Dynamic Parameter Choices in Evolutionary Computation

Carola Doerr
CNRS and Sorbonne University, Paris, France

Tutorial held at GECCO 2019, Prague, Czech Republic
http://gecco-2019.sigevo.org/

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07.
https://doi.org/10.1145/3319619.3323372

Version Management

- Please note that these slides are as of April 2019. I will almost surely revise the slides before July. You can find the **latest version** on my homepage:
  http://www-ia.lip6.fr/~doerr/DoerrGECCO19tutorial.pdf
- This tutorial was designed for GECCO attendees. If you are interested in the topic, but not familiar with the terminology used in evolutionary computation, please do not hesitate to contact me. I will be happy to discuss the ideas, methods, and results in a language that avoids terms like “fitness”, “mutation”, “crossover”, “selection” etc.
- Reference for this tutorial:

```
@inproceedings{Doerr19tutorial,
    author    = {Carola Doerr},
    title     = {Dynamic parameter choices in evolutionary computation},
    booktitle = {Proc. Genetic and Evolutionary Computation Conference (GECCO'19, Companion Material)},
    year      = {2019},
    url       = {https://doi.org/10.1145/3319619.3323372},
    doi       = {10.1145/3319619.3323372},
    publisher = {ACM}}
```


Tutorial Presenter: Carola Doerr

- **Carola Doerr, formerly Winzen,** is a permanent researcher with the French National Center for Scientific Research (CNRS) and the Computer Science Department LIP6 of Sorbonne University in Paris, France
- Main research interests:
  1. Evolutionary Algorithms and other randomized heuristics
     - Theory: Running Time Analysis, Black-Box Complexity, Anytime Performance Measures
     - Empirical Studies: Benchmarking, Landscape-Aware Algorithm Selection and Configuration
  2. Geometric Discrepancy Theory
- Selected formal roles:
  - Program Chair: PPSN 2020, FOGA 2019, GECCO theory track 2017 and 2015
  - Guest Editor of two special issues in Algorithmica
  - Co-organizer of two Dagstuhl seminars on Theory of Randomized Optimization Heuristics (2017 and 2019)
  - Vice chair of COST action 15140 on Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO)

Topic of this Tutorial: Parameter Control

Goals of Parameter Control

- ✓ to **identify** good parameter values “on the fly”
- ✓ to **track** good parameter values when they change during the optimization process

My Goal for this tutorial:
to inspire and to enable you to experiment with dynamic parameter choices

Focus: Discrete Black-Box Optimization

Our focus will be on discrete black-box 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

➔ even if your main interest is in continuous optimization, the mechanisms discussed below can (almost surely) be applied to your settings

Many examples in this tutorial originate from the theory of EC literature

- the problems and algorithms are easy to understand and to explain in the given time frame ("pure", "sterile" environments)
- we can compare performances with that of provably optimal algorithms
- the mechanisms are essentially the same as those used in practice (but algorithms and problems are simplified)

➔ even if you are not (yet ☺) interested in theoretical work, this tutorial offers a structured way to think about parameter control and provides many pointers to relevant literature (cf. also the reference list on the last slides of this handout, I preferred to add more content in the handout than what can be discussed in the tutorial)

Survey Articles

In 110 minutes we cannot discuss all existing works. Summaries of the state-of-the-art and pretty complete lists of references can be found in these surveys (see reference list on the last pages of these tutorial slides for details)

- **Empirical works:**
  - Karafotias, Hoogendoorn, Eiben, 2015 [KHE15] (an up to date survey of empirical works)
  - Aleti, Moser, 2016 [AM16] (additional pointers, systematic literature survey)
  - Eiben, Hinterding, Michalewicz, 1999 [EHM99] (classic seminal paper, introduces a now widely accepted classification scheme)
  - Lobo, Lima, Michalewicz, 2007 [LLM07] (book on parameter selection, includes chapters on tuning and control)
- **Theoretical works:**
  - Doerr, Doerr, 2018 [DD18b] (summarizes the state-of-the-art of theoretical works which prove performance bounds with mathematical rigor; introduces the revised classification scheme discussed below)

Questions and Feedback

- **Don't hesitate to ask questions**
  - If I am using a term 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
- **Comments are very welcome**
  → please share your experience with parameter selection!
- **I appreciate your feedback**
  - which parts did you (not) like?
  - was the speed accurate?
  - is there anything that you would like to see changed?
- **Related literature**
  - If you know of any works that should be cited in this tutorial, please kindly let me know

Part 1: Motivating Example

Motivating Example: LeadingOnes

- Classic benchmark problem often studied in the theory of evolutionary computation (as one the simplest examples of a non-separable function)
- Original function:

$$\mathrm{LO}: \{0,1\}^n \to \mathbb{R}, x \mapsto \mathrm{LO}(x) = \max \{i \in [n] \mid \forall j \leq i : x_i = 1\}$$

1	1	0	1	1	0	1	1

LO-value: 2 (2 initial ones)

- Looks like a "stupid" problem? For most EAs, it is equivalent to this game:

$$\mathrm{LO}_{z,\sigma}: \{0,1\}^n \to \mathbb{R}, x \mapsto \mathrm{LO}_{z,\sigma}(x) = \max \{i \in [n] \mid \forall j \leq i : x_{\sigma(i)} = z_{\sigma(i)}\}$$

0	0	0	1	1	0	1	1
4	8	5	1	7	2	6	3

secret code z

secret permutation σ

1	1	0	1	1	0	1	1

$\mathrm{LO}_{z,\sigma}$-value: 3 (first 3 bits in the order prescribed by σ are coincide with those of z)

Motivating Example: LeadingOnes

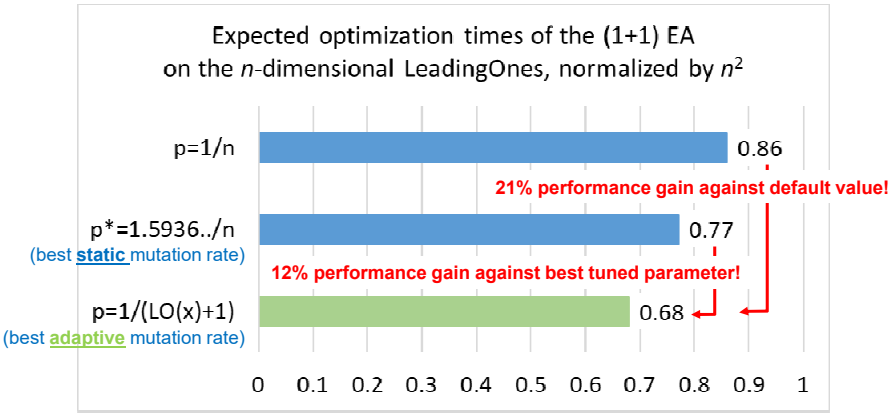
- Only way to optimize the LeadingOnes function is to identify the bits one after the other
  [it can be formally shown that the advantage of a parallel exploration is not very significant [AAD+13]. We won't discussed any details today]
- Most EAs need $\Omega(n^2)$ function evaluations to optimize this function
  - [LW12]: no unary unbiased (i.e., purely mutation-based) EA can have sub-quadratic running time
  - Crossover-based EAs can be faster [DJK+11,AAD+13]]

The (1+1) EA

- Initialization:
  Choose $x \in \{0,1\}^n$ uniformly at random (u.a.r.).
- Optimization: in iteration $t = 1,2,\ldots$ do
  1. create y from x by standard bit mutation \\ "mutation"
     i.e., flip each bit w/ probability p, independently of all other bits
  2. If $f(y) \geq f(x)$ replace x by y \\ "selection"

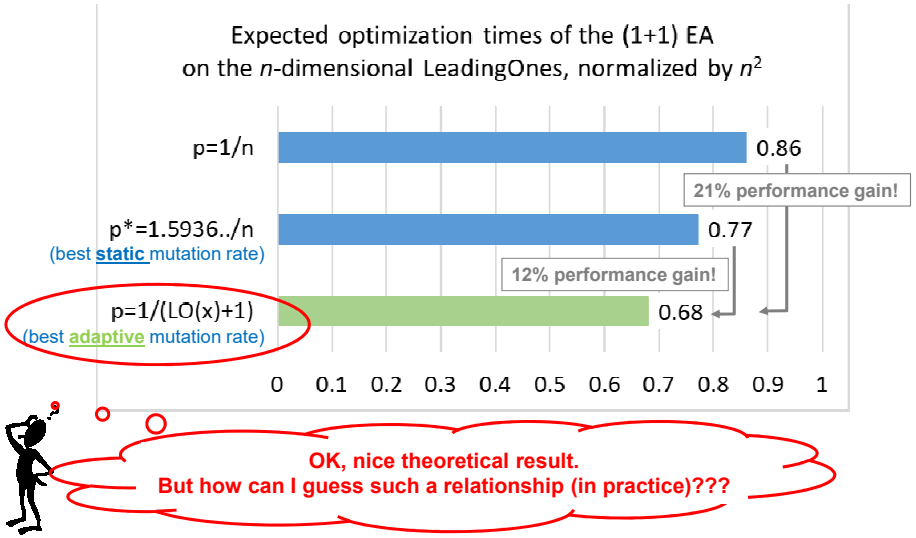
Critical parameter: the mutation rate p
(often recommended choice: $p = 1/n$)

Proven Optimization Times for (1+1) EA on LeadingOnes



These results were proven in [BDN10]

