Tutorial:
A Gentle Introduction to Theory (for Non-Theoreticians)

Benjamin Doerr
Laboratoire d'Informatique (LIX)
École Polytechnique, CNRS
Institut Polytechnique de Paris

lastname@lix.polytechnique.fr

http://gecco-2021.sigevo.org/

GECCO '21 Companion, Lille, France

ISBN 978-1-4503-8351-6/21/07 $15.00
DOI 10.1145/3449726.3461406

Instructor: Benjamin Doerr

- **Benjamin Doerr** is a full professor at the French École Polytechnique.
- He received his diploma (1998), PhD (2000) and habilitation (2005) in mathematics from the university of Kiel (Germany). His research area is the theory of both problem-specific algorithms and randomized search heuristics like evolutionary algorithms. Major contributions to the latter include runtime analyses for existing evolutionary algorithms, the determination of optimal parameter values, and the theory-guided design of novel operators, on-the-fly parameter choices, and whole new evolutionary algorithms.
- Together with Frank Neumann and Ingo Wegener, Benjamin Doerr founded the theory track at GECCO and served as its co-chair 2007-2009 and 2014. He is a member of the editorial boards of several journals, among them *Artificial Intelligence*, *Evolutionary Computation*, *Natural Computing*, and *Theoretical Computer Science*. Together with Frank Neumann, he edited the book *Theory of Evolutionary Computation – Recent Developments in Discrete Optimization* (Springer 2020).

This Tutorial: A *Real* Introduction to Theory

- GECCO, CEC, PPSN always had a good number of theory tutorials.
- They did a great job in educating the theory community.

- However, not much was offered for those attendees which
  - have little experience with theory,
  - but want to understand what the theory people are doing (and why).

- This is the target audience of this tutorial. We try to answer those questions which come before the classic theory tutorials.

Questions Answered in This Tutorial

- What is theory in evolutionary computation (EC)?
- Why do theory? How does it help us understanding EC?
- How do I read and interpret a theory result?
- What type of results can I expect from theory?
- What are current "hot topics" in the theory of EC?

Focus: EAs for Discrete Search Spaces

- In principle, all we say is valid for all subareas of theory.
- However, to not overload you with definitions and notation, we focus mostly on *classic evolutionary algorithms* for *discrete search spaces.*
- Hence we intentionally omit examples from
  - continuous optimization, e.g., CMA-ES, differential evolution, …
  - genetic programming, ant colony optimizers, swarm intelligence, …
    - exception: a discussion of the recent theory advances on estimation-of-distribution algorithms in part V.

The Most Important Point Before We Start

- If I'm saying things you don't understand or if you want to know more than what I had planned to discuss,
  don't be shy to ask questions at any time!
  - This is "your" tutorial and I want it to be as useful for you as possible.
- I'm trying to improve the tutorial each time I give it. For this, your feedback (positive and negative) is greatly appreciated!
  - → So talk to me after the tutorial, during the coffee breaks, social event, late-night beer drinking, … or send me an email.

in case you are in Paris

virtual

Structure of the Tutorial

- **Part I:** What is *Theory of EC?*
- **Part II:** A Guided Walk Through a Famous Theory Result
  - an illustrative example to convey the main messages of this tutorial
- **Part III:** How Theory Has Contributed to a Better Understanding of EAs
  - 3 ways how theory has an impact
- **Part IV:** How Theory Can Help YOU
- **Part V:** Current Hot Topics in the Theory of EAs
  - EDAs, dynamic&noisy optimization, dynamic/adaptive parameter choices
- **Part VI:** Concluding Remarks
- **Appendix:** glossary, references

Part I:

What is *Theory of EC*

- Definition: *theory of EC*
- What can you achieve with theoretical research?
- Comparison: theory vs. experiments

What Do We Mean With *Theory*?

- Definition (for this tutorial):
  By theory, we mean results proven with mathematical rigor.
- Mathematical rigor:
  - make precise the evolutionary algorithm (EA) you regard
  - make precise the problem you try to solve
  - formulate a precise statement how this EA solves this problem
  - prove this statement
- **Example:**
  Theorem: The (1+1) EA finds the optimum of the OneMax benchmark function $f: \{0,1\}^n \to \mathbb{R}; x \mapsto \sum_{i=1}^{n} x_i$ in an expected number of at most $en\ln(n)$ iterations.
  Proof: blah, blah, …

Other Notions of Theory

- **Theory:** Mathematically proven results
- **Experimentally guided theory:** Set up an artificial experiment to experimentally analyze a particular question.
  - Example: add a neutrality bit to two classic test functions, run a GA on these, and derive insight from the outcomes of the experiments.
- **Descriptive theory:** Try to describe/measure/quantify observations.
  - Example: fitness-distance correlation, schema theory, …
- **"Theories":** Unproven claims that (mis-)guide our thinking.
  - Example: building block hypothesis

Other Notions of Theory

- **Theory:** Mathematically proven results

==========<in this tutorial, we focus on the above>==========

- **Experimentally guided theory:** Set up an artificial experiment to experimentally analyze a particular question.
  - Example: add a neutrality bit to two classic test functions, run a GA on these, and derive insight from the outcomes of the experiments.
- **Descriptive theory:** Try to describe/measure/quantify observations.
  - Example: fitness-distance correlation, schema theory, …
- **"Theories":** Unproven claims that (mis-)guide our thinking.
  - Example: building block hypothesis

Why Do Theory? Because of the Results!

- Absolute guarantee that the result is correct (it is proven).
  - You can be sure.
  - Reviewers can check truly the correctness of results.
  - Readers can trust reviewers or, with moderate maths skills, check the correctness themselves.
- Many results can only be obtained by theory; e.g., because you make a statement on a very large or even infinite set:
  - all bit-strings of length n,
  - all TSP instances on n vertices,
  - all input sizes $n \in \mathbb{N}$,
  - all possible algorithms for a problem.

Why Do Theory? Because of the Approach!

- A proof (automatically) gives insight in
  - how things work (→ working principles of EC),
  - why the result is as it is.
- Self-correcting/self-guiding effect of proving:
  - When proving a result, you are automatically pointed to the questions that need more thought.
  - You see what exactly is the bottleneck for a result.
- Trigger for new ideas:
  - clarifying nature of mathematics,
  - playful nature of mathematicians.

Limitations of Theoretical Research

- Restricted scope: So far, mostly simple algorithms could be analyzed for simple optimization problems.
- Less precise results: Constants are not tight, or not explicit as in "$O(n^2)$" = "less than cn^2 for some unspecified constant c".
- Less specific results:
  - You obtain a (weaker) guarantee for *all problem instances,*
  - but not a stronger guarantee for *those instances which show up in your application.*
- Theory results can be very difficult to obtain: The proof might be short and easy to read, but finding it took long hours.
  - Usually, there is no generic way to the solution, but you need a completely new, clever idea.

Part II:

A Guided Walk Through a Famous Theory Result

We use a simple but famous theory result

- as an example for a non-trivial result
- to show how to read a theory result
- to explain the meaning of such a theoretical statement
- to illustrate what we just discussed

A Famous Result

Theorem: The (1+1) evolutionary algorithm finds the maximum of any linear function

$$f: \{0,1\}^n \to \mathbb{R}, (x_1, \dots, x_n) \mapsto \sum_{i=1}^{n} w_i x_i, \qquad w_1, \dots, w_n > 0,$$

in an expected number of $O(n \log n)$ iterations.

Reference:

[DJW02] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science, 276(1–2):51–81, 2002.

– Famous paper (500+ citations, maybe the most-cited pure EA theory paper)

– Famous problem (20+ papers working on exactly this problem, many highly useful methods were developed in trying to solve this problem)

Reading This Result

should be made precise in the paper to avoid any ambiguity

A mathematically proven result

(1+1) evolutionary algorithm to maximize $f: \{0,1\}^n \to \mathbb{R}$:
1. choose $x \in \{0,1\}^n$ uniformly at random
2. while not *terminate* do
3. generate y from x by flipping each bit independently with probability $1/n$ ("standard bit mutation")
4. if $f(y) \geq f(x)$ then $x := y$
5. output x

Theorem: The (1+1) evolutionary algorithm finds the maximum of any linear function

$$f: \{0,1\}^n \to \mathbb{R}, (x_1, \dots, x_n) \mapsto \sum_{i=1}^{n} w_i x_i, \qquad w_1, \dots, w_n > 0,$$

in an expected number of $O(n \log n)$ iterations.

a hidden all-quantifier: we claim the result for all $w_1, \dots, w_n > 0$

at most $Cn \ln n$ for some unspecified constant C

performance measure: number of iterations or fitness evaluations, but not runtime in seconds

Why is This a Good Result?

- Gives a *proven* performance guarantee
- General: a statement for *all* linear functions in *all* dimensions n
- Non-trivial
- Surprising
- Provides insight in how EAs work

→ more on these 3 items on the next slides

Non-Trivial:

Non-Trivial: Hard to Prove & Hard to Explain Why it Should be True

- **Hard to prove**
  - 7 pages complicated maths proof in [DJW02].
  - We can do better now, but only because we developed deep analysis techniques (drift analysis).
- **No "easy" explanation**
  - *monotonicity*: flipping a 0 to a 1 always increases the fitness
    - Are monotonic functions easy to optimize for a EAs (because you only need to collect 1s)?
    - No! Exponential runtimes can occur [DJS+13, LS18].
  - *separability*: a linear function can be written as a sum of functions f_i such that the f_i depend on disjoint sets of bits
    - Is the optimization time of such a sum small?
    - No! The f_i can interact badly [DSW13].

Surprising: Same Runtime For Very Different Fitness Landscapes

- **Example 1:** OneMax, the function counting the number of 1s in a string, $\mathrm{OM}: \{0,1\}^n \to \mathbb{R}, (x_1, \dots, x_n) \mapsto \sum_{i=1}^n x_i$:
  - unique global maximum at $(1, \dots, 1)$
  - perfect fitness distance correlation: if a search point has higher fitness, then it is closer to the global optimum.
- **Example 2:** BinaryValue (BinVal for short), the function mapping a bit-string to the number it represents in binary $\mathrm{BV}: \{0,1\}^n \to \mathbb{R}, (x_1, \dots, x_n) \mapsto \sum_{i=1}^n 2^{n-i} x_i$:
  - unique global maximum at $(1, \dots, 1)$
  - very low fitness-distance correlation:
    - $\mathrm{BV}(10 \dots 0) = 2^{n-1}$, distance to optimum is $n-1$,
    - $\mathrm{BV}(01 \dots 1) = 2^{n-1} - 1$, distance to optimum is 1.

Insight in Working Principles

- Insight from the result:
  - Even if there is a low fitness-distance correlation (as is the case for the BinVal function), EAs can be very efficient optimizers.
- Insight from the proof:
  - The Hamming distance $H(x, x^*)$ of x to the optimum x^* measures very well the quality of the search point x:
  - The expected number $E[T_x]$ of iterations to find the optimum starting from x satisfies

$$en \ln(H(x,x^*)) - O(n) \leq E[T_x] \leq 4en \ln\big(2eH(x,x^*)\big)$$

independent of f.

DJW02: Droste, Jansen, Wegener
DJW12: Doerr, Johannsen, Winzen

A Glimpse on a Modern Proof

- **Theorem [DJW12]:** For all problem sizes n and all linear functions $f: \{0,1\}^n \to \mathbb{R}$ with $f(x) = w_1 x_1 + \dots + w_n x_n$ the (1+1) EA finds the optimum x^* of f in an expected number of at most $4en \ln(2en)$ iterations.
- 1st proof idea: Without loss, we can assume that $w_1 \geq w_2 \geq \dots \geq w_n > 0$.
- 2nd proof idea: Regard an artificial fitness measure!
  - Define $\tilde{f}(x) = \sum_{i=1}^n \left(2 - \frac{i-1}{n}\right) x_i$ "artificial weights" from 2 down to $1 + \frac{1}{n}$
  - Key lemma: Consider the (1+1) EA optimizing the original f. Assume that some iteration starts with the search point x and ends with the random search point x'. Then

$$E\big[\tilde{f}(x^*) - \tilde{f}(x')\big] \leq \left(1 - \frac{1}{4en}\right)\big(\tilde{f}(x^*) - \tilde{f}(x)\big).$$

  → expected artificial fitness distance reduces by a factor of $\left(1 - \frac{1}{4en}\right)$.
- 3rd proof idea: Multiplicative drift theorem translates this expected progress w.r.t. the artificial fitness into a runtime bound.
  - Roughly: the expected runtime is at most the number of iterations needed to get the expected artificial fitness distance below one.

Multiplicative Drift Theorem

- **Theorem [DJW12]:** Let $X_0, X_1, X_2, \dots$ be a sequence of random variables taking values in the set $\{0\} \cup [1, \infty)$. Let $\delta > 0$. Assume that for all $t \in \mathbb{N}$, we have

$$E[X_{t+1} | X_t = x] \leq (1-\delta)\, x.$$

  Let $T := \min\{t \in \mathbb{N} \mid X_t = 0\}$. Then

$$E[T | X_0 = x] \leq \frac{1 + \ln x}{\delta}.$$

  "Drift analysis": Translate *expected progress* into *expected (run-)time*
- On the previous slide, this theorem was used with
  - $\delta = 1/4en$,
  - $X_t = \tilde{f}(x^*) - \tilde{f}\big(x^{(t)}\big)$,
  - and the estimate $X_0 \leq 2n$.
- **Bibliographical notes:** Artificial fitness functions very similar to this $\tilde{f}$ were already used in Droste, Jansen, and Wegener [DJW02] (conference version [DJW98b]). Drift analysis ("translating progress into runtime") was introduced to the field by He and Yao [HY01] to give a simpler proof of the [DJW02] result. A different approach was given by Jägersküpper [Jäg08]. The multiplicative drift theorem by D., Johannsen, and Winzen [DJW12] (conference version [DJW10]) proves the [DJW02] result in one page and is one of the most-used tools today.

