $|V| = 20$; $|E| = 190$; $r_{\max} = 5$; $\delta = 0.1$; $\varepsilon = 1/400$
- Setting 2(S2): $|V| = 50$; $|E| = 1225$; $r_{\max} = 5$; $\delta = 0.1$; $\varepsilon = 1/20{,}000$

	Average runtime	Relative standard deviation
RLS (S1)	$9.62 \cdot 10^3$	47.34%
(1+1) EA (S1)	$26.22 \cdot 10^3$	45.61%
Our algorithm (S1)	$6.25 \cdot 10^3$	52.01%
RLS (S2)	$5.08 \cdot 10^6$	37.75%
(1+1) EA (S2)	–	–
Our algorithm (S2)	$2.70 \cdot 10^6$	36.34%

Example 4: Self-Adjusting RLS on OneMax (4/4)

- As said, we did not try hard to optimize the parameters δ and ε
- If you want to experiment with this learning idea, we suggest that you use the following set-up for the first tries:
  - few different values for the mutation strength (i.e., small r), since the learning effort is proportional to their number (we used $r = 5$)
  - **learning rate δ**: a small constant, e.g., 5% ("price of the learning mechanism")
    - $\delta\left(1 - \frac{1}{r}\right)$ is the rate of iterations using a non-optimal mutation strength (can still give progress, but smaller than best-possible)
    - we used $\delta = 0.1$ and this seems to work well
- **forgetting time $1/\varepsilon$:** this parameter is the most difficult one to set. We recommend to set it so that $1/\varepsilon$ is a small percentage of the envisaged total runtime, e.g., 1% → it takes very roughly that long to change to a new optimal parameter value
  → Too large ε: we quickly forget the outcomes of previous iterations
  ☺ quick adaption to a changed environment
  ☹ risk that a rare exceptional success with a non-ideal r-value has too much influence

Part 4c:
Ageing and Other Mechanisms

Other Control Mechanisms (1/3)

In addition to the simple multiplicative update rules and the learning-type rules, many other mechanisms have been experimented with. Here are a few keywords and references (Again, more or less random selection of references, much more work can be found in the survey papers. The works below can serve as a starting point for further investigations.)

- Krasnogor and Smith [KS00] (GECCO 2000) suggest a control mechanism for the selective pressure of a memetic algorithm. They use *Boltzmann selection* (popular selection mechanism used in Simulated Annealing, probability of 1 to accept better offspring, probability to accept worse offspring depends on the fitness difference of parent and offspring and a "temperature" which decreases over time, making it less and less likely for worse offspring to get accepted) and suggest to
  - increase selective pressure when fitness diversity in the population is large
  - decrease it when fitness diversity is low
  - main idea: low fitness diversity = converged population, increase probability to escape and to search elsewhere

Other Control Mechanisms (2/3)

- Controlling population size is the focus of the **Genetic Algorithm with Variable Population Size (GAVaPS)** by Arabas, Michalewicz, Mulawka (CEC'94) [AMM94]
  - individuals come with their own lifetime
  - at birth their age is set to 0, each iteration increases the age by 1
  - maximum lifetime depends on the fitness values, the better a new individual is, the longer its lifetime (and, hence, the more offspring are created from this individual)
  - there is hence no fixed population size, but the size depends adaptively on the search history.
  - One of the goals of GAVaPS was to remove the population size as parameter, but the update mechanism itself comes again with its own parameters
- **Adaptive Population GA (APGA)** by Bäck, Eiben, van der Vaart (PPSN 2000) [BEvdV00]:
  - similar to GAVaPS, but age of best individual is not increased, thus allowing it a longer life
  - lifetime depends on individual's fitness and current-best as well as average fitness of the individuals in the population

Other Control Mechanisms (3/3)

- On-the-fly population size adjustment by Eiben, Marchiori, and Valko (PPSN'04) [EMV04]: **Population Resizing on Fitness Improvement GA (PRoFIGA)**:
  - variable population size:
    1. fitness improvements → population size increases
       (update is proportional to fitness improvement and number of fitness evaluations remaining until maximum is hit)
    2. short-term lack of fitness improvement → population size decreases
       (multiplicative update, e.g., decrease by 5%)
    3. long-term lack of fitness improvement → population size increases
       (update as in 1 tough in principle a different rule could be applied)