Proven Optimization Times for (1+1) EA on LeadingOnes

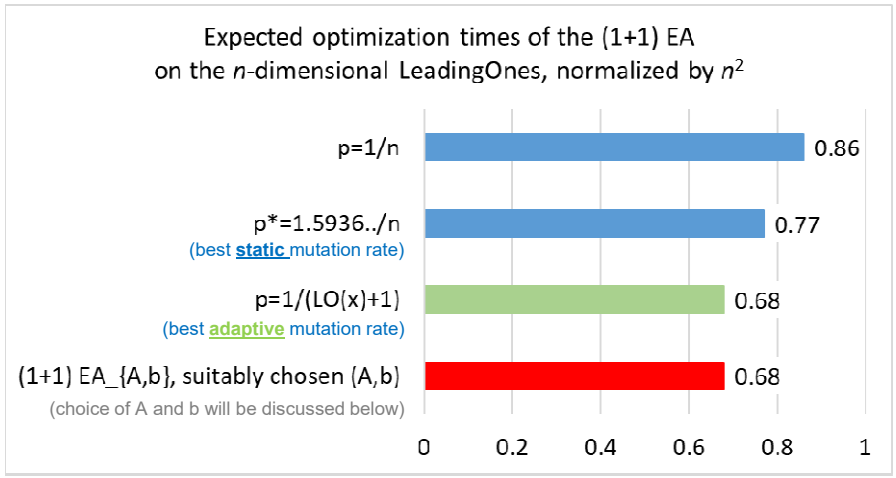


(1+1) EA with Adaptive Mutation Rates

- Initialization:
  1. Choose $x \in \{0,1\}^n$ uniformly at random (u.a.r.).
  2. Initialize $p = 1/n$
- Optimization: in iteration $t = 1,2,\ldots$ do
  1. create y from x by standard bit mutation w/ mutation rate p
  2. If $f(y) \geq f(x)$
     - replace x by y \\ selection
     - replace p by Ap \\ parameter update
  3. If $f(y) < f(x)$
     - replace p by bp \\ parameter update

based on a variant from [DW18]

Proven Optimization Times for (1+1) EA on LeadingOnes



This result was proven in [DDL19, GECCO 2019, theory track]

Illustration taken from [DW18]

- Results for LeadingOnes, $n = 250$
- Update strengths: $A = 2, b = 1/2$ [=optimal number of random bits to flip]
- Plot compares optimal mutation strengths with the ones found by the adaptive (1+1) $EA_{>0}$ [i.e., the (1+1) EA but we enforce $y \neq x$ by re-sampling if needed]

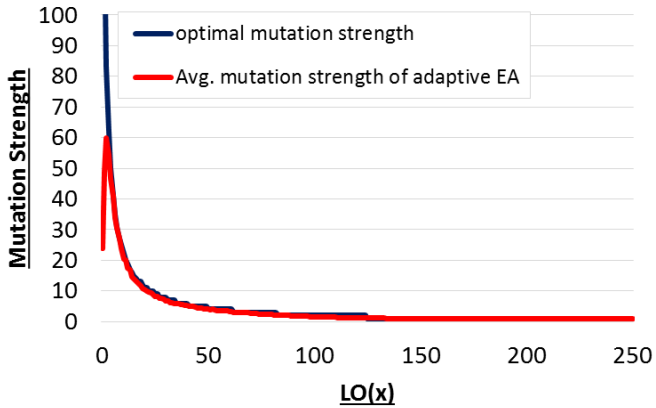


Illustration taken from [DW18]

- Results for LeadingOnes, $n = 250$
- Update strengths: $A = 2, b = 1/2$
- Plot compares optimal mutation strengths with the ones found by the adaptive (1+1) $\mathrm{EA}_{>0}$ [i.e., the (1+1) EA but we enforce $y \neq x$ by re-sampling if needed]
- Same plot, logarithmic scale, zoom into $\mathrm{LO}(x) \leq 150$:

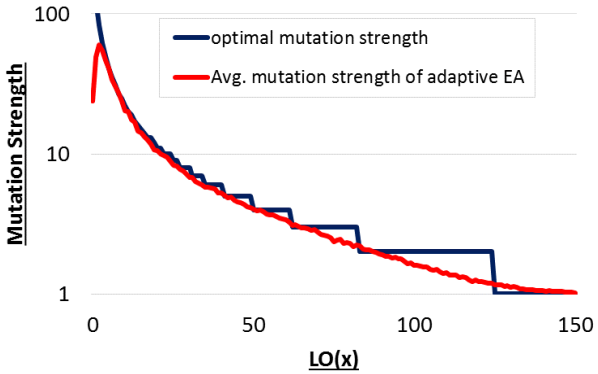
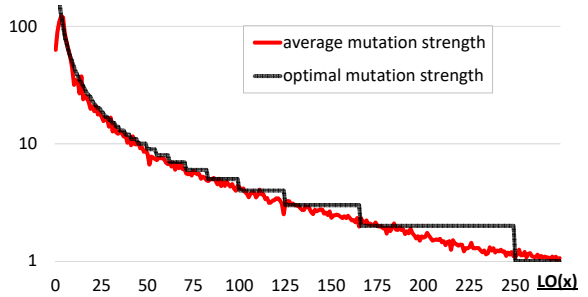


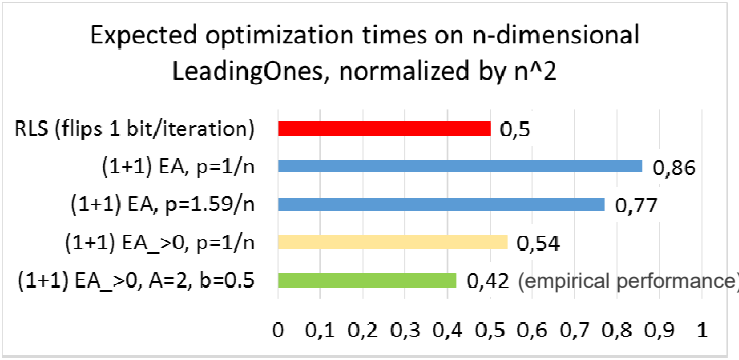
Illustration taken from [DW18]

- Results for LeadingOnes, $n = 500$
- Update strengths: $A = 2, b = 1/2$
- Plot compares optimal mutation strengths with the ones found by the adaptive (1+1) $\mathrm{EA}_{>0}$ [i.e., the (1+1) EA but we enforce $y \neq x$ by re-sampling if needed]
- Logarithmic scale, zoom into $\mathrm{LO}(x) \leq 250$:

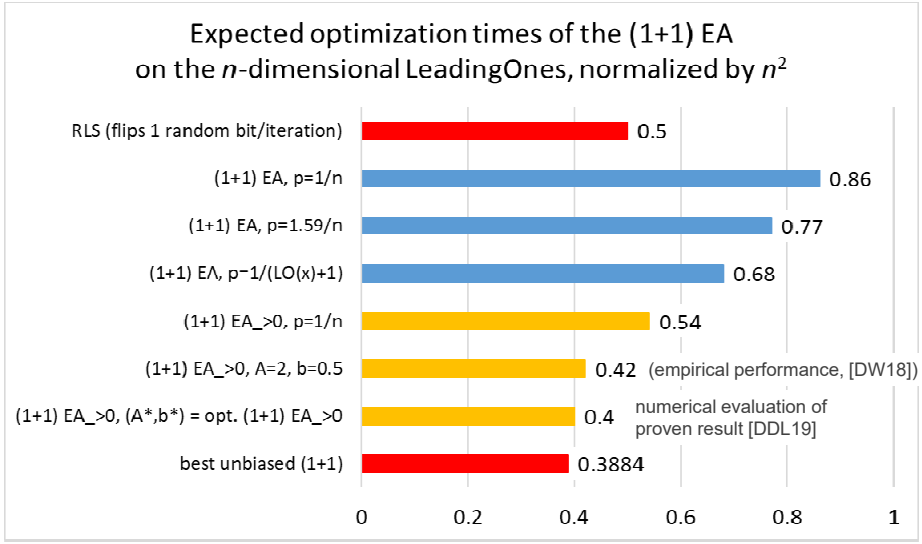


Motivating Example (from [DW18])

- Running time for update strengths $A = 2$, $b = 1/2$
  - around 20.5% performance gain over the (1+1) $\mathrm{EA}_{>0}$ with static mutation rate $p = 1/n$
  - 14% performance gain over RLS
- larger gains possible for other combinations of A and b
  (see next slide for details)

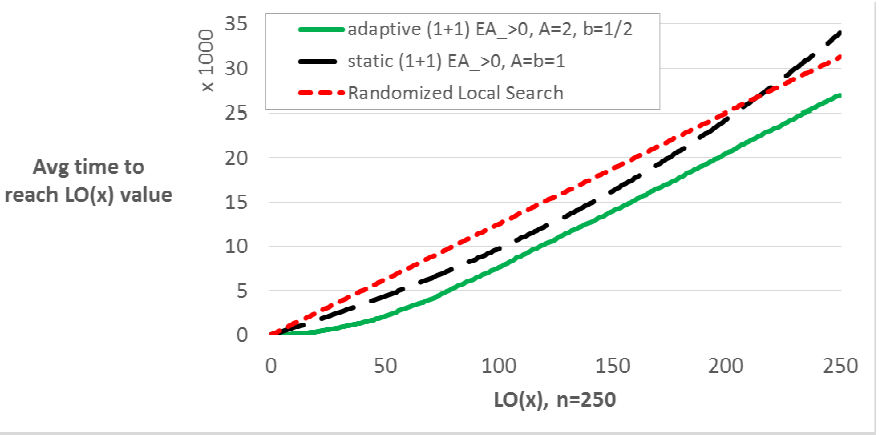


Proven Optimization Times for (1+1) EA on LeadingOnes



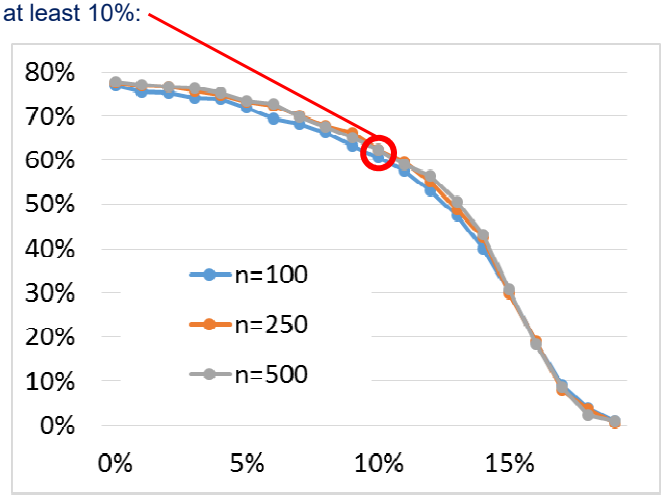
Anytime Performance (data from [DW18])