Limitations of the Linear Functions Result

- An unrealistically simple EA: the (1+1) EA.
- Linear functions are "trivial" artificial test function.
- Not a precise result, but
  - only $O(n \log n)$ in [DJW02]
  - or a most likely significantly too large constant in the [DJW12] result.
- Two replies (details on the following slides):
  - Despite these limitations, we gain insight.
  - The 2002-results was the start, now we know much more.

Limitation 1: Only the Simple (1+1) EA

- Insight: Using nothing else than standard bit mutation is enough to optimize problems with low fitness-distance correlation.
- Newer Result: The (1+λ) EA optimizes any linear function in expected time (= number of fitness evaluations)

$$O(n \log n + \lambda n).$$

  This bound is sharp for BinVal, but not for OneMax, where the expected optimization time is

$$O\left(n \log n + \lambda n \frac{\log \text{lo}}{\log \lambda}\right).$$

  → Not all linear functions have the same optimization time [DK15]!
- We are optimistic that we will make progress towards more complicated EAs. Known open problems include, e.g., how crossover-based algorithms and ant colony optimizers optimize linear functions.

Limitation 2: Only Linear Functions

- Insight: Linear functions are easy, monotonic functions can be difficult → some understanding which problems are easy and hard for EAs.
- Newer runtime analyses for the (1+1) EA (and some other algorithms):
  - Eulerian cycles [Neu04, DHN07, DKS07, DJ07],
  - shortest paths [STW04, DHK07, BBD+09],
  - minimum spanning trees [NW07, DJ10, Wit14],
  - and many other poly-time optimization problems.
- We also have some results on approximate solutions for NP-complete problems like partition [Wit05], vertex cover [FHH+09, OHY09], maximum cliques [Sto06], graph coloring [SZ10, BS19].
- We have some results on dynamic and noisy optimization (→ part V).

Limitation 3: $O(n \log n)$

- Insight: Linear functions are easy for the (1+1) EA.
  - For this insight, a rough result like $O(n \log n)$ is enough.
- Newer result [Wit13]: The exp. runtime of the (1+1) EA on any linear function is $en \ln n + O(n)$, that is, at most $en \ln n + Cn$ for some constant C.
  - Asymptotic result, but the asymptotics are only in a lower order term.
  - [Wit13] also has a non-asymptotic result, but it is harder to digest:

Theorem 4.1. *On any linear function on n variables, the optimization time of the (1+1) EA with mutation probability $0 < p < 1$ is at most*

$$(1-p)^{1-n}\left(\frac{n\alpha^2(1-p)^{1-n}}{\alpha-1} + \frac{\alpha}{\alpha-1}\frac{\ln(1/p)+(n-1)\ln(1-p)+r}{p}\right) =: b(r),$$

with probability at least $1-e^{-r}$ for any $r > 0$, and it is at most $b(1)$ in expectation, where $\alpha > 1$ can be chosen arbitrarily (even depending on n).

Summary "Guided Tour"

- We have seen one of the most influential theory results:
  The (1+1) EA optimizes any linear function in $O(n \log n)$ iterations.
- We have seen how to read and understand such a result.
- We have seen why this result is important:
  - non-trivial and surprising,
  - gives insights in how EAs work, and
  - spurred the development of many important tools (e.g., drift analysis).
- We have discussed the limitations of this theory result.

Part III:

How Theory Can Help *Understanding* and *Designing* EAs

1. Debunk misconceptions
2. Help choosing the right parameters, representations, operators, and algorithms
3. Invent new representations, operators, and algorithms

Contribution 1: Debunk Misconceptions

- When working with EAs, it is easy to conjecture some general rule from observations, but without theory it is hard to distinguish between "we often observe" and "it is true that".
- Reason: It is often hard to falsify a conjecture experimentally.
  - The conjecture might be true "often enough", but not in general.
- Danger: Misconceptions prevail in the EA community and misguide the future development of EAs.
- 2 (light) examples on the following slides

Misconception 1: Functions Without Local Optima are Easy to Optimize

- A function $f: \{0,1\}^n \to \mathbb{R}$ has *no local optima* if each non-optimal search point has a neighbor with better fitness. *"unimodal function"*
  - → If $f(x)$ is not optimal, then by flipping a single bit of x you can reach a better solution.
- Misconception: Such functions are easy to optimize…
  - "because all you need is flipping single bits".
- Truth: There are unimodal functions $f: \{0,1\}^n \to \mathbb{R}$ such that all reasonable EAs with high probability need super-polynomial time to find a reasonable solution [HGD94,Rud96,DJW98a].
- Reason: yes, it is easy to find a better neighbor if you're not optimal yet, but you may need to do this an exponential number of times because all improving paths to the optimum are that long

Misconception 2: Monotonic Functions are Easy to Optimize for EAs

- A function $f: \{0,1\}^n \to \mathbb{R}$ is *monotonically strictly increasing* (monotonic) if the fitness increases whenever you flip a 0-bit into 1.
  - strong version of "no local optima": *each* neighbor with additional ones is better
- Misconception: Such functions are easy to optimize for standard EAs…
  - because already simple hill-climbers flipping single bits (e.g., randomized local search) do the job in time $O(n \log n)$.
- Truth: There are (many) monotonic functions such that with high probability the (1+1) EA with mutation probability $16/n$ needs exponential time to find the optimum [DJS+13].
  - The $16/n$ can be lowered to $\approx 2.13/n$ [LS18].
  - Same result for many mutation-based algorithms [Len20].
  - For any $c > 0$ there is a $\mu = O(n)$ and a monotonic f such that the (1+μ) EA with mutation rate c/n needs super-polynomial time to optimize f [LZ19].

Summary Misconceptions

- Intuitive reasoning or experimental observations can lead to wrong beliefs.
- It is hard to falsify them experimentally, because
  - counter-examples may be rare (so random search does not find them),
  - counter-examples may have an unexpected structure.
- There is nothing wrong with keeping these beliefs as "rules of thumb", but it is important to know what is a rule of thumb and what is really the truth.
  - Theory is the right tool for this!

Contribution 2: Help With Design Choices

- When designing an EA, you have to choose between a huge number of design alternatives: the basic algorithm, the operators and representations, the parameters, ….
- Theory can guide you with deep and reliable analyses of scenarios similar to yours.
  - The question "what is a similar scenario" remains, but you have the same problem when looking for advice from experimental research.
- Examples:
  - use of fitness-proportionate selection
  - representations in graph problems

→ more on these 2 on the next slides

  - use of crossover: [JW02, SW04, FW04, FW05, JW05, Sud05, WJ07, RWP08, DT09, NT10, LY11, KST11, DJK+11, DHK12a, DJK+13, DFK+16, Sud17, DFK+18, CO18, CO20, OSW20,Sut21]
  - parameters: [Müh92, Bäc93, GKS99, JW00, Prü04, JJW05, Wit06, JS07, BDN10, Leh10, Leh11, LY12, Sud13, Wit13, RS14, DK15, GW17, DLMN17, ADFH18, ADY19, AD20, BBD21a, Doe21]

Design Choices: Fitness-Proportionate Selection

- Theorem [OW15]: When the Simple GA (Goldberg [Gol89]) with a population size $n^{0.2499}$ or less optimizes the OneMax test function $f: \{0,1\}^n \to \mathbb{R}; x \mapsto x_1 + \cdots + x_n$, then in any polynomial number of iterations it does not find an individual that is 1% better than a random individual.
- Interpretation: If fitness-proportionate has difficulties already on OneMax, use it with caution! Similar results [HJKN08, NOW09]

Algorithm (SGA)

1. Create a parent population P consisting of μ individuals chosen uniformly at random.
2. $C := \emptyset$.
3. While $|C| < \mu$ do
   - **Fitness proportional selection:** Select two individuals x' and x'' from P according to fitness-proportional selection with replacement.
   - **Uniform crossover:** Create an offspring x by setting each bit $x(i) = x'(i)$ with probability $1/2$ and $x(i) = x''(i)$ otherwise, for $1 \le i \le n$.
   - **Standard Bit Mutation:** Flip each bit $x(i)$ of x with probability $1/n$, for $1 \le i \le n$.
   - $C := C \cup \{x\}$.
4. Set $P := C$ and go to 2.

Design Choices: Representations

- Several theoretical works on shortest path problems [STW04, DHK07, BBD+09]. All use a vertex-based representation:
  - each vertex points to its predecessor in the path
  - mutation: rewire a random vertex to a random neighbor
- [DJ10]: How about an edge-based representation? (typical theory-driven curiosity)
  - individuals are set of edges (forming reasonable paths)
  - mutation: add a random edge (and delete the one made obsolete)
- **Result:** All previous algorithms become faster by a factor of $\approx \frac{|V|^2}{|E|}$
  - [JOZ13]: edge-based representation also preferable for vertex cover
- Interpretation: While there is no guarantee for success, it may be useful to think of an edge-based representation for graph-algorithmic problems

Summary Design Choices

- By rigorously analyzing simplified situations, theory can suggest
  - which algorithm to use,
  - which representation to use,
  - which operators to use,
  - how to choose parameters.
- As with all particular research results, the question remains how representative such a result is for the general usage of EAs.

Contribution 3: Invent New Operators and Algorithms

- Theory can also, both via the deep understanding gained from proofs and by "theory-driven curiosity" invent new operators and algorithms.
- Example: What is the right way to do mutation [DLMN17]?
- Outline (of the next 10+ slides):
  - What is "known" about mutation
  - A thorough analysis how simple EAs optimize the jump benchmark
  - Some unexpected conclusions [best-paper award in the GECCO 2017 Genetic Algorithms track]
- 2nd example [not shown]: Design of the $(1 + (\lambda, \lambda))$ GA based on black-box complexity insight [DDE13, GP14, DDE15, DD15a, DD15b, Doe16, BD17, DD18, KAD19, ADK19, BB19, ABD20a, ADK20, ABD20b, AD20, BB20, FS20, ABD21]

General Belief on Mutation

- **Note:** We only deal with *bit-string representations*, that is, the search space is $\{0,1\}^n$ for some n.
  - [Similar results hold for other discrete search spaces]
- **General belief:** The right way of doing mutation is *standard bit mutation*, that is, flipping each bit independently with some probability p ("mut. rate").
  - Global operator: from any parent you can generate any offspring (possibly with very small probability).
    → Algorithms cannot get stuck forever in a local optimum.
- **General recommendation:** Use a small mutation rate like $p = 1/n$. See, e.g., [Bäc96, BFM97, Och02].

Informal Justifications for $p = 1/n$

- Imitate local search / hill-climbing: A mutation rate of $1/n$ maximizes the probability to flip a single bit.
  - Reducing the rate by a factor of c reduces this prob. by a factor of $\Theta(c)$.
  - Increasing the rate by a factor of c reduces this prob. by a factor of $e^{\Theta(c)}$.
- Mutation is destructive: If your current search point x has a Hamming distance $H(x, x^*)$ of less than $n/2$ from the optimum x^*, then the offspring y has (in expectation) a larger Hamming distance and this increase is proportional to p:
  - $E[H(y, x^*)] = H(x, x^*) + p(n - 2H(x, x^*))$

$O(c)$ = at most γc for some constant γ
$\Omega(c)$ = at least δc for some constant $\delta > 0$
$\Theta(c)$ = both $O(c)$ and $\Omega(c)$

Proven Results Supporting $p = 1/n$

- Optimal mutation rates for (1+1) EA:
  - $\approx \frac{1}{n}$ for OneMax [Müh92; Bäc93, GKS99]
  - $\approx \frac{1.59}{n}$ for LeadingOnes [BDN10]
  - $\approx \frac{1}{n}$ for all linear functions [Wit13]
  - monotonic functions [Jan07, DJSWZ13, LS18, LMS19]:
    - $p = \frac{c}{n}, 0 < c < 1$, gives a $\Theta(n \log n)$ expected runtime on all monotonic functions with unique optimum,
    - $p = \frac{c}{n}, c \in [1, 1+\varepsilon]$ for some $\varepsilon > 0$ gives $O(n \log^2 n)$,
    - $p \geq \frac{2.13\ldots}{n}$ gives an exponential runtime on some monotonic functions.

Theory supports using standard bit mutation with mutation rate around $1/n$

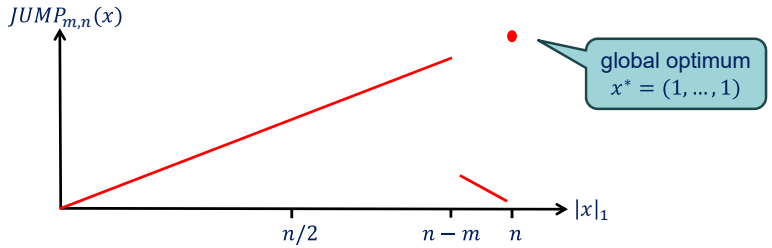
- When $\lambda \leq \ln n$, then the optimal mutation rate for the $(1+\lambda)$ EA optimizing OneMax is $\approx \frac{1}{n}$ [GW17].

Really?

- Can we really say that $1/n$ is good (at least "usually")?
- More provocative: Can we really say that *standard bit mutation* is the right way of doing mutation?
- Note: All results regard easy unimodal optimization problems.
  - OneMax, LeadingOnes, linear functions, monotonic functions.
  - → Flipping single bits is a very good way of making progress
- Let's look at an example with local optima…

Question: What is the Best Mutation Rate for the (1+1) EA on the Jump Functions Benchmark?

- $JUMP_{m,n}$: fitness of an n-bit string x is the number $|x|_1$ of ones, except if $|x|_1 \in \{n-m+1, \ldots, n-1\}$, then $f(x) = n - |x|_1$ [DJW02]



- Novelty (for a theoretical analysis of the mutation rate): There are non-trivial *local optima*: all $x \in \{0,1\}^n$ with $|x|_1 = n - m$.