"Parameter-less" Population Pyramid (P3)

- The following 2 examples do not fall into category of parameter control mechanisms but since it is much related, I want to briefly mention them
- **Parameter-less Population Pyramid (P3)** by Goldman and Punch (GECCO 2014) and (ECJ 2015) [GP14,GP15]
  - instead of generations, P3 works with a pyramid-like structure of populations
  - P3 combines local search with model-based search
  - The pyramid is constructed from scratch as follows:
    - In every iteration, a new random solution is generated, brought to a local optimum, and, if not in the pyramid already, this local optimum is added to the lowest population P_0
    - Solutions are then improved by crossover with individuals on higher pyramid levels. If a better offspring is found, it is added to level $i+1$ of the pyramid, where i is the level of the better of the two parents
  - P3 shows promising performance on several combinatorial problems. First theoretical results are available in [GS16] (Goldman, Sudholt GECCO 2016)

"Parameter-Less" GA

- **Parameter-less Genetic Algorithm (PLGA)** by Harik and Lobo (GECCO 1999) [HL99] and follow-up works
  - a number of populations of different sizes evolve simultaneously
  - the smaller the population size, the more function evaluations it gets
  - a populations becomes extinct when it converges
  - Hope was to remove population size as a parameter, but note that the mechanism itself introduces new parameters, so the term "parameter-less" may be deceptive

Part 6: Controlling Multiple Parameters

Controlling Multiple Parameters

- Most EAs have several parameters
- Intuitively, there is no reason to not control more than one or even all of them
- Several works on controlling more than 1 parameter exists, but we won't have the time to discuss them today
- Check the mentioned surveys for references on ideas that have been experimented with so far

Part 6: Wrap Up

Learning Control

1. What are the main (dis-)advantages of static parameter choices?
2. What are the main (dis-)advantages of non-static parameter choices?
3. How do we distinguish parameter control mechanisms?
4. What type of parameter control mechanisms have we discussed in this tutorial? (and which one do you want to try next?!)
5. Homework ☺
   1. How do non-static parameter choices perform on your favorite optimization problem?
   2. Which update mechanisms work well for your favorite EA?

Summary Static vs. Non-Static Parameter Choices (1/2)

- Clearly exaggerating, one can summarize our main messages as follows:
- Disadvantages of static parameter choices (aka *parameter tuning*):
  - ☹ takes a considerable amount of time
  - ☹ highly complex, multi-dimensional problem: optimal parameters can typically not be found in a sequential fashion (unfortunately still the predominant way of parameter tuning), because of the complex interactions between them
  - ☹ good parameter values for one problem can perform poorly on similarly-looking problems
  - ☹ good parameter values for one algorithm can cause poor performance for similarly-looking algorithm
  - ☹ even "optimal" static parameters can be inferior to dynamic ones as they do not adapt the parameter values to the optimization process
- Possible advantages:
  - no need to worry about suitable update rules

Summary Static vs. Non-Static Parameter Choices (2/2)

- Advantages of non-static parameter choices (aka *parameter control*):
  - ☺ we gain flexibility and the possibility to adjust the parameter values to the current state of the search
  - ☺ If we have no idea how to set the parameter, we let the algorithm discover itself
- Possible disadvantages:
  - how to determine which update scheme to use? → designing parameter control mechanisms can, in principle, be an even more complex task than parameter tuning
    (suggestion: use the "mushroom rule": have a set of 2 or 3 different mechanisms that you declare your favorite ones. Do not try to know all possible mechanisms but rather concentrate on the most promising ones, e.g., one multiplicative update rule, one learning-based rule)
  - update mechanisms often come with their own parameters (remember: hope is that the algorithm is much less sensitive to these)
  - possibly more difficult to understand how the update mechanism influences the overall performance (measured, e.g., by the distribution of the optimization time)

Wrap Up

- My hope was
  To inspire (and to enable) you to test parameter control mechanisms
- So, I hope that you are (now) convinced
  that non-static parameter values should be the new standard in the field
- As mentioned in the tutorial, there is a lot to be done to make this change happen. So let us set out and discover this largely unexplored field
  - enjoy! ☺
  - don't get frightened by the fact that quite some work has been done already. There is still much room for creativity and we are just starting to understand how good mechanisms look like !
  - … and, last but not least, keep in touch ☺

→ If you get to work on parameter control, I would be very much interested in your results, positive and negative!