Zooming into the optimization process:
comparison of **fixed-target running time**



Robustness of Hyper-Parameters (data from [DW18])

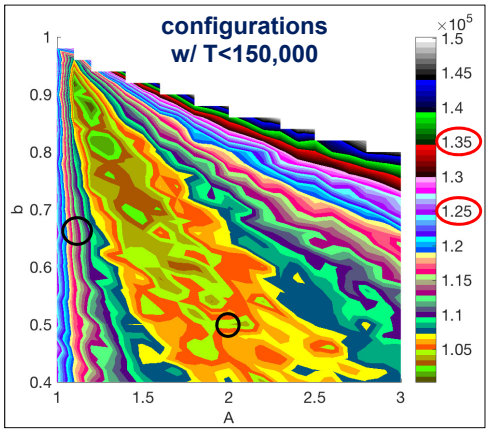
The performance gain is not very sensitive with respect to the choice of the hyper-parameters A and b:
More than 60% of all configurations with $1 < A \leq 2.5$ and $0.4 \leq b < 1$ are better than RLS (!) by at least 10%:



Fitness Landscape of Tuning Problem (data from [DW18])

The performance gain is not very sensitive with respect to the choice of the hyper-parameters A and b:

Heatmap shows average optimization time for different combinations of A and b for the adaptive (1+1) EA on 500-dimensional LeadingOnes
(the static (1+1) $EA_{>0}$ needs ≈ 135,000 function evaluations, RLS 125,000)



Typical values for A and b are

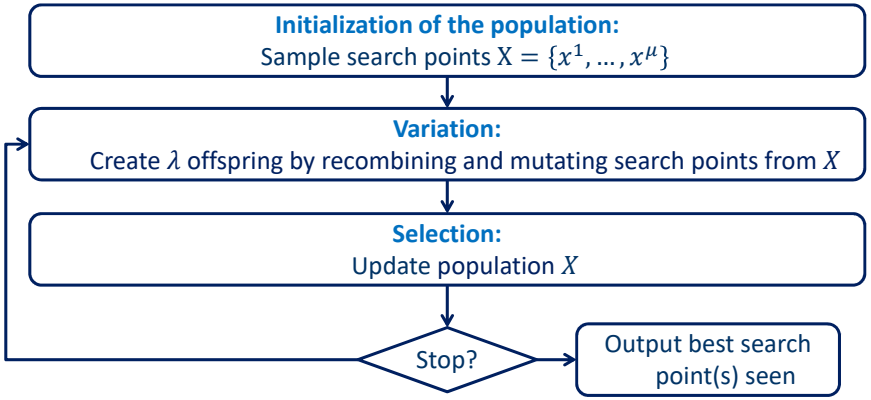
- $A = 2, b = \frac{1}{2}$
  (gives an avg runtime of ≈ 104,000)
- 1/5th success rule:
  $A = \left(\frac{3}{2}\right)^{1/4} \approx 1.11, b = \frac{2}{3}$
  (gives an avg runtime of ≈ 115,000)

Part 2:
Parameter Setting Matters!

(and a little bit of history of parameter control)

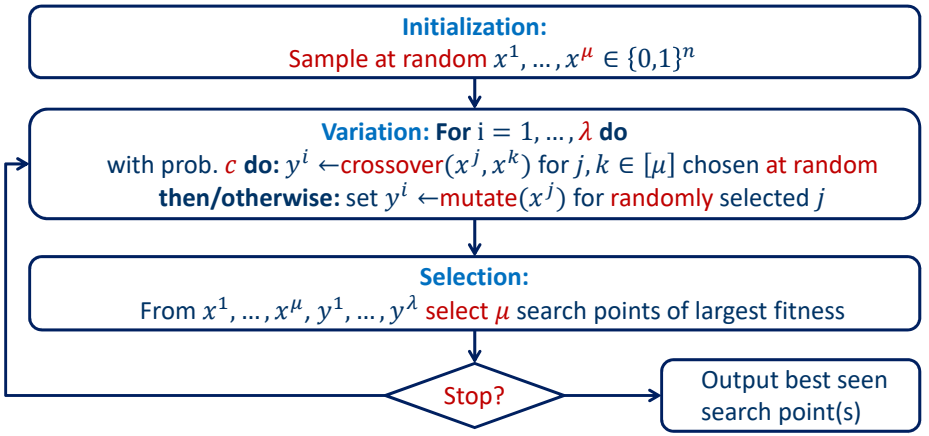
Simplified EA Blueprint

To simplify our discussions, we will use the following blueprint to model evolutionary algorithms. (The mechanisms presented below can also be used to adapt the parameters of other heuristics, which do not follow this scheme!)



(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???

Are Parameter Values Important?

- 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?*

Globally Good Parameter Values?

- "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.

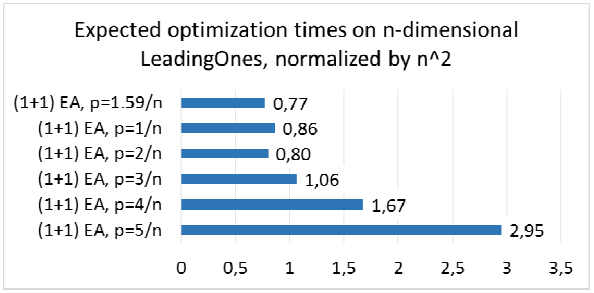
Parameter Tuning

- "Modern view" of parameter selection: no globally optimal parameter values exist
  → parameters need to be adjusted to the problem at hand
- Typical tuning approach:
  - run some initial tests and observe how the performance depends on the chosen parameter values
  - choose the parameter values that seem most promising
- **Quite sophisticated tools** for parameter tuning are available:
  - irace [LDC+16], SPOT [BBFKK10], GGA [AMS+15], ParamILS [HHLBS09], SMAC [HHLB11]
  - Advantage of these tools: automated identification of reasonable parameter values
  - Disadvantage: recommended parameter values are static!

The bulk of EC papers with a focus on discrete optimization problems analyzes EAs' performance with respect to some fixed set of parameters! *(How about your latest work?)*

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
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)
   - Example: LeadingOnes



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
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)
   - Example: (plot on previous slide)
   - Those of you interested in theoretical results can find in [DoerrJS+13] or [LS16] 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

Aim of Parameter Control

- ✓ To **identify** good parameter values "on the fly"
- ✓ To **track** good parameter values when they change during the optimization process

“On the Fly” Identification of Good Parameter Values

Example: OneMax: $\mathrm{OM}: \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_i x_i$ (“drosophila” of EC theory)

1	1	0	1	1	0	1	1

for most EAs, this problem is equivalent to the Hamming distance problem:

$$\mathrm{HD}_z: \{0,1\}^n \to \mathbb{R}, x \mapsto \#\{i \mid x_i = z_i\}$$

0	0	0	1	1	0	0	1

Secret string z

1	1	0	1	1	0	1	1

HD_z-value: 5
(in 5 position the bit value coincides with that of z)

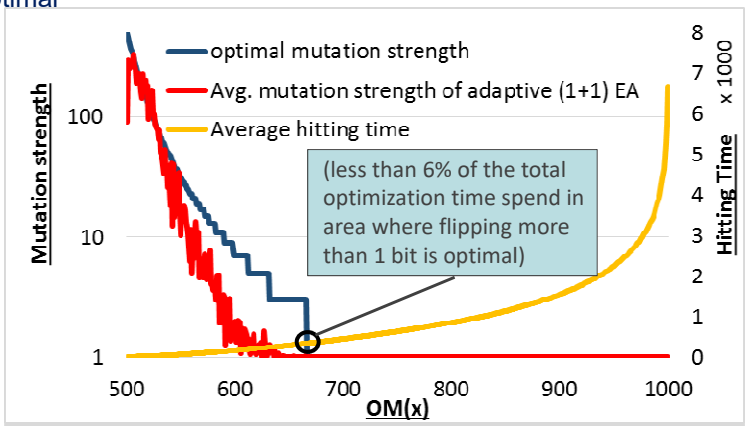
“On the Fly” Identification of Good Parameter Values

Example: OneMax: $\mathrm{OM}: \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_i x_i$ (“drosophila” of EC theory)

for most EAs, this problem is equivalent to the Hamming distance problem:

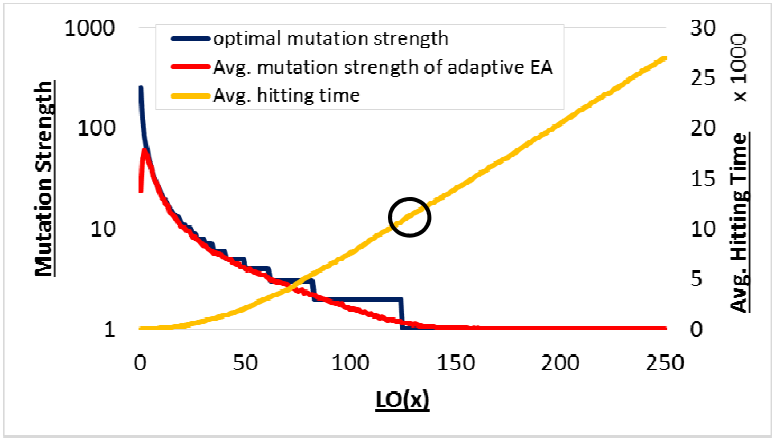
$$\mathrm{HD}_z: \{0,1\}^n \to \mathbb{R}, x \mapsto \#\{i \mid x_i = z_i\}$$

- for most of the time a static choice of flipping one bit per iteration is optimal



Tracking Dynamic Optimal Values

Example: The LeadingOnes Problem.
Looking again at the 250-d example from above, we see that about 40% of the total optimization time is spend in stages in which flipping more than 1 bit is optimal



Aim of Parameter Control

- ✓ To **identify** good parameter values “on the fly”
- ✓ To **track** good parameter values when they change during the optimization process

Important: Not only constant factor improvements, but also asymptotic factor gains possible! (cf. page 71 for an example)