Runtime Analysis

- Let $T_p = T_p(m,n)$ denote the expected optimization time of the (1+1) EA optimizing $JUMP_{m,n}$ with mutation rate $p \leq 1/2$.
- Theorem: For all $2 \leq m \leq n/2$ and $p \leq 1/2$,

$$(1-o(1))\frac{1}{p^m(1-p)^{n-m}} \leq T_p(m,n) \leq \frac{1}{p^m(1-p)^{n-m}} + \frac{2\ln\frac{n}{m}}{p(1-p)^{n-1}}.$$

- Let $T_{opt} = T_{opt}(m,n) := \inf\{T_p(m,n) \mid p \in [0,1/2]\}$.
- Theorem: If $m \leq n/4$, then $T_{opt} = T_{m/n}\,(1 \pm o(1)) \approx \left(\frac{e}{m}\right)^m T_{1/n}$ and $p = m/n$ is essentially the only optimal mutation rate.
- **→ The right mutation rate is much higher than the usual $1/n$ and it gives a huge speed-up!**

Missing the Optimal Mutation Rate

- Theorem: If $p \geq (1+\varepsilon)(m/n)$ or $p \leq (1-\varepsilon)(m/n)$, then

$$T_p(m,n) \geq \frac{1}{6}\exp\left(\frac{m\,\varepsilon^2}{5}\right) T_{opt}(m,n).$$

- In simple words: m/n is essentially the optimal mutation rate, but a small deviation increases the runtime massively.
- → Dilemma: To find the right mutation rate, you need to know "the m", that is, how many bits you need to flip to leave the local optimum ☹.
- Math. reason for the dilemma: When flipping bits independently at random (standard bit mutation), the Hamming distance $H(x,y)$ of parent x and offspring y is strongly concentrated around the mean.
  - → Exponential tails of the binomial distribution
- **→ Maybe standard bit mutation is not the right thing to do?**

From This Analysis to a New Mutation Operator

- Recap: What do we need?
  - No strong concentration of $H(x,y)$
  - Larger numbers of bits flip with reasonable probability
  - 1-bit flips occur with constant probability (→ easy hill-climbing)
- Solution: *Heavy-tailed mutation* (with parameter $\beta > 1$, say $\beta = 1.5$).
  - choose $\alpha \in \{1,2,\dots,n/2\}$ randomly with $\Pr[\alpha] \sim \alpha^{-\beta}$ [power-law].
  - perform standard bit mutation with mutation rate α/n.
- Some maths:
  - The probability to flip k bits is $\Theta(k^{-\beta})$. → No exponential tails ☺
  - $\Pr[H(x,y)=1] = \Theta(1)$, e.g., ≈32% for $\beta = 1.5$ (≈37% for classic mut.)

Note: Random mut-rates have been used before in theory, but not heavy-tailed and only for special purposes (unknown solution length [DDK17], higher arities [DDK18])

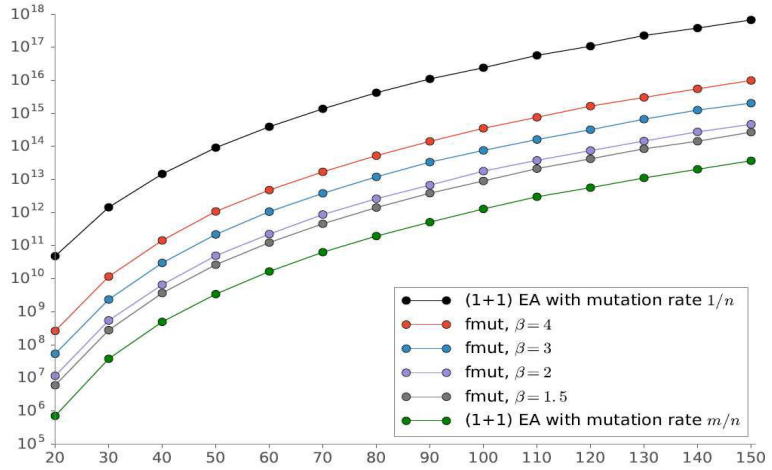
Heavy-tailed Mutation: Results

- Theorem: The (1+1) EA with heavy-tailed mutation ($\beta > 1$) has an expected optimization time on $JUMP_{m,n}$ of

$$O\left(m^{\beta-0.5}\,T_{opt}(m,n)\right).$$

- **This one algorithm for all m is only an $O(m^{\beta-0.5})$ factor slower than the EA using the optimal mutation rate (depending on m)!**
  - **"One size fits all" (apart from a small polynomial factor).**
  - Compared to the classic EA, this is a speed-up by a factor of $m^{\Theta(m)}$.
- Lower bound (not important, but beautiful (also the proof)): The loss of slightly more than $\Theta(m^{0.5})$ – by taking $\beta = 1+\varepsilon$ – is unavoidable:
  - For n sufficiently large, any distribution D_n on the mutation rates in $[0,1/2]$ has an $m \in [2..n/2]$ such that $T_{D_n}(m,n) \geq \sqrt{m}\,T_{opt}(m,n)$.

Experiments (m=8, n=20..150)



Runtime of the (1+1) EA on $JUMP_{8,n}$ (average over 1000 runs). To allow this number of experiments, the runs where stopped once the local optimum was reached and the remaining runtime was sampled directly from the geometric distribution describing this waiting time.

Beyond Jump Functions

- Example (maximum matching): Let G be an undirected graph having n edges. A matching is a set of non-intersecting edges. Let OPT be the size of a maximum matching. Let $m \in \mathbb{N}$ be constant and $\varepsilon = \frac{2}{2m+}$.
  - The classic (1+1) EA finds a matching of size $\frac{OPT}{1+\varepsilon}$ in an expected number of at most $T_{n,\varepsilon}$ iterations, where $T_{n,\varepsilon}$ is some number in $\Theta(n^{2m+2})$. [GW03]
  - The (1+1) EA with heavy-tailed mutation does the same in expected time of at most $\boldsymbol{(1+o(1))\, e\, \zeta(\beta) \left(\frac{e}{m}\right)^m m^{\beta-0.5}\, T_{n,\varepsilon}}$.

Riemann zeta function: $\zeta(\beta) < 2.62$ for $\beta \geq 1.5$

- 2nd example: Vertex cover in bipartite graphs (details omitted).

Performance in Classic Results

- Since the heavy-tailed mutation operator flips any constant number of bits with constant probability, many classic results for the standard (1+1) EA remain valid (apart from constant factor changes):
  - $O(n \log n)$ runtime on OneMax
  - $O(n^2)$ runtime on LeadingOnes
  - $O\big(m^2 \log(nw_{\max})\big)$ runtime on MinimumSpanningTree [NW07]
  - and many others…
- The largest expected runtime that can occur on an $f: \{0,1\}^n \to \mathbb{R}$ is …
  - $\Theta(n^n)$ for the classic (1+1) EA: Trap function [DJW02], minimum makespan scheduling [Wit05]
  - $O(n^\beta\, 2^n)$ for the heavy-tailed (1+1) EA

Working Principle of Heavy-Tailed Mutation

- Reduce the probability of a 1-bit flip slightly (say from 37% to 32%)
- Distribute this free probability mass in a power-law fashion on all other k-bit flips
  → increases the prob. for a k-bit flip from roughly $\frac{1}{e \cdot k!}$ to roughly $k^{-\beta}$
  → reduces the waiting time for a k-bit flip from $e \cdot k!$ to k^β
- This redistribution of probability mass is a good deal, because we usually spend much more time on finding a good multi-bit flip
  - $JUMP_{m,n}$: spend $\Theta(n \log n)$ time on all 1-bit flips, but $\binom{n}{m}$ time to find the one necessary m-bit flip
- These elementary observations are a good reason to believe that heavy-tailed mutation is advantageous for a wide range of multi-modal problems.
  - Other theory works: [FQW18, FGQW18a, FGQW18b, WQT18, ABD20a, ABD20b, DZ21, ABD21]

Choose all 3 parameters of an algorithm heavy-tailed and get essentially the performance of optimal parameters.

Heavy-Tailed → "Fast"

- Heavy-tailed mutation has been experimented with in *continuous optimization* (with mixed results as far as I understand):
  - Simulated annealing [Szu, Hartley '87]
  - Evolutionary programming [Yao, Lui, Lin '99]
  - Evolution strategies [Yao, Lui '97; Hansen, Gemperle, Auger, Koumoutsakos '06; Schaul, Glasmachers, Schmidthuber '11]
  - Estimation of distribution algorithms [Posik '09, '10]
- Algorithms using heavy-tailed mutation were called *fast* by their inventors, e.g., *fast simulated annealing*.
  - → We propose to call our mutation *fast mutation* and the resulting EAs *fast*, e.g., $(1+1)\ FEA_\beta$.

Summary: Fast Mutation on $\{0,1\}^n$ – A Theory-Guided Invention

- By rigorously analyzing the performance of a simple mutation-based EA on the non-unimodal JUMP fitness landscape, we observe that
  - higher mutation rates are useful to leave local optima
  - standard bit mutation with a fixed rate is sub-optimal on most problems
- Solution: Use standard bit mutation, but with a random mutation rate sampled from a power-law distribution
  - $m^{\Theta(m)}$ factor speed-up for $JUMP_{m,n}$
  - $m^{\Theta(m)}$ factor improvement of the runtime guarantee for max. matching
  - first promising experimental results
- Big question: Will this work in practice and will practitioners use it?
  - First results are very promising

Summary Part 3

Theory has contributed to the understanding and use of EAs by

- debunking misbeliefs (drawing a clear line between rules of thumb and proven fact)
  - e.g., "no local optima" and "monotonic" do not mean "easy"
- giving hints how to choose parameters, representations, operators, and algorithms
  - e.g., if fitness-proportionate selection with comma selection cannot even optimize OneMax, maybe it is not a good combination
- inventing new representations, operators, and algorithms: this is fueled by the deep understanding gained in theoretical analyses and "theory-driven curiosity"
  - e.g., if leaving local optima generally needs more bits to be flipped, then we need a mutation operator that does so sufficiently often → heavy-tailed mutation

Part IV:

How Theory Can Help YOU

How Theory Can Help YOU

- Message of this talk so far: Theory people can produce mathematical analyses and from these gain insights that are useful also outside theory.
- Two ways how you can profit from theory:
  - Try to read some theory works and (at least) understand their meaning for the general use of EAs
    → could be difficult
  - Try to imitate the theory approach (without proving everything)
    → could be easy ☺
    - Next few slides: How you could have invented the heavy-tailed mutation operator with theory-style thinking

An Example of *Theory-Style Thinking*

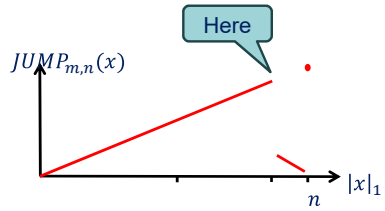
- Problem: You run your favorite evolutionary algorithm on your favorite problem and you feel that it takes too long to leave local optima.
- You try without success all your tricks:
  - parameter tuning
  - landscape analysis
  - asking colleagues that are true experts in experimental work
  - etc.
- but nothing really solves the problem.
- You're so desperate that you try theory-style thinking…

1st Step: Take a Really Simple Example Situation

- Really simple example situation (that hopefully still is representative for your problem of leaving local optima):
  - You take a very simple optimization problem in which every reasonable heuristic gets stuck in a local optimum → jump function
  - You take the most simple evolutionary algorithm you know
    → the (1+1) EA with mutation rate p
  - You only look at the problem of leaving the local optimum (and not at the whole runtime)
- Note: If you later see that this is too simple and not helpful, you can still make it more complex later. But don't be shy to start off really simple!

2nd Step: Analyze Your Example Precisely and in Full Generality

- Simple example situation: The (1+1) EA optimizes a jump function and is already in the local optimum.



- Question: How long does it take to leave the local optimum?
- What is the probability to generate an offspring better than the local opt.?
- Local optimum: any bit-string with $n-m$ ones and m zeroes
- To leave this, you have to flip the m missing bits and not flip the other bits
- Probability for this $P(p) = p^m(1-p)^{n-m}$

Full generality:
A formula for all n, m, p

3rd Step: Generate Useful Data

- We plot the function $p \mapsto P(p) = p^m(1-p)^{n-m}$ for *interesting values* of n, m, p:
  - a moderate problems size $n = 50$, a small jump size $m = 4$;
  - for the mutation rate p, we recall that the standard choice is $1/n$. So let's use the scaling $p = \alpha/n$ and take $\alpha \in [0.05,20]$:

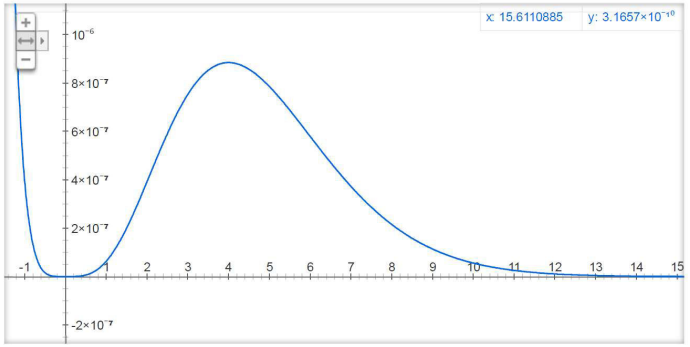
$$P(\alpha) = \left(\frac{\alpha}{50}\right)^4 \left(1 - \frac{\alpha}{50}\right)^{46}, \alpha \in [0.05,20]$$

- [type "y = (x/50)^4*(1-x/50)^46" into Google]