Carola.Doerr@mpi-inf.mpg.de

Feedback

- This is the first time that I am presenting this tutorial at GECCO
  - any feedback (positive and negative!) is highly welcome!
    - which parts did you (not) like?
    - was the speed accurate?
    - is there anything that you would like to see changed?

Acknowledgments

- This work was supported by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH,LabEx LMH, in a joint call with Programme Gaspard Monge en Optimisation et Recherche Operationnelle
- This tutorial is also based upon work from COST Action CA15140 `Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO)' supported by COST (European Cooperation in Science and Technology).

cost
EUROPEAN COOPERATION
IN SCIENCE AND TECHNOLOGY

- Thanks also to the organizers and participants of Dagstuhl seminar "Automated Algorithm Selection and Configuration" (16412) for several very insightful discussions on parameter control mechanisms

References

References

[ACBF02] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

[AMM94] Jaroslaw Arabas, Zbigniew Michalewicz, and Jan J. Mulawka. GAVaPS - A genetic algorithm with varying population size. In *Proc. of International Conference on Evolutionary Computation (ICEC'94)*, pages 73–78. IEEE, 1994.

[Aug09] Anne Auger. Benchmarking the (1+1) evolution strategy with one-fifth success rule on the BBOB-2009 function testbed. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'09), (Companion)*, pages 2447–2452. ACM, 2009.

[Bäc92] Thomas Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In *Proc. of Parallel Problem Solving from Nature (PPSN'92)*, pages 87–96. Elsevier, 1992.

[BD17] Maxim Buzdalov and Benjamin Doerr. Runtime analysis of the $(1 + (\lambda, \lambda))$ Genetic Algorithm on random satisfiable 3-CNF formulas. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17)*, page tbd. ACM, 2017.

[BDN10] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Proc. of Parallel Problem Solving from Nature (PPSN'10)*, volume 6238 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2010.

[BeS00] Helio J.C. Barbosa and Asla Medeiros e Sá. On adaptive operator probabilities in real coded genetic algorithms. In *Proc. of Conference of the Chilean Computer Science Society*, 2000.

[BEvdV00] Thomas Bäck, A. E. Eiben, and Nikolai A. L. van der Vaart. An empirical study on gas "without parameters". In *Proc. of Parallel Problem Solving from Nature (PPSN'00)*, volume 1917 of *Lecture Notes in Computer Science*, pages 315–324. Springer, 2000.

[BLS14] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. Unbiased black-box complexity of parallel search. In *Proc. of Parallel Problem Solving from Nature (PPSN'14)*, volume 8672 of *Lecture Notes in Computer Science*, pages 892–901. Springer, 2014.

[CFSS08] Luís Da Costa, Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. Adaptive operator selection with dynamic multi-armed bandits. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'08)*, pages 913–920. ACM, 2008.

[CTR99] Carlos J. Costa, R. Tavares, and A. Rosa. An experimental study on dynamic random variation of population size. In *Proc. of Systems, Man, and Cybernetics (SMC'99)*, pages 607–612. IEEE, 1999.

References

[Dav89] Lawrence Davis. Adapting operator probabilities in genetic algorithms. In *Proc. of International Conference on Genetic Algorithms (ICGA'89)*, pages 61–69. Morgan Kaufmann, 1989.

[DD15a] Benjamin Doerr and Carola Doerr. Optimal parameter choices through self-adjustment: Applying the 1/5-th rule in discrete settings. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'15)*, pages 1335–1342. ACM, 2015.