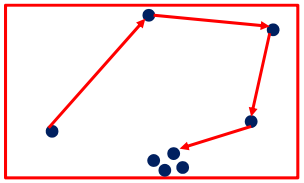
Part 3:
Parameter Control – Introduction

Basic Intuition

The most basic parameter control techniques use the following intuition

- beginning of the optimization process = "*exploration phase*", i.e., we want to explore different areas of the search space

  → use a large mutation rates to allow for large jumps
  → use small selective pressure to overcome local optima more easily
- end of the optimization process = "*exploitation phase*"
  → small mutation rates/high selective pressure to focus the search



Main Questions in Parameter Control

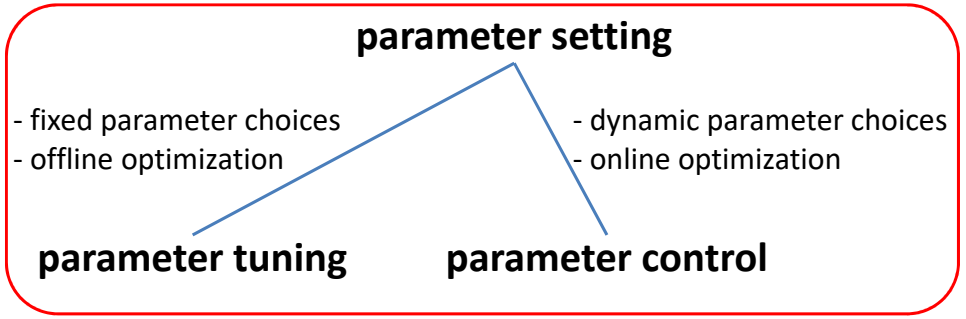
1. *Which parameter* is adapted?
   (and who is affected: 1 individual vs. whole population)
   1. Population size
   2. Mutation rate, Crossover probability
   3. Selection pressure
   4. Fitness function (e.g., penalty terms for constraints)
   5. Representation
   6. …
2. What is the *basis/evidence for the update*?
   1. time elapsed: number of fitness evaluations, generation count, CPU time
   2. progress, e.g., in terms of absolute or relative fitness gain
   3. diversity measures
   4. …
3. *How* do we select the parameter(s):
   1. multiplicative updates
   2. learning-inspired parameter selection
   3. endogenous/self-adaptive parameter selection: use EAs to find good values
   4. hyper-heuristics
   5. …

Classification Scheme of [EHM99]

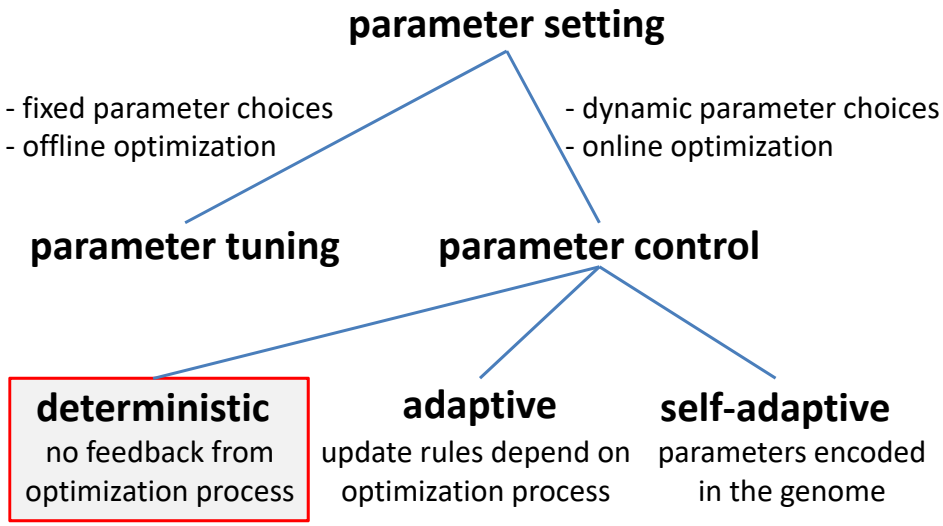
- Many attempts to find unifying taxonomy for parameter choices exist (cf. page 168 in [KHE15] for a survey)
- To date, the most popular classification scheme is that of Eiben, Hinterding, Michalewicz [EHM99], which we discuss on the next slides

Classification Scheme of [EHM99]

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



Classification Scheme of [EHM99]



"Deterministic" Parameter Control

Key intuition:

- Belief that optimal parameters often follow a similar pattern
- Example pattern: "first allow for exploration, then for exploitation"
- to stimulate or enforce such a pattern, time-dependent parameter settings can be used (where time = number of generations, fitness evaluations, wall-clock time, etc.)
  - **Examples:**
    1. cooling schedule of the selective pressure ("temperature") in selective pressure of Simulated Annealing. Often used update scheme: $T(t) = \alpha^t T(0)$ (multiplicative updates)
    2. start with some (large) mutation rate $p(0)$, decrease p after every 10,000 fitness evaluations
    3. after each 1,000 iterations, draw a random mutation probability

Remarks on "Deterministic" Parameter Control

- The last example on the previous slide shows that---as already acknowledged in [EHM99]---the term "deterministic" is not very well chosen
  → the choice can be random!
  → the only important feature is that it depends only on the time elapsed so far, and not on any other feedback of the optimization process
- 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
- Also 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 Control (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

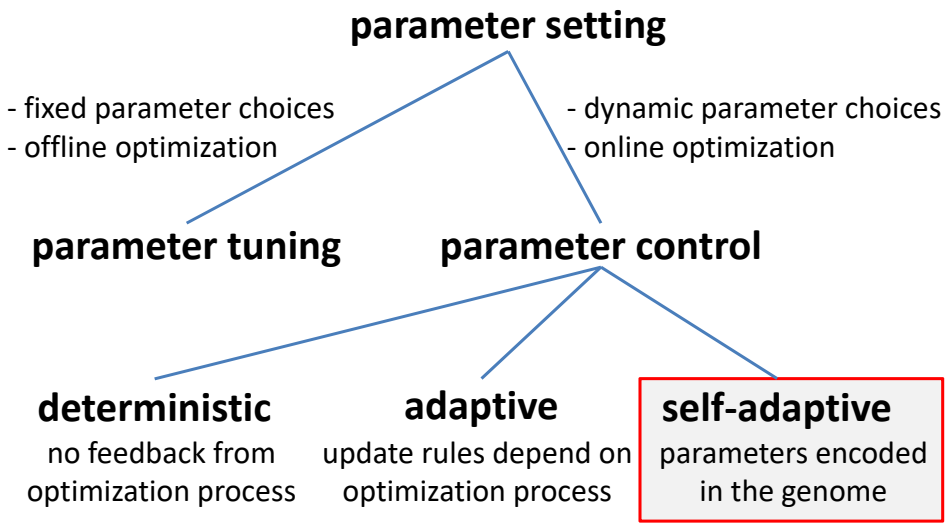
    - worse performance on simple functions like OneMax, linear functions, LeadingOnes, etc.

    + examples where better performance than any static choice can be proven
  - Doerr, Doerr, Kötzing [DDK18]: in every iteration, a random step size is used for a multi-valued OneMax-type problem (this problem will be discussed in more detail in the next section, along with a self-adjusting parameter choice. the algorithm that we refer to here is the one using a static probability distribution from which the step sizes are sampled)

Examples for "Deterministic" Parameter Control (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

Classification Scheme of [EHM99]

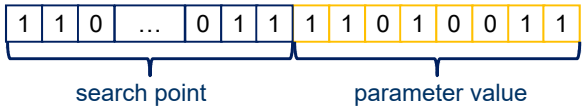


Self-Adaptive Parameter Control

- Parameter Control Idea 2:

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

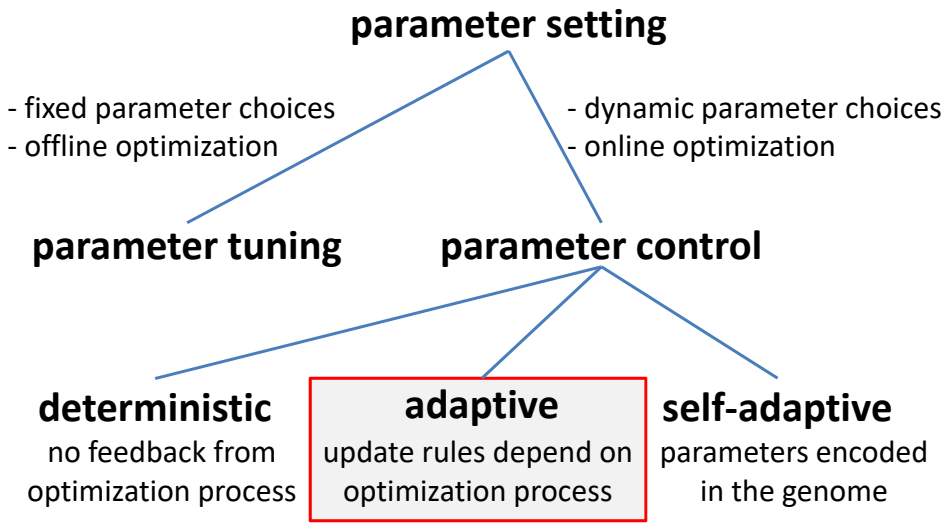
- Many different ways to do this. Examples (sketched, much room for creativity here!):
  1. Create a new population of parameter values, choose from this parameter values, possibly apply variation to them, and employ them in your EA, select based on progress made
  2. append to the solution candidates 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



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] and B. Doerr, Witt, Yang [DWY18] : theoretical works on a self-adaptive choice of the mutation strength in a non-elitist population

Classification Scheme of [EHM99]



Adaptive Parameter Control

- Parameter Control Idea 3:
  - use feedback from the optimization process
  - change the parameters according to some pre-described rule
- Relevant feedback includes:
  - *function values* of the search points in the population
  - *diversity* of the search points
  - *absolute or relative progress* obtained within the last τ iterations
  - …