3rd Step: Generate Useful Data

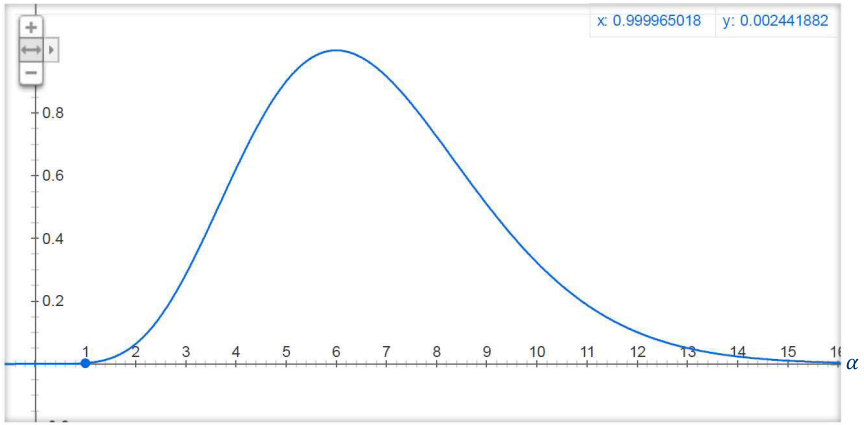
- We plot the function $p \mapsto P(p) = p^m(1-p)^{n-m}$
  - a moderate problems size $n = 50$, a small jump size $m = 4$;
  - for the mutation rate p, we recall that the standard choice is $1/n$. So let's use the scaling $p = \alpha/n$ and take $\alpha \in [0.05,20]$:

Graph for (x/50)^4*(1-x/50)^46



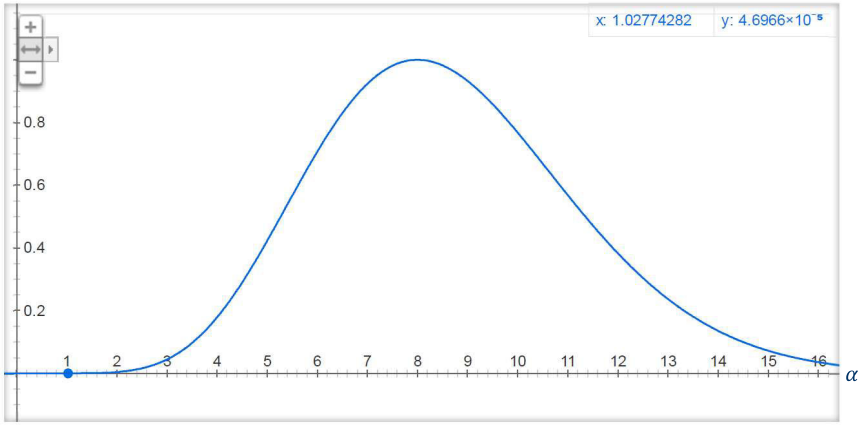
$$P(\alpha) = \left(\frac{\alpha}{n}\right)^m \left(1 - \frac{\alpha}{n}\right)^{n-m}, n = 50, \boldsymbol{m = 6}$$

Graph for (x/50)^6*(1-x/50)^44/((6/50)^6*(1-6/50)^44)



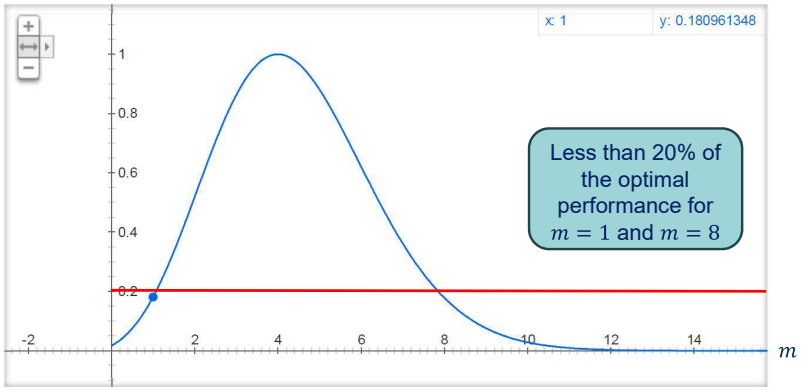
$$P(\alpha) = \left(\frac{\alpha}{n}\right)^m \left(1 - \frac{\alpha}{n}\right)^{n-m}, n = 50, \boldsymbol{m = 8}$$

Graph for (x/50)^8*(1-x/50)^42/((8/50)^8*(1-8/50)^42)



Inverse Plot: Loss From Taking Rate $4/n$ Instead of the Optimal Rate m/n

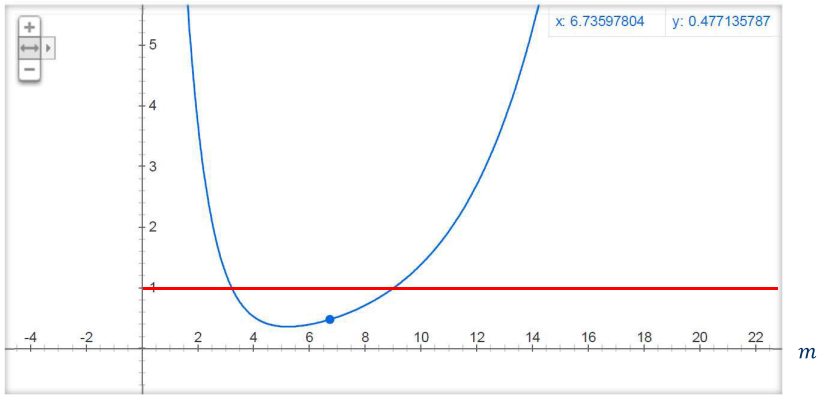
Graph for (4/50)^x*(1-4/50)^(50-x)/((x/50)^x*(1-x/50)^(50-x))



4th Step: Interpret the Data and Find a Solution

- The plots clearly show:
  - The classic mutation rate of $1/n$ is highly suboptimal:
    - e.g., a factor-500 performance loss for $m = 6$
  - There is no "right" mutation rate: Each rate α/n is good for values of m that are close to α only
    - e.g., $4/n$ is perfect for $m = 4$, but gives only 20% of the optimal performance for $m = 1$ and $m = 8$
- Solution attempt: "average" over different mutation rates!
  - e.g., take rate $2/n$ and $8/n$ each with probability 50%

$P(\text{average}(2/n, 8/n))/P(6/n)$



- Averaging gives significant speed-ups for $m \notin [3,9]$
- Next steps (omitted here): Optimize this averaging strategy

Summary: Theory-Style Thinking

- Step 1: Choose a really, really simple example situation.
- Step 2: Analyze this example precisely and in full generality.
  → Mathematical formula
- Step 3: Use the formula to cheaply generate very trustworthy data for any parameter values you want.
- Step 4: Interpret the data, find a solution.

Part V:

Current Topics of Interest in the Theory of EC

- Populations
- Estimation-of-distribution algorithms (EDAs)
- Dynamic and noisy optimization
- Dynamic/adaptive parameter choices
- Fine-grained runtime analysis: fixed budget/target, parameterized complexity

Hot Topic 1: Populations

- While most EAs in practice use non-trivial populations, EA theory has not been very successful in understanding why this is good (but some interesting results exist).

- Elitist mutation-based algorithms:
  - Larger offspring population size [JJW05, DK15, GW17, GW18]:
    - Allow parallel implementations (faster).
    - Usually no speed-up w.r.t. the total number of fitness evaluations.
    - Research question: Up to which pop. size you have a linear speed-up, that is, the total runtime does not increase?
  - Larger parent population size: Rather slows down things, but by surprisingly little [Wit06, ADFH18, AD20].
  - Both can provably give robustness against noise and dynamic changes of the problem [JS05, GK16, LW16, DJL17, LM20].

Populations (2)

- Non-elitist mutation-based algorithms: Need sufficiently large populations [JS07, NOW09, Leh10, Leh11, RS14, DL16a, CDEL18, DK19, Doe20b]
  - Small offspring pop: you lose good solutions too quickly and cannot really optimize (exponential runtimes).
  - Large offspring pop: you usually generate a copy of a good parent and thus imitate an elitist algorithm.
  - Inside the phase transition: strange things happen [ADY19].
  - Problem: Not too much argument for non-elitism in theory so far!
    Example: for no choice of the population sizes, the (μ,λ) EA shows an interesting speed-up over the $(\mu+\lambda)$ EA on jump functions [Doe20a]

- Large parent population plus diversity mechanism: The diversity mechanism can force the population to spread out, this can aid leaving local optima [FHN07, Sto08, FHN09, DFK+16, DFK+18, CS18, OSZ19].

Populations (3)

- Crossover-based algorithms obviously need populations. The real problem is getting crossover to be useful.

- Summary:
  - Populations can ensure robustness and parallel speed-ups
  - They are needed for non-elitist algorithms, but not many useful applications of non-elitism could be analyzed theoretically
  - They are needed for cross-over based algorithms, but again our understanding of the usefulness of crossover remains low.
  - → Much work do be done!

Hot Topic 2: Estimation-of-distribution Algorithms (EDA)

- Example: compact Genetic Algorithm (cGA) of Harik, Lobo, and Goldberg [HLG99] with hypothetical pop. size $K \in 2\mathbb{N}$ to maximize $f: \{0,1\}^n \to \mathbb{R}$
  - initialize $\tau = (0.5, \dots, 0.5) \in [0,1]^n$
  - while not terminate
    - sample $x \in \{0,1\}^n$ such that $\Pr[x_i = 1] = \tau_i$ indep. for all $i \in [1..n]$
    - sample $y \in \{0,1\}^n$ such that $\Pr[y_i = 1] = \tau_i$ indep. for all $i \in [1..n]$
    - if $f(y) > f(x)$ then $(x, y) := (y, x)$
    - for all $i \in [1..n]$ do $\tau_i := \tau_i + (x_i - y_i)/K$
- Instead of storing concrete search points, EDAs develop a probabilistic model (represented by the frequency vector τ in the cGA).
  - much richer representation of knowledge

What Can EDAs Do That EAs Can't?

- Robustness to noise:
  - The cGA can cope well with normally distributed additive posterior noise [FKKS17]
  - The UMDA can cope well with 1-bit prior noise [LN19b]
  - (similar result for ACO found earlier [DHK12b, FK13, ST12])
- Leaving local optima: EDAs can optimize multimodal functions faster than many classic EAs [HS18, Doe19a, Doe19b, DK20c]
  - (similar result for ACO found later [BBD21b])
- Model building = representing many good solutions at once:
  - MIMIC can build a probabilistic model that allows to sample a huge number of distant good solutions (experimental) [DK20a]

Difficulty: Genetic Drift

- When a bit i has no influence on whether x or y is better (because other bits have a higher impact), then the frequency τ_i performs a random walk:
  - $\tau_i^{new} = \tau_i + 1/K$ with probability $\tau_i(1 - \tau_i)$
  - $\tau_i^{new} = \tau_i - 1/K$ with probability $\tau_i(1 - \tau_i)$
  - $\tau_i^{new} = \tau_i$ otherwise
- Such random movements can bring the frequency to a random boundary value $\{0,1\}$ → convergence to a sub-optimal solution.
- Insufficient solution: Artificially cap the frequencies into $[1/n, 1 - 1/n]$
- Problems: If frequencies are mostly at the artificial boundaries, then…
  - our probabilistic model is not richer than that of the (1+1) EA
  - the performance can drop [Witt17+LSW21, LN19a+DK20c, Doe19+DZ20a,DL15+DK21b]

Quantifying Genetic Drift

- Good news: From many previous works specifically targeting genetic drift [Sha02, Sha05, Sha06, FKK16] and many runtime analyses coping with genetic drift [Dro06, DLN19, LN17, LN18, HS18, Doe19b, Doe19a, SW19, Wit19, KW20, LSW21], we now understand genetic drift well.
- Genetic drift can be quantified via drift analysis applied to the variance and Martingale concentration results [DZ20b]:
- Theorem (stated for the cGA only): Assume that the cGA optimizes a function with a neutral bit.
  - The first time the frequency of this bit is at the boundary, is $O(K^2)$.
  - The first time this frequencies leaves $[0.25, 0.75]$ is $\Omega(K^2)$.
  - The probability that this frequency leaves the interval $[0.5 - \varepsilon, 0.5 + \varepsilon]$ in the first T iterations, is at most

$$2 \exp\left(-\frac{\varepsilon^2 K^2}{2T}\right).$$

Overcoming Genetic Drift

EDA variants trying to avoid genetic drift outright:

- stable-cGA [FKK16]: cGA with an artificially modified frequency update.
  - $O(n \log n)$ runtime on LeadingOnes
  - exponential runtime on OneMax [DK20b].
- sig-cGA [**DK20b**,DWZ21]: regards a longer history and changes frequencies only when there is sufficient evidence for it
  - $O(n \log n)$ runtime on OneMax, LeadingOnes, and DeceivingLeadingBlocks

Automated ways to set the parameters right:

- Parallel runs with diverse parameter values [Doe19b]
- Smart restarts: Restart with a larger K when the theorem on the previous slide says that genetic drift could have occurred [DZ20a]

Summary: EDA Theory

- Significant progress in the last 5 years:
  - many runtime analyses, many strong methods
  - understanding genetic drift.
- Cool particular results:
  - EDAs work well in the presence of noise
  - EDAs can leave local optima quicker than most EAs
- Many open problems:
  - Tight runtime bounds for classic problems (OneMax)
  - More complete picture how EDAs cope with noise (2 particular results)
  - Theory for multi-variate EDAs (no result yet)

Hot Topic 3: Dynamic and Noisy Optimization

- Dynamic optimization: Optimization when the problem to be solved changes over time
- Noisy optimization: Optimization in the presence of (typically random) errors in the problem data
- Common question: How do EAs perform when the evolutionary optimization process is disturbed by some external (random) source.
- General belief: due to their randomized nature, EAs can cope well with such stochastic disturbances

Dynamic OneMax

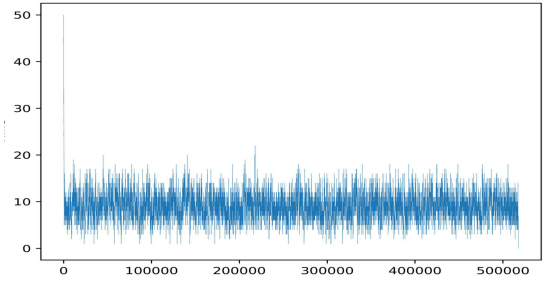
- First theory result (Droste [Dro02]):
  - OneMax function with optimum z: $OM_z(x) = |\{i \in [1..n] \mid x_i = z_i\}|$
  - Dynamic OneMax with 1-bit dynamics: in each iteration, with some small probability p the current optimum z is replaced by a random Hamming neighbor (=a random bit of z is flipped)
  - Result: If $p \leq c\frac{\ln n}{n}$, then the (1+1) EA finds the optimum of this dynamic OneMax function in $O(n^{ce+1+o(1)})$ iterations (expectation).
- Droste [Dro03]: If the dynamic is such that independently with prob. $p' = c\frac{\ln n}{n^2}$ each bit of the optimum is flipped (same expected change), then the runtime bound is $O(n^{4ce+1+o(1)})$.
  - Improved to $22\, n^{1.76..ce+2} \ln n$ by Kötzing, Lissovoi, Witt [KLW15]
  - Improved to $\big(3.77 + \mathrm{o}(1)\big)n^{1.39..ce+1}$ by Dang-Nhu et al. [DNDD+18], valid for all dynamics changing the opt. by at most $c\frac{\ln n}{n}$ in expect.