[DD15b] Benjamin Doerr and Carola Doerr. A tight runtime analysis of the $(1+(\lambda, \lambda))$ genetic algorithm on OneMax. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'15)*, pages 1423–1430. ACM, 2015.

[DD17] Benjamin Doerr and Carola Doerr. Optimal static and self-adjusting parameter choices for the $(1 + (\lambda, \lambda))$ genetic algorithm. *Algorithmica*, 2017. Accepted subject to a minor revision. Available upon request.

[DDE13] Benjamin Doerr, Carola Doerr, and Franziska Ebel. Lessons from the black-box: Fast crossover-based genetic algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'13)*, pages 781–788. ACM, 2013.

[DDE15] Benjamin Doerr, Carola Doerr, and Franziska Ebel. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567:87–104, 2015.

[DDK16a] Benjamin Doerr, Carola Doerr, and Timo Kötzing. Provably optimal self-adjusting step sizes for multi-valued decision variables. In *Proc. of Parallel Problem Solving from Nature (PPSN'16)*, volume 9921 of *Lecture Notes in Computer Science*, pages 782–791. Springer, 2016.

[DDK16b] Benjamin Doerr, Carola Doerr, and Timo Kötzing. The right mutation strength for multi-valued decision variables. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'16)*, pages 1115–1122. ACM, 2016. Full version available at http://arxiv.org/abs/1604.03277.

[DDY16a] Benjamin Doerr, Carola Doerr, and Jing Yang. k-bit mutation with self-adjusting k outperforms standard bit mutation. In *Proc. of Parallel Problem Solving from Nature (PPSN'16)*, volume 9921 of *Lecture Notes in Computer Science*, pages 824–834. Springer, 2016.

[DDY16b] Benjamin Doerr, Carola Doerr, and Jing Yang. Optimal parameter choices via precise black-box analysis. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'16)*, pages 1123–1130. ACM, 2016.

[Dev72] Luc Devroye. *The compound random search*. Ph.D. dissertation, Purdue Univ., West Lafayette, IN, 1972.

[DGWY17] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. The $(1 + \lambda)$ evolutionary algorithm with self-adjusting mutation rate. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17)*, page tbd. ACM, 2017.

References

[DJ75] Kenneth Alan De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[DJS+13] Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges. Mutation rate matters even when optimizing monotonic functions. *Evolutionary Computation*, 21:1–27, 2013.

[DL16] Duc-Cuong Dang and Per Kristian Lehre. Self-adaptation of mutation rates in non-elitist populations. In *Proc. of Parallel Problem Solving from Nature (PPSN'16)*, volume 9921 of *Lecture Notes in Computer Science*, pages 803–813. Springer, 2016.

[Doe16] Benjamin Doerr. Optimal parameter settings for the $(1 + (\lambda, \lambda))$ genetic algorithm. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'16)*, pages 1107–1114. ACM, 2016.

[EHM99] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3:124–141, 1999.

[EMSS07] A. E. Eiben, Zbigniew Michalewicz, Marc Schoenauer, and James E. Smith. Parameter control in evolutionary algorithms. In *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 19–46. Springer, 2007.

[EMV04] A. E. Eiben, Elena Marchiori, and V. A. Valkó. Evolutionary algorithms with on-the-fly population size adjustment. In *Proc. of Parallel Problem Solving from Nature (PPSN'04)*, volume 3242 of *Lecture Notes in Computer Science*, pages 41–50. Springer, 2004.

[ES11] A. E. Eiben and Selmar K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1:19–31, 2011.

[FCSS08] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. Extreme value based adaptive operator selection. In *Proc. of Parallel Problem Solving from Nature (PPSN'08)*, volume 5199 of *Lecture Notes in Computer Science*, pages 175–184. Springer, 2008.

[FCSS10] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Ann. Math. Artif. Intell.*, 60:25–64, 2010.

[GP14] Brian W. Goldman and William F. Punch. Parameter-less population pyramid. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'14)*, pages 785–792. ACM, 2014.

References

[GP15] Brian W. Goldman and William F. Punch. Fast and efficient black box optimization using the parameter-less population pyramid. *Evolutionary Computation*, 23:451–479, 2015.