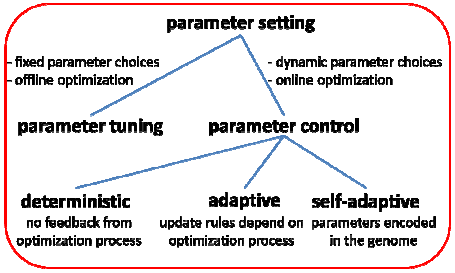
In my opinion, adaptive control mechanisms offer a very promising direction for future work. The remainder of this tutorial therefore has a strong focus on such update mechanisms

Example: Success-Based Multiplicative Success

The above-mentioned (1+1) $\mathrm{EA}_{>0}$ variant from [DW18] uses success-based multiplicative updates:

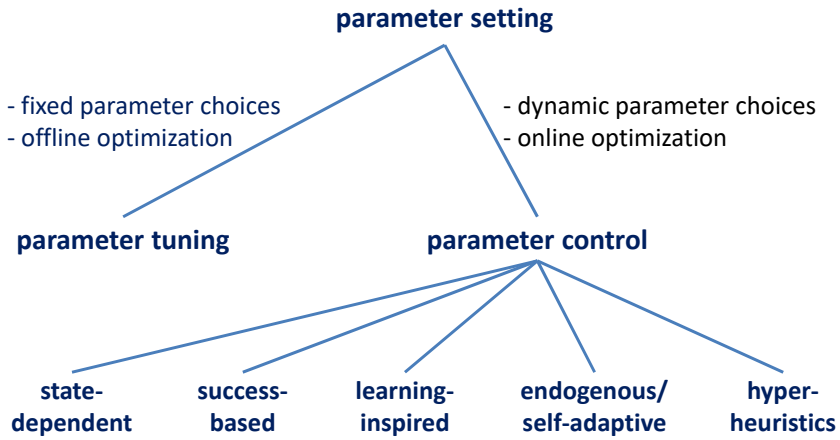
- Initialization:
  1. Choose $x \in \{0,1\}^n$ uniformly at random (u.a.r.).
  2. Initialize $p = 1/n$
- Optimization: in iteration $t = 1,2,\ldots$ do
  1. create y from x by standard bit mutation w/ mutation rate p (make sure that $y \neq x$, by re-sampling if necessary)
  2. If $f(y) \geq f(x)$
     1. replace x by y \\ selection
     2. replace p by Ap \\ parameter update
  3. If $f(y) < f(x)$
     replace p by bp \\ parameter update

Comment on Classification Scheme of [EHM99]



- The terms "deterministic", "adaptive", and "self-adaptive" have not been formally defined
  → be aware that they are not used very consistently in the literature
- Since [EHM99] almost 20 years have passed.
  → The field has advanced considerably
  (but maybe not to the extend it should have, as also noted in [KHM15])
  → we feel that time has come to introduce a different taxonomy

Revised Classification [DD18b]



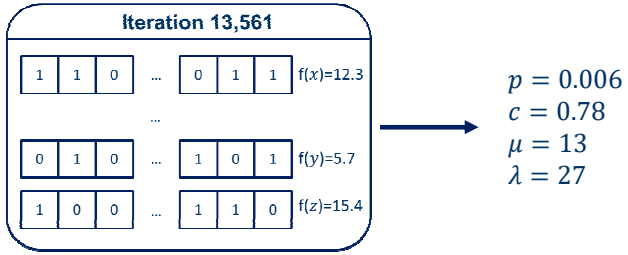
This classification scheme will be discussed in detail in the next section

Part 4: Examples for Parameter Control Mechanisms

Part 4a: State-Dependent Parameter Selection

State-Dependent Parameter Selection

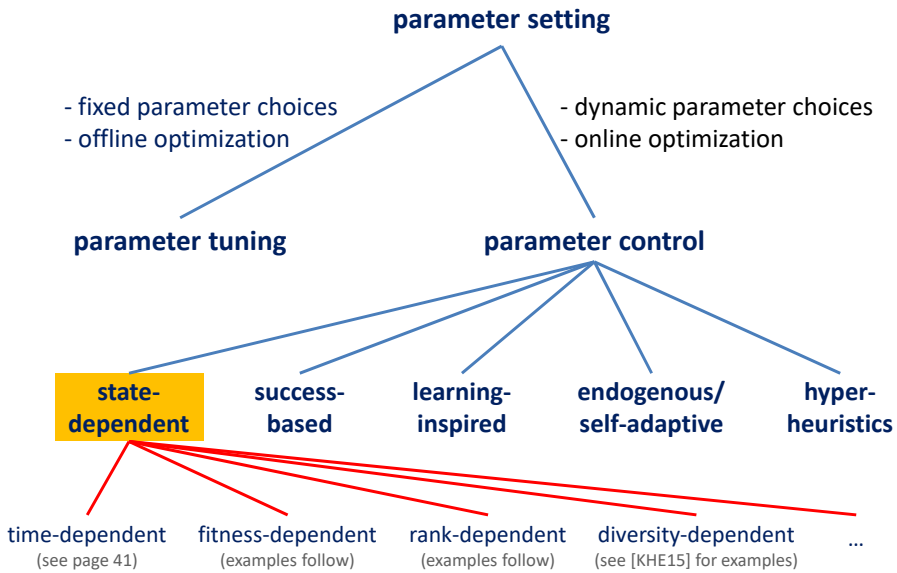
- State-dependent parameter selection mechanisms do not depend on the history of the optimization process, but only on the current state
- Analogy for this functional dependence: take a "screenshot" of the current population and map it to parameter values



$$p = 0.006$$
$$c = 0.78$$
$$\mu = 13$$
$$\lambda = 27$$

- Most commonly used indicators for the state of the algorithm:
  - time elapsed so far (# fitness evaluations, iteration counter, CPU time, ...)
    → corresponds to "deterministic" parameter setting in the classification [EHM99]
  - function values (absolute values, diversity, ranks,…)
  - genotypic properties (e.g., diversity of the population)

Revised Classification [DD18b]



Fitness-Dependent Parameter Selection

- Requires a good understanding of how the parameters should depend on the function values
- Has been looked at
  - empirically, e.g., Bäck [Bac92,Bac96], Fialho, Da Costa, Schoenauer, Sebag PPSN'08 [FCSS08] and follow-up works for OneMax

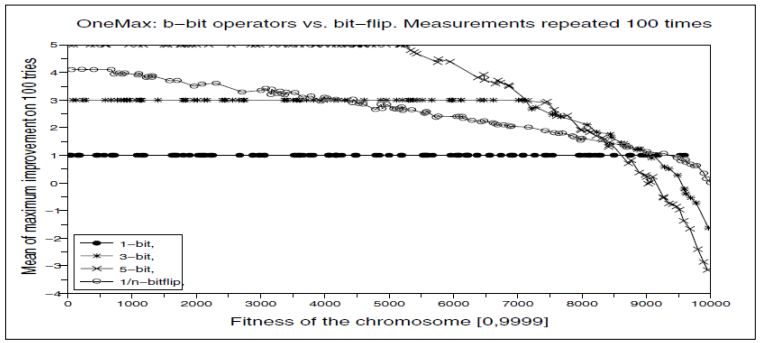


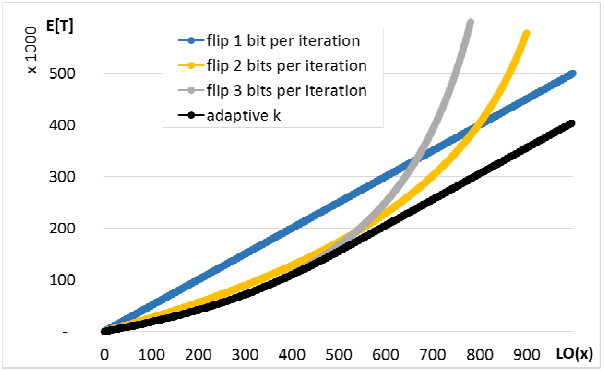
Figure 1 in [FCSS08]

Fitness-Dependent Parameter Selection

- Requires a good understanding of how the parameters should depend on the function values
- Has been looked at
  - empirically, e.g., Bäck [Bac92,Bac96], Fialho, Da Costa, Schoenauer, Sebag PPSN'08 [FCSS08] for OneMax
  - theoretically, e.g.,
    [BD19, DDY16b,BLS14] for OneMax and
    [D18,BDN10,DW18] for LeadingOnes

Fitness-Dependent Parameter Selection

In situation in which the optimal parameter choice depends only on the function value of the current best solution* these bounds help us bound the maximal potential of parameter control, i.e., they tell us how much an ideal parameter control technique could gain over the best static parameter setting



Fixed-Target running times for LeadingOnes in dimension n=1000

Note that the gradient of these fixed-target curves is essential here!