Interpretation of These Results

- Evolutionary algorithms can be surprisingly robust to dynamically changing problem instances!
- If $p = c\frac{\ln n}{n}$ in the 1-bit dynamic, then in average, every $\frac{n}{c \ln n}$ iterations the optimum moves to a Hamming neighbor
  → and we lose a fitness level (almost always)
- If the fitness distance is d, then we need a roughly $\frac{en}{d}$ iterations to improve the fitness (without dynamic changes)
- When close to the optimum (d constant),
  - it takes $\Theta(n)$ expected time to gain one fitness level without dynamics
  - but we lose expected $\Theta(\log n)$ fitness levels because of the dynamic.
- Despite this, the EA finds the optimum in polynomial time

Why?

- From the proofs in Dang-Nhu et al. [DNDD+18] it seems that EAs make progress by repeatedly
  - hoping for a phase of few dynamic changes
  - and then making exceptionally fast progress
    → supports the general belief that the randomized nature of EAs is the reason for their robustness



A plot of a typical run (fitness distance over time) for n=100, 1-bit dynamic with p=(ln$n$)/n

Warning: If the fitness-distance correlation is weak, EAs find it harder to reoptimize [DDN19]

Noisy Optimization

- Very roughly speaking, similar results hold for noisy optimization, see, e.g., Droste [Dro04], Giessen, Kötzing [GK16], Qian, Bian, Jiang, Tang [QBJT19], Dang-Nhu et al. [DNDD+18], Gavenciak, Geissmann, Lengler [GGL19], Sudholt [Sud21]
- Additional aspect: We can tolerate higher noise levels by
  - resampling (Akimoto, Astete-Morales, Teytaud [AMT15], Qian et al. [QBJT19], D. and Sutton [DS19]),
  - using larger population sizes (Giessen and Kötzing [GK16]),
  - using other algorithms like
    - ant colony optimizer (e.g. Sudholt and Thyssen [ST12]), or
    - EDAs (Friedrich, Kötzing, Krejca, Sutton [FKKS17], Lehre, Nguyen [LN19b])

Summary Dynamic and Noisy Optimization

- Due to their randomized nature, EAs cope well with moderate levels of noise and moderate changes of the problem instance.
- For noisy optimization, one can try to reduce the effect of noise by resampling, larger population size, etc. For dynamic optimization, nothing is known on how to make algorithms more robust.

Hot Topic 4: Dynamic Parameter Choices

- Instead of fixing a parameter (mutation rate, population size, …) once and forever (*static* parameter choice), it might be preferable to change the parameter values during the run of the EA
- Hope:
  - different parameter settings may be optimal at different stages of the optimization process, so by changing the parameter value we can improve the performance
  - we can let the algorithm optimize the parameters itself (on-the-fly parameter choice, *self-adjusting* parameters)
- Experimental work suggests that dynamic parameter choices often outperform static ones (for surveys see [EHM99,KHE15])

Theory for Dynamic Parameter Choices: *Deterministic Schedules*

- *Deterministic variation schedule* for the mutation rate (Jansen and Wegener [JW00, JW06]):
  - Toggle through the mutation rates $\frac{1}{n}, \frac{2}{n}, \frac{4}{n}, \dots, \approx \frac{1}{2}$
  - Result: There is a function where this dynamic EA takes time $O(n^2 \log n)$, but any static EA takes exponential time
  - For most functions, the dynamic EA is slower by a factor of $\log n$
  - [unpublished] For jump functions with (not too small) jump size m, this gives a significant improvement:
    - faster than standard-bit mutation by a factor of $m^{\Omega(m)}/\log n$
    - slower than fast mutation by a factor of $2^{\Omega(m)} \log n$
- → First example proving that dynamic parameter choices can be beneficial.

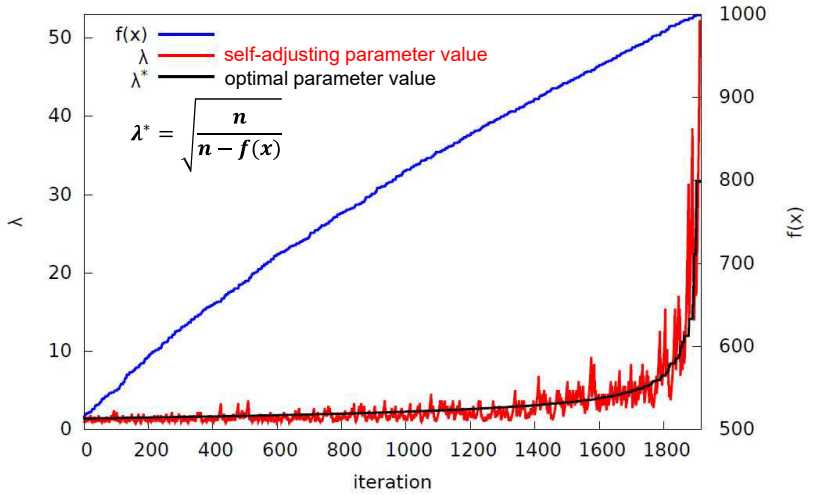
Theory for Dynamic Parameter Choices: *Depending on the Fitness*

- *Fitness-dependent mutation rate* [BDN10]: When optimizing the LeadingOnes test function $LO: \{0,1\}^n \to \{0, \dots, n\}$ with the (1+1) EA
  - the fixed mutation rate $p = \frac{1}{n}$ gives a runtime of $\approx 0.86\, n^2$
  - the fixed mutation rate $p = \frac{1.59}{n}$ gives $\approx 0.77\, n^2$ (optimal fixed mut. rate)
  - the mutation rate $\boldsymbol{p = \frac{1}{LO(x)+1}}$ gives $\approx 0.68\, n^2$ (optimal dynamic rate)
- *Fitness-dependent offspring pop. size*
  - with the right fitness-dependent λ, the $(1,\lambda)$ EA optimizes OneMax in time $O(n\,\lambda/\log\lambda + n \log n)$ [BLS14]
  - with $\boldsymbol{\lambda = \sqrt{\frac{n}{n-f(x)}}}$, the $(1+(\lambda,\lambda))$ GA optimizes OneMax in time $O(n)$ instead of roughly $n\sqrt{\log n}$ with best static λ [DDE15]
- → Fitness-dependent parameters can pay off. It is hard to find the optimal dependence, but many others give improvements as well.

Theory for Dynamic Parameter Choices: *Success-based Dynamics*

- *Success-based* choice of island number: You can reduce of the parallel runtime (but not the total work) of an island model when choosing the number of islands dynamically (Lässig and Sudholt [LS11]):
  - double the number of islands after each iteration without fitness gain
  - half the number of islands after each improving iteration
- *Success-based* choice (1/5-th rule) of λ in the (1+(λ,λ)) GA finds the optimal mutation strength [DD18] ($F > 1$ a constant):
  - $\lambda := \sqrt[4]{F}\,\lambda$ after each iteration without fitness gain
  - $\lambda := \lambda/F$ after each improving iteration
  - Important that the fourth root is taken (→ 1/5-th rule). The doubling scheme of [LS11] would not work
- Simple mechanisms to automatically find the current-best parameter setting (note: this is great even when the optimal parameter does not change over time, but is hard to know beforehand)

A Run of the Self-Adjusting $(1 + (\lambda, \lambda))$ GA on OneMax ($n = 1000$)



Theory for Dynamic Parameter Choices: *Success-based Dynamics II – Stagnation Detection*

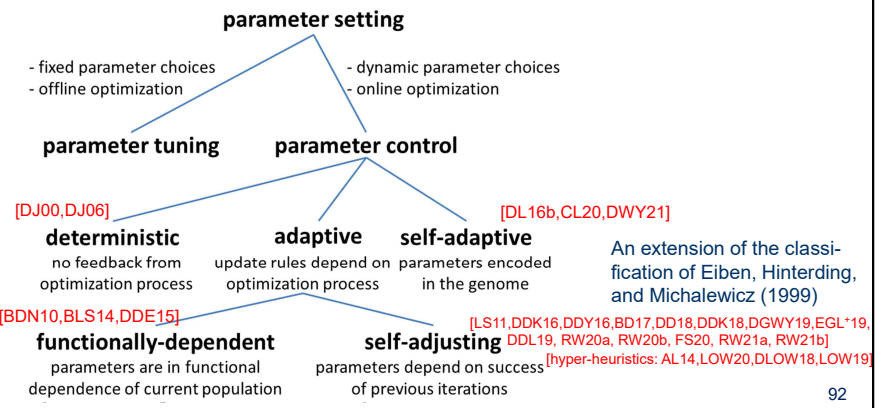
- Previous success-based dynamics:
  - Work well when the characteristics of the landscape changes slowly → good parameter values "follow" the changes of the landscape.
  - Not much known for abrupt changes, e.g., jump functions.
- Very recent idea: "Stagnation detection" [RW20b, RW21a, RW21b]
  - When for too long (details omitted) no improvement happened, increase the mutation strength (because there'll be no improvement close by).
  - Can be added to all mutation-based algorithms.
  - Gives the best runtime for the (1+1) EA on jump functions.
- → Very simple, very promising idea!

Theory for Dynamic Parameter Choices: *Self-Adaptation*

- So far: An extra mechanism added onto the EA controls the parameters.
- *Self-adaptation*: Let the usual variation-selection cycle do this for you!
  - Add the parameter to the individual (extended representation)
  - Extended mutation: first mutate the parameter, then mutate the individual taking into account the new parameter value
  - Hope: Better parameter values lead to fitter individuals which are preferred by the (non-extended) selection mechanisms of the EA
- First proof that this can work (artificial example) [DL16b]
- Self-adaptation can find the right mutation rate for the $(1,\lambda)$ EA on OneMax (classic benchmark) [DWY21]
- Self-adaptation can find the right mutation rate for the (μ,λ) EA on LeadingOnes (also with unknown-solution length) [CL20]
- → Generic way to adapt parameters, but not well-understood

Summary Dynamic Parameter Choices

- State of the art: A growing number of results, some very promising
  - personal opinion: this is the future of discrete EC, as it allows to integrate very powerful natural principles like adaption and learning
  - survey on theory: [DD20]



An extension of the classification of Eiben, Hinterding, and Michalewicz (1999)

Hot Topic 5: Fine-grained Runtime Analysis

- Classic runtime analysis: Analyze the time until the optimum is found.
- Recently: Runtime notions that give more or more relevant information.

- Fixed budget perspective: Analyze the (expected) solution quality obtainable in a given time budget [**JZ12**,DJWZ13,JZ14a,JZ14b,LS15,NNS17,DDY20,KW20]
  - Interesting from the application perspective, but difficult to analyze!

- Fixed target analysis, starting with good solutions: Classic runtime notion extended to arbitrary starting/ target solution qualities [**BDDV20**,**ABD20a**,DK21a].
  - Classic proofs can be re-used, but different algorithms become good.

- Parameterized complexity: Analyze the runtime relative to a parameter of the input instance [Sto06,Sto07,KLNO10,SN12,**KN13**,SNN14,FN15,CLNP16,Sut21]
  - Hot topic in classic algorithms since 1999

Part VI:

Conclusion

Summary

- Theoretical research gives deep insights in the working principles of EC, with results that are of a different nature than in experimental work
  - "very true" (=proven), but often apply to idealized settings only
  - *for all* instances and problem sizes, but sometimes less precise
    - often only asymptotic results instead of absolute numbers
  - proofs tell us *why* certain facts are true

- The different nature of theoretical and experimental results implies that a real understanding is best obtained from a combination of both

- *Theory-driven curiosity* and the *clarifying nature of mathematical proofs* can lead to new ideas, insights and algorithms

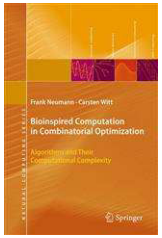
Summary (2): How to Use Theory in Your Work?

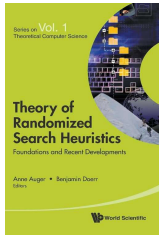
- Try to read theory papers (or listen to the talks in one of the theory track sessions), but don't expect more than from other papers
  - Neither a theory nor an experimental paper can tell you the best algorithm for your particular problem, but both can suggest ideas

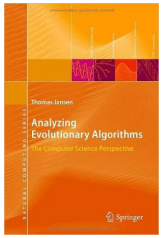
- Try "theory-style thinking": take a very very simplified version of your problem and imagine what could work and why

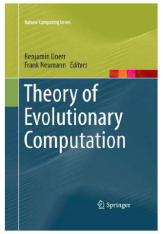
- Don't be shy to talk to the theory people!
  - they will not have the ultimate solution and their mathematical education makes them very cautious presenting an ultimate solution
  - but they might be able to prevent you from a wrong path or suggest alternatives to your current approach

Theory Books (Written for Theory People, But Not Too Hard to Read)









- Neumann/Witt (2010). Bioinspired Computation in Combinatorial Optimization, Springer
- Auger/Doerr (2011). Theory of Randomized Search Heuristics, World Scientific
- Jansen (2013). Analyzing Evolutionary Algorithms, Springer
- Doerr/Neumann (2020). Theory of Evolutionary Computation – Recent Developments in Discrete Optimization, Springer.
  → Most chapters are on the arxiv; author-generated version online ←

Acknowledgments

- This tutorial is also based upon work from COST Action CA15140 `Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO)' supported by COST (European Cooperation in Science and Technology).

cost
EUROPEAN COOPERATION IN SCIENCE AND TECHNOLOGY

Thanks for your attention!