[Gre86] John J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans. Systems, Man, and Cybernetics*, 16:122–128, 1986.

[GS16] Brian W. Goldman and Dirk Sudholt. Runtime analysis for the parameter-less population pyramid. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'160*, pages 669–676. ACM, 2016.

[GW15] Christian Gießen and Carsten Witt. Population size vs. mutation strength for the $(1 + \lambda)$ EA on OneMax. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'15)*, pages 1439–1446. ACM, 2015.

[GW16] Christian Gießen and Carsten Witt. Optimal mutation rates for the $(1+\lambda)$ EA on OneMax. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'16)*, pages 1147–1154. ACM, 2016.

[GW17] Christian Gießen and Carsten Witt. The interplay of population size and mutation probability in the $(1+\lambda)$ EA on OneMax. *Algorithmica*, 78(2):587–609, 2017.

[HHB10] Ting Hu, Simon Harding, and Wolfgang Banzhaf. Variable population size and evolution acceleration: a case study with a parallel evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11:205–225, 2010.

[HL99] Georges R. Harik and Fernando G. Lobo. A parameter-less genetic algorithm. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 258–265. ACM, 1999.

[HM90] Jürgen Hesser and Reinhard Männer. Towards an optimal mutation probability for genetic algorithms. In *Proc. of Parallel Problem Solving from Nature (PPSN'90)*, volume 496 of *Lecture Notes in Computer Science*, pages 23–32. Springer, 1990.

[JDW05] Thomas Jansen, Kenneth A. De Jong, and Ingo Wegener. On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13:413–440, 2005.

[Jul95] Bryant A. Julstrom. What have you done for me lately? adapting operator probabilities in a steady-state genetic algorithm. In *Proc. of International Conference on Genetic Algorithms (ICGA'95)*, pages 81–87. Morgan Kaufmann, 1995.

[JW06] Thomas Jansen and Ingo Wegener. On the analysis of a dynamic evolutionary algorithm. *J. Discrete Algorithms*, 4:181–199, 2006.

References

[KHE15] G. Karafotias, M. Hoogendoorn, and A.E. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19:167–187, 2015.

[KK06] V. K. Koumousis and C. P. Katsaras. A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Transactions on Evolutionary Computation*, 10:19–28, 2006.

[KMH+04] Stefan Kern, Sibylle D. Müller, Nikolaus Hansen, Dirk Büche, Jiri Ocenasek, and Petros Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms - a comparative review. *Natural Computing*, 3:77–112, 2004.

[KS00] Natalio Krasnogor and Jim Smith. A memetic algorithm with self-adaptive local search: TSP as a case study. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'00)*, pages 987–994. Morgan Kaufmann, 2000.

[LS11] Jörg Lässig and Dirk Sudholt. Adaptive population models for offspring populations and parallel evolutionary algorithms. In *Proc. of Foundations of Genetic Algorithms (FOGA'11)*, pages 181–192. ACM, 2011.

[LS16] Johannes Lengler and Angelika Steger. Drift analysis and evolutionary algorithms revisited. *CoRR*, abs/1608.03226, 2016.

[Müh92] Heinz Mühlenbein. How genetic algorithms really work: Mutation and hillclimbing. In *Proc. of Parallel Problem Solving from Nature (PPSN'92)*, pages 15–26. Elsevier, 1992.

[Rec73] Ingo Rechenberg. *Evolutionsstrategie*. Friedrich Fromman Verlag (Günther Holzboog KG), Stuttgart, 1973.

[Smi12] Selmar K. Smit. *Parameter Tuning and Scientific Testing in Evolutionary Algorithms*. PhD thesis, VU University of Amsterdam, 2012. PhD thesis.

[SS68] Michael A. Schumer and Kenneth Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13:270–276, 1968.

[Thi05] Dirk Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'05)*, pages 1539–1546. ACM, 2005.

[TR98] Andrew Tuson and Peter Ross. Adapting operator settings in genetic algorithms. *Evolutionary Computation*, 6:161–184, 1998.

[Wit13] Carsten Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability & Computing*, 22:294–318, 2013.