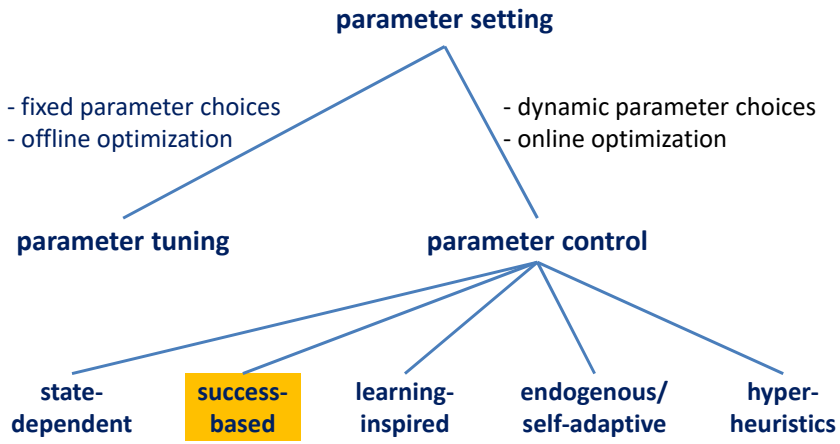
*for many EAs, in particular (1+λ)-type EAs, this is the case when optimizing OneMax or LeadingOnes problems

Rank-Dependent Parameter Selection

- **Basic idea:**
  - bad search points should undergo large variation (→ large mutation rates)
  - good individuals should be modified only moderately (→ small mutation rates)
- **Example:**
  - Cervantes, Stephens IEEE TEC [CS09]:
    - rank search points in the current population
    - each search point is assigned a mutation rate that depends on its rank:
      - rank 1: mutation rate $p_{\min}$ // best individual of population
      - ... (linear interpolation)
      - rank s: mutation rate $p_{\max}$ // worst individual of population
    - the rank-based GA first selects an individual from the population and then modifies it with the mutation rate given by this ranking
  - Theoretical study of this algorithm are available in [OLN09]

Part 4b: Success-Based Parameter Selection

Revised Classification [DD18b]



Success-Based Parameter Selection

- **Basic idea:** after each (or after every τ) iteration(s) adjust the current parameter value depending on whether or not the last (τ) iteration(s) have been successful
- Examples for "success":
  - a strictly better search point has been found
    // this is probably the most common measure
  - a search point of at least the same fitness has been found
    // used by the adaptive (1+1) $\mathrm{EA}_{>0}$ from [DW18]
  - a fitness-increase of at least x% could be observed
  - the diversity has been increased
  - …
- Success-based parameter selection is classified as "adaptive parameter control" in the taxonomy of [EHM99]

The 1/5th Success Rule (1/2)

- Probably *the* most famous success-based parameter adaptation rule
- Rechenberg [Rec73]:
  - observed that for the sphere function and a corridor landscape the optimal success rate of the (1+1) ES is around 1/5 (i.e., there is some theoretical foundation of this rule)
  - Suggestion:
    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]

The 1/5th Success Rule (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
- **Intuition:**
  - when success is too likely to happen, we seem to be in an easy part of the optimization problem
    →increasing mutation rates might result in larger progress per step
  - when success is happening too seldom, we could be approaching the optimum and should focus our search
    →decrease mutation rate for a more conservative search

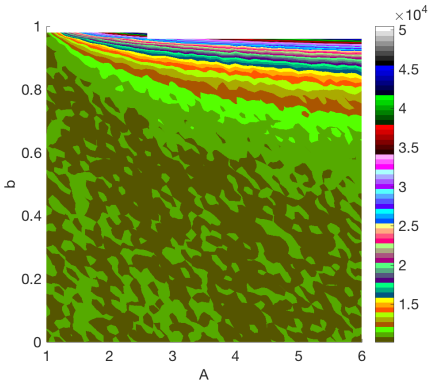
- Note 1: there is also justification to do this the other way around, i.e.,
  If (iteration successful) → decrease mutation rate
  If (iteration not successful) → increase mutation rate
  (think of jump functions or other functions with a local optimum from which the algorithm needs to escape)
- Note 2: the same idea can also be used to control other parameters, such as the population size, crossover probabilities, etc.

Multiplicative Updates Inspired by 1/5-th rule

- We have seen in Section 1 an example for a success-based update rule, the (1+1) EA with dynamic mutation rates. It uses the following update rule
  1. If iteration was successful (i.e., if $f(y) \geq f(x)$)
     replace p by Ap \\ update strength $A > 1$
  2. If iteration was not successful (i.e., if $f(y) < f(x)$)
     replace p by bp \\ update strength $b < 1$
- An interpretation of the 1/5th success rule from [KMH+04] recommends to use $\boldsymbol{A = (1/b)^{1/4}}$
- **Intuition:** if one our of 5 iterations is successful, the parameter value does not change
- **Example:** $b = 2/3, A = (3/2)^{1/4} \approx 1.10668\ldots$ [also used in [Aug09]]

(1+1) $\mathrm{EA}_{>0}$ with Success-Based Mutation Rates

- The algorithm from Section 1 works also on OneMax, for a broad range of parameters:



Average optimization times for 1500-dimensional OneMax for different combinations of update strengths A and b

Simple Success-Based Rules: Example 1

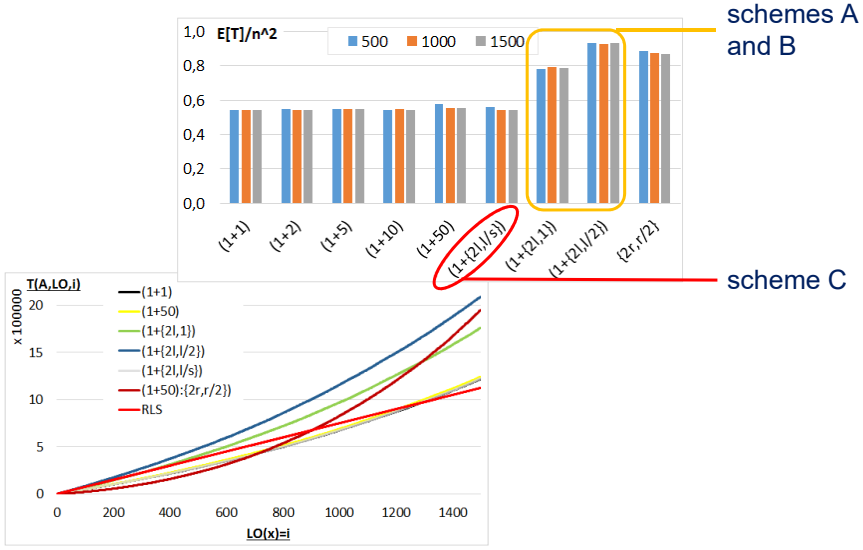
- Lässig, Sudholt: Adaptive Population Models for Offspring Populations and Parallel Evolutionary Algorithms, FOGA 2011 [LS11]:
  - regard the $(1+\lambda)$ EA
  - an iteration is called *successful* if it produces an offspring of better than previous best fitness value
  - **Scheme A:**
    - If (iteration 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 1

Below are results from [DYvR+18] for LeadingOnes



Simple Success-Based Rules: Example 2

- In [DDK18] 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 minimizing 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 $x \leftarrow y$;

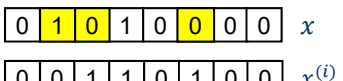
- For suitable $a > 1$ and $b < 1$ (e.g., $a \in [1.7, 2]$ and $b \in [0.8, 0.9]$) this algorithm achieves an expected optimization time of $\Theta\big(n(\log n + \log r)\big)$, which is best possible among all (static and non-static) parameter choices
- We do not know if any static parameter choice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

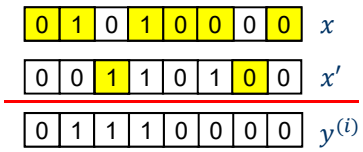
Simple Success-Based Rules: Example 3

- 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, and this not only by a constant factor
  - there are quite a few open questions, interesting for both empirically- and theory-oriented researchers
- References for this part:
  1. [DDE13] (GECCO 2013) and [DDE15] (TCS 2015, journal version of [DDE13]) suggested the (1+(λ,λ)) GA
  2. [DD18a], which is a summary of
     - [DD15b] (GECCO'15): optimal bounds for static parameter settings
     - [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;





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** $f(y) \geq f(x)$ **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 [Doe16]

How to Chose λ in the (1+(λ, λ)) GA?

- In [DDE13] we also observed that a fitness-based choice of λ gives a better result: for $\lambda = \sqrt{\frac{n}{n - f(x)}}$, the expected runtime of the (1+(λ, λ)) GA on OneMax is $O(n)$
  - This linear runtime is better than what any (!) static parameter value can achieve (by the results presented in [Doe16])

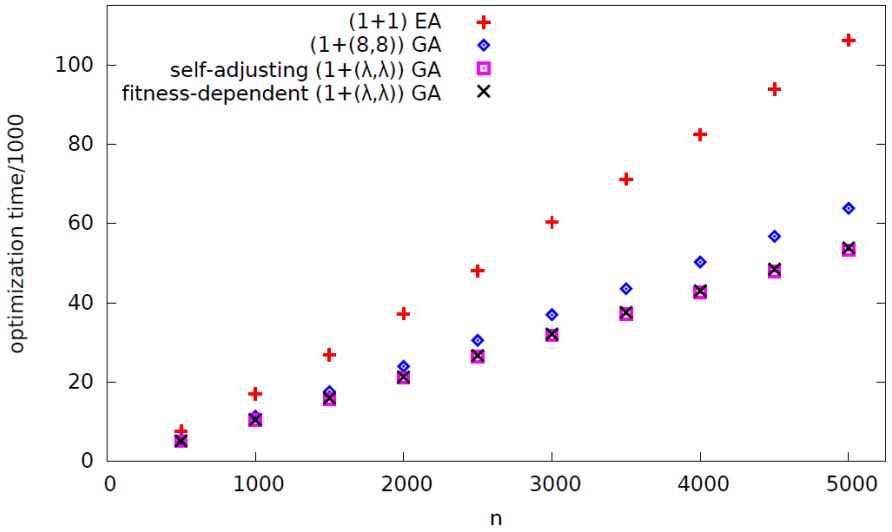
    → one of the few examples where a non-static choice can be proven (with mathematical rigor) to outperform static parameter setting
  - Linear runtime can also be shown to be the best possible achievable runtime
- Disadvantage of this non-static, fitness-dependent choice: hard to guess such a functional relationship! (it was not very difficult to see it from the proofs, but in practice, guessing such a relationship is probably not feasible)
- **Main question:**

  Is there a way to achieve similar performance in an automated way?