Appendix A

Big Oh Notation

Big-Oh Notation: Motivation

- *Big-Oh notation*, also called *asymptotic notation* or *Landau symbols*, are a convenient way to roughly describe how a quantity depends on another, e.g., how the runtime $T(n)$ depends on the problem size n.
- We need this, because often
  - it is often impossible to precisely compute $T(n)$ as function of n, and
  - we sometimes intentionally only aim at a general description of a phenomenon (e.g., the runtime is linear, quadratic, or exponential) than a precise, but hard to understand formula (e.g., the following result from [Wit13]).

Theorem 4.1. *On any linear function on n variables, the optimization time of the (1+1) EA with mutation probability $0 < p < 1$ is at most*

$$(1-p)^{1-n}\left(\frac{n\alpha^2(1-p)^{1-n}}{\alpha-1}+\frac{\alpha}{\alpha-1}\frac{\ln(1/p)+(n-1)\ln(1-p)+r}{p}\right)=:b(r),$$

with probability at least $1-e^{-r}$ for any $r>0$, and it is at most $b(1)$ in expectation, where $\alpha>1$ can be chosen arbitrarily (even depending on n).

Big-Oh Notation: Definition $O(.)$

- Let us continue to use the example of the expected runtime $T(n) > 0$ of some algorithm on some problem that is defined for all problems sizes n (e.g., the expected runtime of the (1+1) EA on the n-dimensional ONEMAX function.
- Big-Oh notation allows to describe the *asymptotic behavior* of the runtime, that is, how the runtime depends on n when we think of n being large. On the other hand, we do not say anything for a concrete, fixed value of n like $n = 1000$.
- **Definition**: We say that $T(n)$ is $O(f(n))$ for some function $f: \mathbb{N} \to \mathbb{R}_{>0}$ if there is a constant c such $T(n) \leq cf(n)$ for all $n \in \mathbb{N}$.
  - We write $T = O(f)$ or $T \in O(f)$. Note that the first version does not make much sense, but is more common.
  - We write $T = O(n^2)$ when $f(n) = n^2$

Big-Oh Notation: $O, \Omega, \Theta, o, \omega$

- Asymptotic upper bound:
  - $T = O(f)$ if there is a constant c such $T(n) \leq cf(n)$ for all $n \in \mathbb{N}$.
- Asymptotic lower bound:
  - $T = \Omega(f)$ if there is a constant $c \geq 0$ such $T(n) \geq cf(n)$ for all $n \in \mathbb{N}$.
- Asymptotically equal:
  - $T = \Theta(f)$ if $T = O(f)$ and $T = \Omega(f)$.
- Asymptotically smaller, T grows slower than f:
  - $T = o(f)$ if $\lim_{n\to\infty} \frac{T(n)}{f(n)} = 0$
- Asymptotically larger, T grows faster than f:
  - $T = \omega(f)$ if $\lim_{n\to\infty} \frac{T(n)}{f(n)} = \infty$

Appendix B

List of References

References

[ABD20a] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. Fast mutation in crossover-based algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1268–1276. ACM, 2020.

[ABD20b] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. First steps towards a runtime analysis when starting with a good solution. In *Parallel Problem Solving From Nature, PPSN 2020, Part II*, pages 560–573. Springer, 2020.

[ABD21] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. Lazy parameter tuning and control: choosing all parameters randomly from a power-law distribution. In *Genetic and Evolutionary Computation Conference, GECCO 2021*. ACM, 2021. To appear.

[AD11] Anne Auger and Benjamin Doerr, editors. *Theory of Randomized Search Heuristics*. World Scientific Publishing, 2011.

[AD20] Denis Antipov and Benjamin Doerr. Runtime analysis of a heavy-tailed $(1 + (\lambda, \lambda))$ genetic algorithm on jump functions. In *Parallel Problem Solving From Nature, PPSN 2020, Part II*, pages 545–559. Springer, 2020.

[ADFH18] Denis Antipov, Benjamin Doerr, Jiefeng Fang, and Tangi Hetet. Runtime analysis for the $(\mu + \lambda)$ EA optimizing OneMax. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 1459–1466. ACM, 2018.

[ADK19] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. A tight runtime analysis for the $(1 + (\lambda, \lambda))$ GA on LeadingOnes. In *Foundations of Genetic Algorithms, FOGA 2019*, pages 169–182. ACM, 2019.

[ADK20] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. The $(1 + (\lambda, \lambda))$ GA is even faster on multimodal problems. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1259–1267. ACM, 2020.

[ADY19] Denis Antipov, Benjamin Doerr, and Quentin Yang. The efficiency threshold for the offspring population size of the (μ, λ) EA. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 1461–1469. ACM, 2019.

[AL14] Fawaz Alanazi and Per Kristian Lehre. Runtime analysis of selection hyper-heuristics with classical learning mechanisms. In *Congress on Evolutionary Computation, CEC 2104*, pages 2515–2523. IEEE, 2014.

[AMT15] Youhei Akimoto, Sandra Astete Morales, and Olivier Teytaud. Analysis of runtime of optimization algorithms for noisy functions over discrete codomains. *Theoretical Computer Science*, 605:42–50, 2015.

[Bäc93] Thomas Bäck. Optimal mutation rates in genetic search. In *International Conference on Genetic Algorithms, ICGA 1993*, pages 2–8. Morgan Kaufmann, 1993.

[Bäc96] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice – Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[BB19] Anton Bassin and Maxim Buzdalov. The 1/5-th rule with rollbacks: on self-adjustment of the population size in the $(1+(\lambda, \lambda))$ GA. In *Genetic and Evolutionary Computation Conference Companion, GECCO 2019*, pages 277–278. ACM, 2019.

[BB20] Anton Bassin and Maxim Buzdalov. The $(1 + (\lambda, \lambda))$ genetic algorithm for permutations. In *Genetic and Evolutionary Computation Conference, GECCO 2020, Companion Volume*, pages 1669–1677. ACM, 2020.

[BBD+09] Surender Baswana, Somenath Biswas, Benjamin Doerr, Tobias Friedrich, Piyush P. Kurur, and Frank Neumann. Computing single source shortest paths using single-objective fitness. In *Foundations of Genetic Algorithms, FOGA 2009*, pages 59–66. ACM, 2009.

[BBD21a] Henry Bambury, Antoine Bultel, and Benjamin Doerr. Generalized jump functions. In *Genetic and Evolutionary Computation Conference, GECCO 2021*. ACM, 2021. To appear.

[BBD21b] Riade Benbaki, Ziyad Benomar, and Benjamin Doerr. A rigorous runtime analysis of the 2-MMAS$_{\text{ib}}$ on jump functions: ant colony optimizers can cope well with local optima. In *Genetic and Evolutionary Computation Conference, GECCO 2021*. ACM, 2021. To appear.

[BD17] Maxim Buzdalov and Benjamin Doerr. Runtime analysis of the $(1 + (\lambda, \lambda))$ genetic algorithm on random satisfiable 3-CNF formulas. In *Genetic and Evolutionary Computation Conference, GECCO 2017*, pages 1343–1350. ACM, 2017.

[BDDV20] Maxim Buzdalov, Benjamin Doerr, Carola Doerr, and Dmitry Vinokurov. Fixed-target runtime analysis. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1295–1303. ACM, 2020.

[BDN10] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Parallel Problem Solving from Nature, PPSN 2010*, pages 1–10. Springer, 2010.

[BFM97] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., 1997.

[BLS14] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. Unbiased black-box complexity of parallel search. In *Parallel Problem Solving from Nature, PPSN 2014*, pages 892–901. Springer, 2014.

[BS19] Jakob Bossek and Dirk Sudholt. Time complexity analysis of RLS and (1+1) EA for the edge coloring problem. In *Foundations of Genetic Algorithms, FOGA 2019*, pages 102–115. ACM, 2019.

[CDEL18] Dogan Corus, Duc-Cuong Dang, Anton V. Eremeev, and Per Kristian Lehre. Level-based analysis of genetic algorithms and other search processes. *IEEE Transactions on Evolutionary Computation*, 22:707–719, 2018.

[CL20] Brendan Case and Per Kristian Lehre. Self-adaptation in nonelitist evolutionary algorithms on discrete problems with unknown structure. *IEEE Transactions on Evolutionary Computation*, 24:650–663, 2020.

[CLNP16] Dogan Corus, Per Kristian Lehre, Frank Neumann, and Mojgan Pourhassan. A parameterised complexity analysis of bi-level optimisation with evolutionary algorithms. *Evolutionary Computation*, 24:183–203, 2016.

[CO18] Dogan Corus and Pietro S. Oliveto. Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. *IEEE Transactions on Evolutionary Compututation*, 22:720–732, 2018.

[CO20] Dogan Corus and Pietro S. Oliveto. On the benefits of populations for the exploitation speed of standard steady-state genetic algorithms. *Algorithmica*, 82:3676–3706, 2020.

[CS18] Edgar Covantes Osuna and Dirk Sudholt. Runtime analysis of probabilistic crowding and restricted tournament selection for bimodal optimisation. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 929–936. ACM, 2018.

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

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

[DD18] Benjamin Doerr and Carola Doerr. Optimal static and self-adjusting parameter choices for the $(1 + (\lambda, \lambda))$ genetic algorithm. *Algorithmica*, 80:1658–1709, 2018.

[DD20] Benjamin Doerr and Carola Doerr. Theory of parameter control for discrete black-box optimization: provable performance gains through dynamic parameter choices. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 271–321. Springer, 2020. Also available at https://arxiv.org/abs/1804.05650.

[DDE13] Benjamin Doerr, Carola Doerr, and Franziska Ebel. Lessons from the black-box: Fast crossover-based genetic algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2013*, 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.

[DDK14a] Benjamin Doerr, Carola Doerr, and Timo Kötzing. Unbiased black-box complexities of jump functions: how to cross large plateaus. In *Genetic and Evolutionary Computation Conference, GECCO 2014*, pages 769–776. ACM, 2014.

[DDK14b] Benjamin Doerr, Carola Doerr, and Timo Kötzing. The unbiased black-box complexity of partition is polynomial. *Artificial Intelligence*, 216:275–286, 2014.

[DDK17] Benjamin Doerr, Carola Doerr, and Timo Kötzing. Unknown solution length problems with no asymptotically optimal run time. In *Genetic and Evolutionary Computation Conference, GECCO 2017*, pages 921–928. ACM, 2017.

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

[DDL19] Benjamin Doerr, Carola Doerr, and Johannes Lengler. Self-adjusting mutation rates with provably optimal success rules. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 1479–1487. ACM, 2019.

[DDN19] Benjamin Doerr, Carola Doerr, and Frank Neumann. Fast re-optimization via structural diversity. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 233–241. ACM, 2019.

[DDST16] Benjamin Doerr, Carola Doerr, Reto Spöhel, and Henning Thomas. Playing Mastermind with many colors. *Journal of the ACM*, 63:42:1–42:23, 2016.

[DDY16] Benjamin Doerr, Carola Doerr, and Jing Yang. k-bit mutation with self-adjusting k outperforms standard bit mutation. In *Parallel Problem Solving from Nature, PPSN 2016*, pages 824–834. Springer, 2016.

[DDY20] Benjamin Doerr, Carola Doerr, and Jing Yang. Optimal parameter choices via precise black-box analysis. *Theoretical Computer Science*, 801:1–34, 2020.

[DFK+16] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. Escaping local optima with diversity mechanisms and crossover. In *Genetic and Evolutionary Computation Conference, GECCO 2016*, pages 645–652. ACM, 2016.

[DFK+18] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation*, 22:484–497, 2018.

[DGWY19] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. The $(1 + \lambda)$ evolutionary algorithm with self-adjusting mutation rate. *Algorithmica*, 81:593–631, 2019.

[DHK07] Benjamin Doerr, Edda Happ, and Christian Klein. A tight bound for the (1 + 1)-EA for the single source shortest path problem. In *Congress on Evolutionary Computation, CEC 2007*, pages 1890–1895. IEEE, 2007.

[DHK12a] Benjamin Doerr, Edda Happ, and Christian Klein. Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science*, 425:17–33, 2012.

[DHK12b] Benjamin Doerr, Ashish Ranjan Hota, and Timo Kötzing. Ants easily solve stochastic shortest path problems. In *Genetic and Evolutionary Computation Conference, GECCO 2012*, pages 17–24. ACM, 2012.

[DHN07] Benjamin Doerr, Nils Hebbinghaus, and Frank Neumann. Speeding up evolutionary algorithms through asymmetric mutation operators. *Evolutionary Computation*, 15:401–410, 2007.

[DJ07] Benjamin Doerr and Daniel Johannsen. Adjacency list matchings: an ideal genotype for cycle covers. In *Genetic and Evolutionary Computation Conference, GECCO 2007*, pages 1203–1210. ACM, 2007.

[DJ10] Benjamin Doerr and Daniel Johannsen. Edge-based representation beats vertex-based representation in shortest path problems. In *Genetic and Evolutionary Computation Conference, GECCO 2010*, pages 759–766. ACM, 2010.

[DJK+11] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Per Kristian Lehre, Markus Wagner, and Carola Winzen. Faster black-box algorithms through higher arity operators. In *Foundations of Genetic Algorithms, FOGA 2011*, pages 163–172. ACM, 2011.

[DJK+13] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Frank Neumann, and Madeleine Theile. More effective crossover operators for the all-pairs shortest path problem. *Theoretical Computer Science*, 471:12–26, 2013.

[DJL17] Duc-Cuong Dang, Thomas Jansen, and Per Kristian Lehre. Populations can be essential in tracking dynamic optima. *Algorithmica*, 78:660–680, 2017.

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

[DJW98a] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the optimization of unimodal functions with the $(1+1)$ evolutionary algorithm. In *Parallel Problem Solving from Nature, PPSN 1998*, pages 13–22. Springer, 1998.

[DJW98b] Stefan Droste, Thomas Jansen, and Ingo Wegener. A rigorous complexity analysis of the (1+1) evolutionary algorithm for linear functions with Boolean inputs. In *International Conference on Evolutionary Computation, ICEC 1998*, pages 499–504. IEEE, 1998.

[DJW02] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[DJW10] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. In *Genetic and Evolutionary Computation Conference, GECCO 2010*, pages 1449–1456. ACM, 2010.

[DJW12] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. *Algorithmica*, 64:673–697, 2012.

[DJWZ13] Benjamin Doerr, Thomas Jansen, Carsten Witt, and Christine Zarges. A method to derive fixed budget results from expected optimisation times. In *Genetic and Evolutionary Computation Conference, GECCO 2013*, pages 1581–1588. ACM, 2013.

[DK15] Benjamin Doerr and Marvin Künnemann. Optimizing linear functions with the $(1+\lambda)$ evolutionary algorithm—different asymptotic runtimes for different instances. *Theoretical Computer Science*, 561:3–23, 2015.

[DK19] Benjamin Doerr and Timo Kötzing. Multiplicative up-drift. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 1470–1478. ACM, 2019.

[DK20a] Benjamin Doerr and Martin S. Krejca. Bivariate estimation-of-distribution algorithms can find an exponential number of optima. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 796–804. ACM, 2020.

[DK20b] Benjamin Doerr and Martin S. Krejca. Significance-based estimation-of-distribution algorithms. *IEEE Transactions on Evolutionary Computation*, 24:1025–1034, 2020.

[DK20c] Benjamin Doerr and Martin S. Krejca. The univariate marginal distribution algorithm copes well with deception and epistasis. In *Evolutionary Computation in Combinatorial Optimization, EvoCOP 2020*, pages 51–66. Springer, 2020.

[DK21a] Benjamin Doerr and Timo Kötzing. Lower bounds from fitness levels made easy. In *Genetic and Evolutionary Computation Conference, GECCO 2021*. ACM, 2021. To appear.

[DK21b] Benjamin Doerr and Martin S. Krejca. A simplified run time analysis of the univariate marginal distribution algorithm on LeadingOnes. *Theoretical Computer Science*, 851:121–128, 2021.

[DKS07] Benjamin Doerr, Christian Klein, and Tobias Storch. Faster evolutionary algorithms by superior graph representation. In *Foundations of Computational Intelligence, FOCI 2007*, pages 245–250. IEEE, 2007.

[DL15] Duc-Cuong Dang and Per Kristian Lehre. Simplified runtime analysis of estimation of distribution algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2015*, pages 513–518. ACM, 2015.

[DL16a] Duc-Cuong Dang and Per Kristian Lehre. Runtime analysis of non-elitist populations: from classical optimisation to partial information. *Algorithmica*, 75:428–461, 2016.

[DL16b] Duc-Cuong Dang and Per Kristian Lehre. Self-adaptation of mutation rates in non-elitist populations. In *Parallel Problem Solving from Nature, PPSN 2016*, pages 803–813. Springer, 2016.

[DLMN17] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. Fast genetic algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2017*, pages 777–784. ACM, 2017.

[DLN19] Duc-Cuong Dang, Per Kristian Lehre, and Phan Trung Hai Nguyen. Level-based analysis of the univariate marginal distribution algorithm. *Algorithmica*, 81:668–702, 2019.

[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 *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 1015–1022. ACM, 2018.

[DN20] Benjamin Doerr and Frank Neumann, editors. *Theory of Evolutionary Computation—Recent Developments in Discrete Optimization*. Springer, 2020. Also available at https://cs.adelaide.edu.au/~frank/papers/TheoryBook2019-selfarchived.pdf.

[DNDD+18] Raphaël Dang-Nhu, Thibault Dardinier, Benjamin Doerr, Gautier Izacard, and Dorian Nogneng. A new analysis method for evolutionary optimization of dynamic and noisy objective functions. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 1467–1474. ACM, 2018.

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

[Doe19a] Benjamin Doerr. An exponential lower bound for the runtime of the compact genetic algorithm on jump functions. In *Foundations of Genetic Algorithms, FOGA 2019*, pages 25–33. ACM, 2019.

[Doe19b] Benjamin Doerr. A tight runtime analysis for the cGA on jump functions: EDAs can cross fitness valleys at no extra cost. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 1488–1496. ACM, 2019.

[Doe20a] Benjamin Doerr. Does comma selection help to cope with local optima? In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1304–1313. ACM, 2020.

[Doe20b] Benjamin Doerr. Lower bounds for non-elitist evolutionary algorithms via negative multiplicative drift. In *Parallel Problem Solving From Nature, PPSN 2020, Part II*, pages 604–618. Springer, 2020.

[Doe21] Benjamin Doerr. Exponential upper bounds for the runtime of randomized search heuristics. *Theoretical Computer Science*, 851:24–38, 2021.

[dPdLDD15] Axel de Perthuis de Laillevault, Benjamin Doerr, and Carola Doerr. Money for nothing: speeding up evolutionary algorithms through better initialization. In *Genetic and Evolutionary Computation Conference, GECCO 2015*, pages 815–822. ACM, 2015.

[Dro02] Stefan Droste. Analysis of the (1+1) EA for a dynamically changing OneMax-variant. In *Congress on Evolutionary Computation, CEC 2002*, pages 55–60. IEEE, 2002.

[Dro03] Stefan Droste. Analysis of the (1+1) EA for a dynamically bitwise changing OneMax. In *Genetic and Evolutionary Computation Conference, GECCO 2003*, pages 909–921. Springer, 2003.

[Dro04] Stefan Droste. Analysis of the (1+1) EA for a noisy OneMax. In *Genetic and Evolutionary Computation Conference, GECCO 2004*, pages 1088–1099. Springer, 2004.

[Dro06] Stefan Droste. A rigorous analysis of the compact genetic algorithm for linear functions. *Natural Computing*, 5:257–283, 2006.

[DS19] Benjamin Doerr and Andrew M. Sutton. When resampling to cope with noise, use median, not mean. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 242–248. ACM, 2019.

[DSW13] Benjamin Doerr, Dirk Sudholt, and Carsten Witt. When do evolutionary algorithms optimize separable functions in parallel? In *Foundations of Genetic Algorithms, FOGA 2013*, pages 48–59. ACM, 2013.

[DT09] Benjamin Doerr and Madeleine Theile. Improved analysis methods for crossover-based algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2009*, pages 247–254. ACM, 2009.

[DW12a] Benjamin Doerr and Carola Winzen. Memory-restricted black-box complexity of OneMax. *Information Processing Letters*, 112:32–34, 2012.

[DW12b] Benjamin Doerr and Carola Winzen. Reducing the arity in unbiased black-box complexity. In *Genetic and Evolutionary Computation Conference, GECCO 2012*, pages 1309–1316. ACM, 2012.

[DW14] Benjamin Doerr and Carola Winzen. Ranking-based black-box complexity. *Algorithmica*, 68:571–609, 2014.

[DWY21] Benjamin Doerr, Carsten Witt, and Jing Yang. Runtime analysis for self-adaptive mutation rates. *Algorithmica*, 83:1012–1053, 2021.

[DWZ21] Benjamin Doerr, Shouda Wang, and Weijie Zheng. Choosing the right algorithm with hints from complexity theory. In *International Joint Conference on Artificial Intelligence, IJCAI 2021*. ijcai.org, 2021. To appear.

[DZ20a] Benjamin Doerr and Weijie Zheng. A parameter-less compact genetic algorithm. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 805–813. ACM, 2020.

[DZ20b] Benjamin Doerr and Weijie Zheng. Sharp bounds for genetic drift in estimation-of-distribution algorithms. *IEEE Transactions on Evolutionary Computation*, 24:1140–1149, 2020.

[DZ21] Benjamin Doerr and Weijie Zheng. Theoretical analyses of multi-objective evolutionary algorithms on multi-modal objectives. In *Conference on Artificial Intelligence, AAAI 2021*. AAAI Press, 2021. To appear.

[EGL+19] Hafsteinn Einarsson, Marcelo Matheus Gauy, Johannes Lengler, Florian Meier, Asier Mujika, Angelika Steger, and Felix Weissenberger. The linear hidden subset problem for the (1+1) EA with scheduled and adaptive mutation rates. *Theoretical Computer Science*, 785:150–170, 2019.

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

[FGQW18a] Tobias Friedrich, Andreas Göbel, Francesco Quinzan, and Markus Wagner. Evolutionary algorithms and submodular functions: Benefits of heavy-tailed mutations. *CoRR*, abs/1805.10902, 2018.

[FGQW18b] Tobias Friedrich, Andreas Göbel, Francesco Quinzan, and Markus Wagner. Heavy-tailed mutation operators in single-objective combinatorial optimization. In *Parallel Problem Solving from Nature, PPSN 2018, Part I*, pages 134–145. Springer, 2018.

[FHH+09] Tobias Friedrich, Jun He, Nils Hebbinghaus, Frank Neumann, and Carsten Witt. Analyses of simple hybrid algorithms for the vertex cover problem. *Evolutionary Computation*, 17:3–19, 2009.

[FHN07] Tobias Friedrich, Nils Hebbinghaus, and Frank Neumann. Rigorous analyses of simple diversity mechanisms. In *Genetic and Evolutionary Computation Conference, GECCO 2007*, pages 1219–1225. ACM, 2007.

[FHN09] Tobias Friedrich, Nils Hebbinghaus, and Frank Neumann. Comparison of simple diversity mechanisms on plateau functions. *Theoretical Computer Science*, 410:2455–2462, 2009.

[FK13] Matthias Feldmann and Timo Kötzing. Optimizing expected path lengths with ant colony optimization using fitness proportional update. In *Foundations of Genetic Algorithms, FOGA 2013*, pages 65–74. ACM, 2013.

[FKK16] Tobias Friedrich, Timo Kötzing, and Martin S. Krejca. EDAs cannot be balanced and stable. In *Genetic and Evolutionary Computation Conference, GECCO 2016*, pages 1139–1146. ACM, 2016.

[FKKS17] Tobias Friedrich, Timo Kötzing, Martin S. Krejca, and Andrew M. Sutton. The compact genetic algorithm is efficient under extreme Gaussian noise. *IEEE Transactions on Evolutionary Computation*, 21:477–490, 2017.

[FN15] Tobias Friedrich and Frank Neumann. Maximizing submodular functions under matroid constraints by evolutionary algorithms. *Evolutionary Computation*, 23:543–558, 2015.

[FQW18] Tobias Friedrich, Francesco Quinzan, and Markus Wagner. Escaping large deceptive basins of attraction with heavy-tailed mutation operators. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 293–300. ACM, 2018.

[FS20] Mario Alejandro Hevia Fajardo and Dirk Sudholt. On the choice of the parameter control mechanism in the $(1 + (\lambda, \lambda))$ genetic algorithm. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 832–840. ACM, 2020.

[FW04] Simon Fischer and Ingo Wegener. The Ising model on the ring: mutation versus recombination. In *Genetic and Evolutionary Computation, GECCO 2004*, pages 1113–1124. Springer, 2004.

[FW05] Simon Fischer and Ingo Wegener. The one-dimensional Ising model: Mutation versus recombination. *Theoretical Computer Science*, 344:208–225, 2005.

[GGL19] Tomas Gavenciak, Barbara Geissmann, and Johannes Lengler. Sorting by swaps with noisy comparisons. *Algorithmica*, 81:796–827, 2019.

[GK16] Christian Gießen and Timo Kötzing. Robustness of populations in stochastic environments. *Algorithmica*, 75:462–489, 2016.

[GKS99] Josselin Garnier, Leila Kallel, and Marc Schoenauer. Rigorous hitting times for binary mutations. *Evolutionary Computation*, 7:173–203, 1999.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., 1989.

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

[GW03] Oliver Giel and Ingo Wegener. Evolutionary algorithms and the maximum matching problem. In *Symposium on Theoretical Aspects of Computer Science, STACS 2003*, pages 415–426. Springer, 2003.

[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.

[GW18] Christian Gießen and Carsten Witt. Optimal mutation rates for the $(1 + \lambda)$ EA on OneMax through asymptotically tight drift analysis. *Algorithmica*, 80:1710–1731, 2018.

[HGAK06] Nikolaus Hansen, Fabian Gemperle, Anne Auger, and Petros Koumoutsakos. When do heavy-tail distributions help? In *Parallel Problem Solving from Nature, PPSN 2006*, pages 62–71. Springer, 2006.

[HGD94] Jeffrey Horn, David E. Goldberg, and Kalyanmoy Deb. Long path problems. In *Parallel Problem Solving from Nature, PPSN 1994*, pages 149–158. Springer, 1994.

[HJKN08] Edda Happ, Daniel Johannsen, Christian Klein, and Frank Neumann. Rigorous analyses of fitness-proportional selection for optimizing linear functions. In *Genetic and Evolutionary Computation Conference, GECCO 2008*, pages 953–960. ACM, 2008.

[HLG99] Georges R. Harik, Fernando G. Lobo, and David E. Goldberg. The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 3:287–297, 1999.

[HS18] Václav Hasenöhrl and Andrew M. Sutton. On the runtime dynamics of the compact genetic algorithm on jump functions. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 967–974. ACM, 2018.

[HY01] Jun He and Xin Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127:51–81, 2001.

[Jäg08] Jens Jägersküpper. A blend of Markov-chain and drift analysis. In *Parallel Problem Solving From Nature, PPSN 2008*, pages 41–51. Springer, 2008.

[Jan07] Thomas Jansen. On the brittleness of evolutionary algorithms. In *Foundations of Genetic Algorithms, FOGA 2007*, pages 54–69. Springer, 2007.

[Jan13] Thomas Jansen. *Analyzing Evolutionary Algorithms – The Computer Science Perspective.* Springer, 2013.

[JJW05] 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.

[JOZ13] Thomas Jansen, Pietro S. Oliveto, and Christine Zarges. Approximating vertex cover using edge-based representations. In *Foundations of Genetic Algorithms, FOGA 2013*, pages 87–96. ACM, 2013.