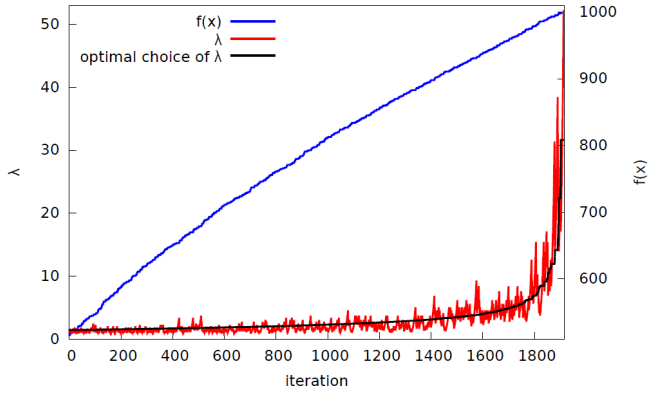
Self-Adjusting λ in the (1+(λ, λ)) GA (2/2)

- 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 from the works [Aug09] and [KMH+04], 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 ☺ (If you are 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

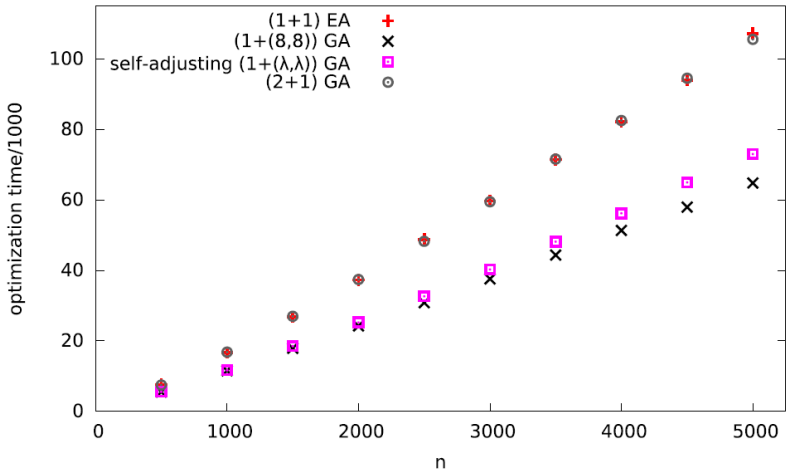
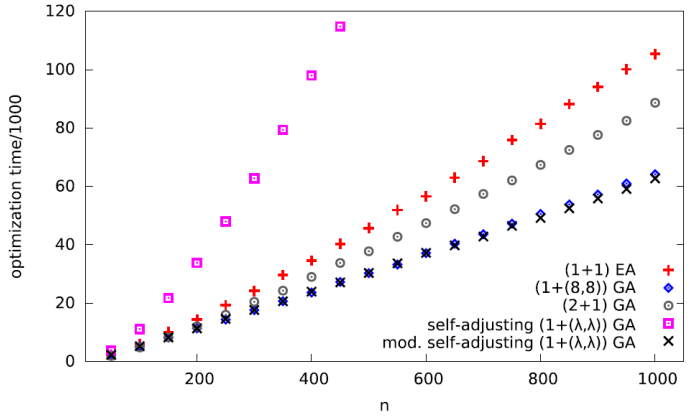


Fig. 6. Average optimization time of the $(1 + (\lambda, \lambda))$ GA on linear functions with random weights $w_i \in [1, 2]$.

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

Good Performance Also for Other Test Functions

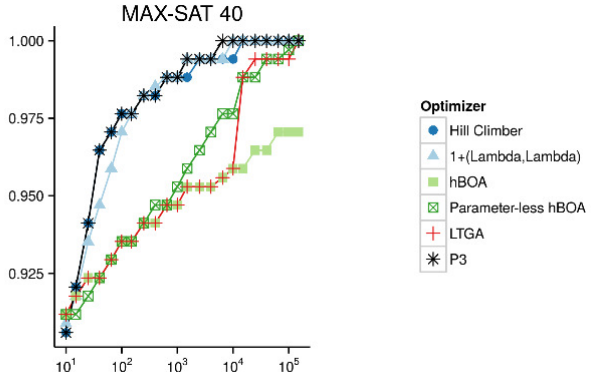


- Performance on royal road functions with block size 5

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

- 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+($\boldsymbol{\lambda}$,$\boldsymbol{\lambda}$)) GA on MaxSAT available [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 [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 $T^{\text{gen}} = \Theta\left(\frac{n}{\log \lambda} + \frac{n \log n}{\log \lambda}\right)$ 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

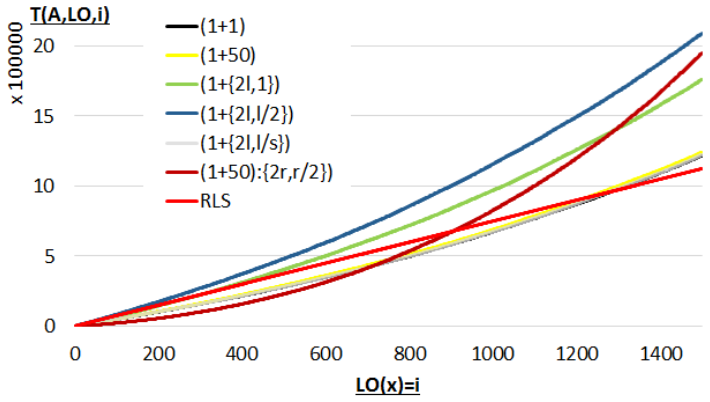
Example 4: The $(1+\lambda)$ EA on OneMax

Doerr, Giessen, Witt, Yang (GECCO'17) [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 $T^{\text{gen}} = \Theta\left(\frac{n}{\log \lambda} + \frac{n \log n}{\log \lambda}\right)$ performance

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

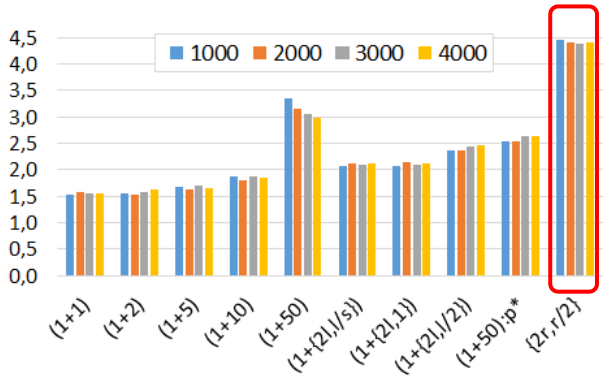
In [DYvR+18] we tested this algorithm from [DGWY17] and obtain the following interesting behavior on LeadingOnes:



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

In [DYvR+18] we tested this algorithm from [DGWY17] and obtain the following interesting behavior on LeadingOnes…

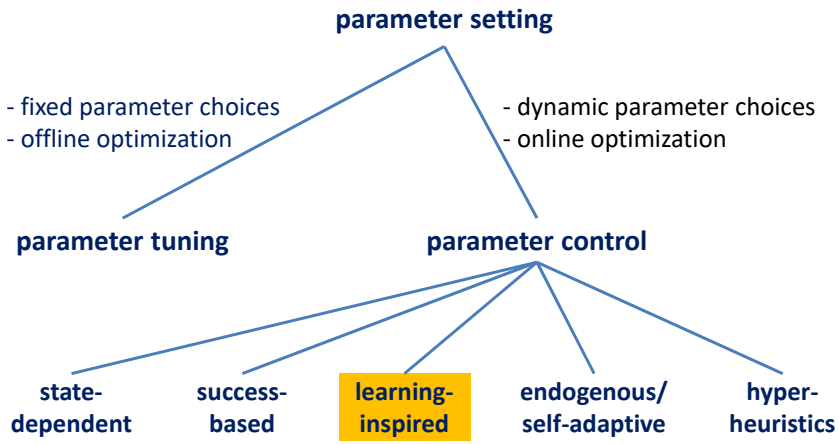
… while on OneMax the following happens:



→ this calls for a more detailed analysis of the hyper-parameters

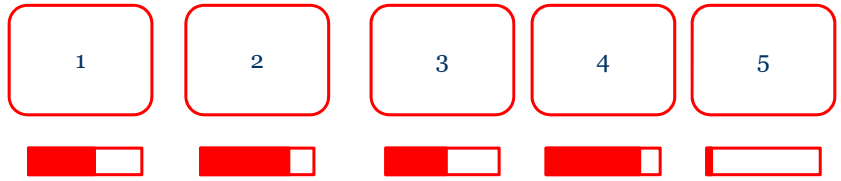
Part 4c: Learning-Inspired Parameter Selection

Revised Classification [DD18b]



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: multi-armed bandits (MAB)
  - K 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



(Another) Exploration/Exploitation Trade-Off

- **Main difficulty: exploitation vs. exploration trade off**

exploitation: we want, of course, to use an optimal parameter value as often as possible

exploration: 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)

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). Some *hyper-heuristics* are also learning-based.
- 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!

Dynamic Multi-Armed Bandits View

- K different parameter values
  - p_t^i probability to chose operator i in iteration t $(p_t^1, p_t^2, \dots, p_t^K)$
  - c_t^i confidence in operator i at iteration t $(c_t^1, c_t^2, \dots, c_t^K)$
- Main questions: how to update probabilities? how to updates confidence?
  - well-studied questions in *machine learning*!
  - But: main focus in ML is for static "*rewards*"
  - main difference to EC: our "rewards" (success rate, fitness increase, etc) change over time.
- 2 first ideas:
  1. **Probability Matching:**
     - $c_{t+1}^i = (1-\alpha)\, c_t^i + \alpha r^t$, where i is the operator selected in iteration t and r^t is the reward of that iteration
     - $p_{t+1}^i = p_{\min} + (1 - K p_{\min}) \frac{c_{t+1}^i}{\sum_{j=1,\dots,K} c_{t+1}^j}$,
     - p_t^i is proportional to c_t^i while maintaining a minimal amount of exploration

α controls the speed of confidence adaptation

minimal level of exploration

Dynamic Multi-Armed Bandits View

- K different parameter values
  - p_t^i probability to chose operator i in iteration t $(p_t^1, p_t^2, \dots, p_t^K)$
  - c_t^i confidence in operator i at iteration t $(c_t^1, c_t^2, \dots, c_t^K)$
- Main questions: how to update probabilities? how to updates confidence?
  - well-studied questions in *machine learning*!
  - But: main focus in ML is for static "*rewards*"
  - main difference to EC: our "rewards" (success rate, fitness increase, etc) change over time.
- 2 first ideas:
  2. **Adaptive Pursuit** [Thierens GECCO 2005]**:**
     - $c_{t+1}^i = (1-\alpha)\, c_t^i + \alpha r^t$, where i is the operator selected in iteration t and r^t is the reward of that iteration
     - $p_{t+1}^i = (1-\beta) p_t^i + \beta p_{\max}$, for current best "arm" $i = i^*$
     - $p_{t+1}^i = (1-\beta) p_t^i + \beta p_{\min}$, for other arms $i \neq i^*$
     - "winner takes it all"

controls speed of selection adaptation

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 strengths are updated after every use of operator i

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

- 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)

UCB = Upper Confidence Bound

- 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:

    $$\text{expected reward} + \sqrt{c \log \frac{\sum_k n_{k,t}}{n_{j,t}}},$$

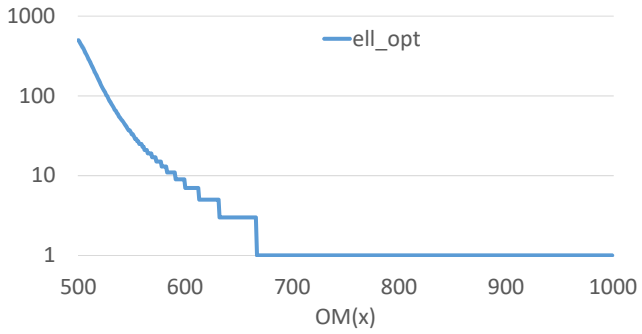
    where
    - $n_{k,t}$ is the number of times the k-th arm has been pulled in the first t iterations and
    - c is a parameter that allows to control the exploration likelihood (vs. exploitation, which is controlled by the first summand)
  - tuned and other variants of this algorithm exist, cf. [ACBF02] 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



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 (see previous chart)
- In [DDY16b] (GECCO'16) we analyzed this dependence and showed that an algorithm using the fitness-based step sizes that maximize the drift towards the target string z is almost (!) optimal
- 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, etc.)
- In [DDY16a] (PPSN'16) we showed how a learning-type mechanism automatically chooses parameter values that are close to optimal

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 ε, use a random $k \in [r]$ "exploring mut. strengths"
    - with prob. $1-\varepsilon$, use a k that maximized $v_{t-1}[k]$ "take the most efficient k"
  - Update the expected fitness gain estimations
- This strategy is called an ***ε-greedy selection*** in the machine learning literature

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

Main Results of [DDY16a]:

- The ε-greedy strategy uses in almost all iterations 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:
$$v_t[k] := \frac{\sum_{s=1}^{t} 1_{r_s=k}\,(1-\delta)^{t-s}\big(f(x_s)-f(x_{s-1})\big)}{\sum_{s=1}^{t} 1_{r_s=k}\,(1-\delta)^{t-s}}$$
- $1/\delta$: "*forgetting rate*", determines the decrease of the importance of older information. $1/\delta$is (roughly) the information half-life
- The "velocity" can be computed iteratively in constant time by introducing a new parameter $w_t[r] := \sum_{s=1}^{t} \mathbf{1}_{r_s=r}(1-\delta)^{t-s}$
- 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

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 5: Selected Additional Examples

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
or
"The Patchwork Problem" [KHE15]

Controlling Multiple Parameters

- Most EAs have several parameters
- Intuitively, there is no reason to not control more than one or even all of them
- A few works on controlling more than 1 parameter exists, cf. [KHE15]
- The problem how to best control several parameters is, however, widely open (given the non-conclusive state-of-the-art in controlling one parameter, this is perhaps not very surprising)

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 picking 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
  - Dynamic parameter choices can help to significantly improve the performance of your EA
  - Already quite simple mechanisms can be surprisingly efficient
  - Research on parameter control can be fun ☺
  - non-static parameter values should be the new standard in the field ☺
- As mentioned in the tutorial, a lot needs to be done to make this change happen
  - 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

Acknowledgments

- I am very grateful to Benjamin Doerr, Thomas Bäck, Markus Wagner, Jing Yang, Johannes Lengler, Dirk Sudholt, Pietro S. Oliveto, Johann Dreo, and the organizers and participants of the Dagstuhl seminars "Automated Algorithm Selection and Configuration" (16412) and "Theory of Randomized Optimization Heuristics" (17191) for many insightful discussions on parameter control mechanisms
- This work was supported by
  - the Paris Ile-de-France Region
  - a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH, in a joint call with Gaspard Monge Program for optimization, operations research and their interactions with data sciences.
- 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).

References

[AAD+13] Peyman Afshani, Manindra Agrawal, Benjamin Doerr, Carola Doerr, Kasper Green Larsen, and Kurt Mehlhorn. The query complexity of finding a hidden permutation. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2013.

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

[AM16] Aldeida Aleti and Irene Moser. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Computing Surveys*, 49:56:1–56:35, 2016.

[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.

[AMS+15] Carlos Ansótegui, Yuri Malitsky, Horst Samulowitz, Meinolf Sellmann, and Kevin Tierney. Model-based genetic algorithms for algorithm configuration. In *Proc. of International Conference on Artificial Intelligence (IJCAI'15)*, pages 733–739. AAAI Press, 2015.

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

[Bäc96] Thomas Bäck. *Evolutionary algorithms in theory and practice - evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.

[BBFKK10] Thomas Bartz-Beielstein, Oliver Flasch, Patrick Koch, and Wolfgang Konen. SPOT: A toolbox for interactive and automatic tuning in the R environment. In *Proc. of the 20. Workshop Computational Intelligence*, pages 264–273. Universitätsverlag Karlsruhe, 2010.

[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)*, pages 1343–1350. ACM, 2017.

[BD19] Nathan Buskulic and Carola Doerr. Maximizing drift is not optimal for solving onemax. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19, Companion)*. ACM, 2019. Full version available online at http://arxiv.org/abs/1904.07818. See also https://github.com/NathanBuskulic/OneMaxOptimal for more project data.

[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.

[CS09] Jorge Cervantes and Christopher R. Stephens. Limitations of existing mutation rate heuristics and how a rank GA overcomes them. *IEEE Transactions on Evolutionary Computation*, 13:369–397, 2009.

[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.

[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.

[DD18a] Benjamin Doerr and Carola Doerr. Optimal static and self-adjusting parameter choices for the $(1+(\lambda,\lambda))$ genetic algorithm. *Algorithmica*, 80:1658–1709, 2018.

[DD18b] Benjamin Doerr and Carola Doerr. Theory of parameter control mechanisms for discrete black-box optimization: Provable performance gains through dynamic parameter choices. In Benjamin Doerr and Frank Neumann, editors, *Theory of Randomized Search Heuristics in Discrete Search Spaces*. Springer, 2018. To appear. Available online at https://arxiv.org/abs/1804.05650.

[DD19] Nguyen Dang and Carola Doerr. Offspring population size matters when comparing evolutionary algorithms with self-adjusting mutation rates. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19)*. ACM, 2019. Full version available online at https://arxiv.org/abs/1904.04608.

[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.

[DDK18] Benjamin Doerr, Carola Doerr, and Timo Kötzing. Static and self-adjusting mutation strengths for multi-valued decision variables. *Algorithmica*, 80:1732–1768, 2018.

[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)*, pages 1351–1358. ACM, 2017.

[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 *LNCS*, pages 803–813. Springer, 2016.

[DL19] Benjamin Doerr and Carola Doerr Johannes Lengler. Self-adjusting mutation rates with provably optimal success rules. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19)*. ACM, 2019. Full version available online at http://arxiv.org/abs/1902.02588.

[DLOW18] Benjamin Doerr, Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. On the runtime analysis of selection hyper-heuristics with adaptive learning periods. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 1015–1022. ACM, 2018.

[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.

[Doe18] Benjamin Doerr. Better runtime guarantees via stochastic domination. In *Proc. of Evolutionary Computation in Combinatorial Optimization (EvoCOP'18)*, volume 10782 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2018. Full version available at http://arxiv.org/abs/1801.04487.

[DW18] Carola Doerr and Markus Wagner. On the effectiveness of simple success-based parameter selection mechanisms for two classical discrete black-box optimization benchmark problems. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 943–950. ACM, 2018.

[DWY18] Benjamin Doerr, Carsten Witt, and Jing Yang. Runtime analysis for self-adaptive mutation rates. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 1475–1482. ACM, 2018.

[DYvR+18] Carola Doerr, Furong Ye, Sander van Rijn, Hao Wang, and Thomas Bäck. Towards a theory-guided benchmarking suite for discrete black-box optimization heuristics: profiling $(1+\lambda)$ EA variants on onemax and leadingones. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 951–958. ACM, 2018.

[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. *Annals of Mathematics and Artificial Intelligence*, 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.

[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.

[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*, 78:587–609, 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.

[HHLB11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of Learning and Intelligent Optimization (LION'11)*, pages 507–523. Springer, 2011.

[HHLBS09] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[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.

[JDJW05] 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. *Journal of Discrete Algorithms*, 4:181–199, 2006.

[KHE15] Giorgos Karafotias, Mark 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.

[LDC+16] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[Len18] Johannes Lengler. A general dichotomy of evolutionary algorithms on monotone functions. In *Proc. of Parallel Problem Solving from Nature (PPSN'18)*, volume 11102 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2018.

[LLM07] Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.

[LOW17] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. On the runtime analysis of generalised selection hyper-heuristics for pseudo-Boolean optimisation. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 849–856. ACM, 2017. Extended version available at https://arxiv.org/abs/1801.07546.

[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.

[OLN09] Pietro Simone Oliveto, Per Kristian Lehre, and Frank Neumann. Theoretical analysis of rank-based mutation - combining exploration and exploitation. In *Proc. of Congress on Evolutionary Computation (CEC'09)*, pages 1455–1462. IEEE, 2009.

[RABD19] Anna Rodionova, Kirill Antonov, Arina Buzdalova, and Carola Doerr. Offspring population size matters when comparing evolutionary algorithms with self-adjusting mutation rates. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19)*. ACM, 2019. Full version available online at https://arxiv.org/abs/1904.08032.

[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.

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

[YDB19] Furong Ye, Carola Doerr, and Thomas Bäck. Interpolating Local and Global Search by Controlling the Variance of Standard Bit Mutation. In *Proc. Conference on Evolutionary Computation (CEC'19)*. IEEE, 2019. To appear. Available online at http://arxiv.org/abs/1901.05573.