[JS05] Thomas Jansen and Ulf Schellbach. Theoretical analysis of a mutation-based evolutionary algorithm for a tracking problem in the lattice. In *Genetic and Evolutionary Computation Conference, GECCO 2005*, pages 841–848. ACM, 2005.

[JS07] Jens Jägersküpper and Tobias Storch. When the plus strategy outperforms the comma strategy and when not. In *Foundations of Computational Intelligence, FOCI 2007*, pages 25–32. IEEE, 2007.

[JW00] Thomas Jansen and Ingo Wegener. On the choice of the mutation probability for the (1+1) EA. In *Parallel Problem Solving from Nature, PPSN 2000*, pages 89–98. Springer, 2000.

[JW02] Thomas Jansen and Ingo Wegener. The analysis of evolutionary algorithms – a proof that crossover really can help. *Algorithmica*, 34:47–66, 2002.

[JW05] Thomas Jansen and Ingo Wegener. Real royal road functions – where crossover provably is essential. *Discrete Applied Mathematics*, 149:111–125, 2005.

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

[JZ12] Thomas Jansen and Christine Zarges. Fixed budget computations: a different perspective on run time analysis. In Terence Soule and Jason H. Moore, editors, *Genetic and Evolutionary Computation Conference, GECCO 2012*, pages 1325–1332. ACM, 2012.

[JZ14a] Thomas Jansen and Christine Zarges. Performance analysis of randomised search heuristics operating with a fixed budget. *Theoretical Computer Science*, 545:39–58, 2014.

[JZ14b] Thomas Jansen and Christine Zarges. Reevaluating immune-inspired hypermutations using the fixed budget perspective. *IEEE Transactions on Evolutionary Computation*, 18:674–688, 2014.

[KAD19] Vitalii Karavaev, Denis Antipov, and Benjamin Doerr. Theoretical and empirical study of the $(1 + (\lambda, \lambda))$ EA on the Leading-Ones problem. In *Genetic and Evolutionary Computation Conference, GECCO 2019, Companion Material*, pages 2036–2039. ACM, 2019.

[KHE15] Giorgos Karafotias, Mark Hoogendoorn, and Ágoston E. Eiben. Parameter control in evolutionary algorithms: trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19:167–187, 2015.

[KLNO10] Stefan Kratsch, Per Kristian Lehre, Frank Neumann, and Pietro Simone Oliveto. Fixed parameter evolutionary algorithms and maximum leaf spanning trees: a matter of mutation. In *Parallel Problem Solving from Nature, PPSN 2010, Part I*, pages 204–213. Springer, 2010.

[KLW15] Timo Kötzing, Andrei Lissovoi, and Carsten Witt. (1+1) EA on generalized dynamic OneMax. In *Foundations of Genetic Algorithms, FOGA 2015*, pages 40–51. ACM, 2015.

[KN13] Stefan Kratsch and Frank Neumann. Fixed-parameter evolutionary algorithms and the vertex cover problem. *Algorithmica*, 65:754–771, 2013.

[KST11] Timo Kötzing, Dirk Sudholt, and Madeleine Theile. How crossover helps in pseudo-Boolean optimization. In *Genetic and Evolutionary Computation Conference, GECCO 2011*, pages 989–996. ACM, 2011.

[KW20a] Timo Kötzing and Carsten Witt. Improved fixed-budget results via drift analysis. In *Parallel Problem Solving from Nature, PPSN 2020, Part II*, pages 648–660. Springer, 2020.

[KW20b] Martin S. Krejca and Carsten Witt. Lower bounds on the run time of the Univariate Marginal Distribution Algorithm on OneMax. *Theoretical Computer Science*, 832:143–165, 2020.

[Leh10] Per Kristian Lehre. Negative drift in populations. In *Parallel Problem Solving from Nature, PPSN 2010*, pages 244–253. Springer, 2010.

[Leh11] Per Kristian Lehre. Fitness-levels for non-elitist populations. In *Genetic and Evolutionary Computation Conference, GECCO 2011*, pages 2075–2082. ACM, 2011.

[Len20] Johannes Lengler. A general dichotomy of evolutionary algorithms on monotone functions. *IEEE Transactions Evolutionary Computation*, 24:995–1009, 2020.

[LM20] Johannes Lengler and Jonas Meier. Large population sizes and crossover help in dynamic environments. In *Parallel Problem Solving from Nature, PPSN 2020, Part I*, pages 610–622. Springer, 2020.

[LMS19] Johannes Lengler, Anders Martinsson, and Angelika Steger. When does hillclimbing fail on monotone functions: an entropy compression argument. In *Analytic Algorithmics and Combinatorics, ANALCO 2019*, pages 94–102. SIAM, 2019.

[LN17] Per Kristian Lehre and Phan Trung Hai Nguyen. Improved runtime bounds for the univariate marginal distribution algorithm via anti-concentration. In *Genetic and Evolutionary Computation Conference, GECCO 2017*, pages 1383–1390. ACM, 2017.

[LN18] Per Kristian Lehre and Phan Trung Hai Nguyen. Level-based analysis of the population-based incremental learning algorithm. In *Parallel Problem Solving From Nature, PPSN 2018*, pages 105–116. Springer, 2018.

[LN19a] Per Kristian Lehre and Phan Trung Hai Nguyen. On the limitations of the univariate marginal distribution algorithm to deception and where bivariate EDAs might help. In *Foundations of Genetic Algorithms, FOGA 2019*, pages 154–168. ACM, 2019.

[LN19b] Per Kristian Lehre and Phan Trung Hai Nguyen. Runtime analysis of the univariate marginal distribution algorithm under low selective pressure and prior noise. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 1497–1505. ACM, 2019.

[LOW19] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. On the time complexity of algorithm selection hyper-heuristics for multimodal optimisation. In *Conference on Artificial Intelligence, AAAI 2019*, pages 2322–2329. AAAI Press, 2019.

[LOW20] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. Simple hyper-heuristics control the neighbourhood size of randomised local search optimally for LeadingOnes$^+$. *Evolutionary Computation*, 28:437–461, 2020.

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

[LS15] Johannes Lengler and Nicholas Spooner. Fixed budget performance of the (1+1) EA on linear functions. In *Foundations of Genetic Algorithms, FOGA 2015*, pages 52–61. ACM, 2015.

[LS18] Johannes Lengler and Angelika Steger. Drift analysis and evolutionary algorithms revisited. *Combinatorics, Probability & Computing*, 27:643–666, 2018.

[LSW21] Johannes Lengler, Dirk Sudholt, and Carsten Witt. The complex parameter landscape of the compact genetic algorithm. *Algorithmica*, 83:1096–1137, 2021.

[LW12] Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. *Algorithmica*, 64:623–642, 2012.

[LW16] Andrei Lissovoi and Carsten Witt. MMAS versus population-based EA on a family of dynamic fitness functions. *Algorithmica*, 75:554–576, 2016.

[LY11] Per Kristian Lehre and Xin Yao. Crossover can be constructive when computing unique input-output sequences. *Soft Computing*, 15:1675–1687, 2011.

[LY12] Per Kristian Lehre and Xin Yao. On the impact of mutation-selection balance on the runtime of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 16:225–241, 2012.

[LZ19] Johannes Lengler and Xun Zou. Exponential slowdown for larger populations: the $(\mu + 1)$-EA on monotone functions. In *Foundations of Genetic Algorithms, FOGA 2019*, pages 87–101. ACM, 2019.

[Müh92] Heinz Mühlenbein. How genetic algorithms really work: mutation and hillclimbing. In *Parallel Problem Solving from Nature, PPSN 1992*, pages 15–26. Elsevier, 1992.

[Neu04] Frank Neumann. Expected runtimes of evolutionary algorithms for the eulerian cycle problem. In *Congress on Evolutionary Computation, CEC 2004*, pages 904–910. IEEE, 2004.

[NNS17] Samadhi Nallaperuma, Frank Neumann, and Dirk Sudholt. Expected fitness gains of randomized search heuristics for the traveling salesperson problem. *Evolutionary Computation*, 25, 2017.

[NOW09] Frank Neumann, Pietro S. Oliveto, and Carsten Witt. Theoretical analysis of fitness-proportional selection: landscapes and efficiency. In *Genetic and Evolutionary Computation Conference, GECCO 2009*, pages 835–842. ACM, 2009.

[NT10] Frank Neumann and Madeleine Theile. How crossover speeds up evolutionary algorithms for the multi-criteria all-pairs-shortest-path problem. In *Parallel Problem Solving from Nature, PPSN 2010, Part I*, pages 667–676. Springer, 2010.

[NW07] Frank Neumann and Ingo Wegener. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378:32–40, 2007.

[NW10] Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Springer, 2010.

[Och02] Gabriela Ochoa. Setting the mutation rate: scope and limitations of the 1/L heuristic. In *Genetic and Evolutionary Computation Conference, GECCO 2002*, pages 495–502. Morgan Kaufmann, 2002.

[OHY09] Pietro S. Oliveto, Jun He, and Xin Yao. Analysis of the (1+1)-EA for finding approximate solutions to vertex cover problems. *IEEE Transactions on Evolutionary Computation*, 13:1006–1029, 2009.

[OSW20] Pietro S. Oliveto, Dirk Sudholt, and Carsten Witt. A tight lower bound on the expected runtime of standard steady state genetic algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1323–1331. ACM, 2020.

[OSZ19] Pietro S. Oliveto, Dirk Sudholt, and Christine Zarges. On the benefits and risks of using fitness sharing for multimodal optimisation. *Theoretical Computer Science*, 773:53–70, 2019.

[OW15] Pietro S. Oliveto and Carsten Witt. Improved time complexity analysis of the simple genetic algorithm. *Theoretical Computer Science*, 605:21–41, 2015.

[Pos09] Petr Posik. BBOB-benchmarking a simple estimation of distribution algorithm with Cauchy distribution. In *Genetic and Evolutionary Computation Conference, GECCO 2009, Companion Material*, pages 2309–2314. ACM, 2009.

[Pos10] Petr Posík. Comparison of Cauchy EDA and BIPOP-CMA-ES algorithms on the BBOB noiseless testbed. In *Genetic and Evolutionary Computation Conference, GECCO 2010, Companion Material*, pages 1697–1702. ACM, 2010.

[Prü04] Adam Prügel-Bennett. When a genetic algorithm outperforms hill-climbing. *Theoretical Computer Science*, 320:135–153, 2004.

[QBJT19] Chao Qian, Chao Bian, Wu Jiang, and Ke Tang. Running time analysis of the $(1 + 1)$-EA for OneMax and LeadingOnes under bit-wise noise. *Algorithmica*, 81:749–795, 2019.

[RS14] Jonathan E. Rowe and Dirk Sudholt. The choice of the offspring population size in the $(1, \lambda)$ evolutionary algorithm. *Theoretical Computer Science*, 545:20–38, 2014.

[Rud96] Günter Rudolph. How mutation and selection solve long path problems in polynomial expected time. *Evolutionary Computation*, 4:195–205, 1996.

[RW20a] Amirhossein Rajabi and Carsten Witt. Evolutionary algorithms with self-adjusting asymmetric mutation. In *Parallel Problem Solving from Nature, PPSN 2020, Part I*, pages 664–677. Springer, 2020.

[RW20b] Amirhossein Rajabi and Carsten Witt. Self-adjusting evolutionary algorithms for multimodal optimization. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1314–1322. ACM, 2020.

[RW21a] Amirhossein Rajabi and Carsten Witt. Stagnation detection in highly multimodal fitness landscapes. In *Genetic and Evolutionary Computation Conference, GECCO 2021*. ACM, 2021. To appear.

[RW21b] Amirhossein Rajabi and Carsten Witt. Stagnation detection with randomized local search. In *Evolutionary Computation in Combinatorial Optimization, EvoCOP 2021*, pages 152–168. Springer, 2021.

[RWP08] J. Neal Richter, Alden H. Wright, and John Paxton. Ignoble trails – where crossover is provably harmful. In *Parallel Problem Solving from Nature, PPSN 2008*, pages 92–101. Springer, 2008.

[SGS11] Tom Schaul, Tobias Glasmachers, and Jürgen Schmidhuber. High dimensions and heavy tails for natural evolution strategies. In *Genetic and Evolutionary Computation Conference, GECCO 2011*, pages 845–852. ACM, 2011.

[SH87] Harold H. Szu and Ralph L. Hartley. Fast simulated annealing. *Physics Letters A*, 122:157–162, 1987.

[Sha02] Jonathan L. Shapiro. The sensitivity of PBIL to its learning rate, and how detailed balance can remove it. In *Foundations of Genetic Algorithms, FOGA 2002*, pages 115–132. Morgan Kaufmann, 2002.

[Sha05] Jonathan L. Shapiro. Drift and scaling in estimation of distribution algorithms. *Evolutionary Computing*, 13:99–123, 2005.

[Sha06] Jonathan L. Shapiro. Diversity loss in general estimation of distribution algorithms. In *Parallel Problem Solving from Nature, PPSN 2006*, pages 92–101. Springer, 2006.

[SN12] Andrew M. Sutton and Frank Neumann. A parameterized runtime analysis of evolutionary algorithms for the Euclidean traveling salesperson problem. In *AAAI Conference on Artificial Intelligence, AAAI 2012*. AAAI Press, 2012.

[SNN14] Andrew M. Sutton, Frank Neumann, and Samadhi Nallaperuma. Parameterized runtime analyses of evolutionary algorithms for the planar Euclidean traveling salesperson problem. *Evolutionary Computation*, 22:595–628, 2014.

[ST12] Dirk Sudholt and Christian Thyssen. A simple ant colony optimizer for stochastic shortest path problems. *Algorithmica*, 64:643–672, 2012.

[Sto06] Tobias Storch. How randomized search heuristics find maximum cliques in planar graphs. In *Genetic and Evolutionary Computation Conference, GECCO 2006*, pages 567–574. ACM, 2006.

[Sto07] Tobias Storch. Finding large cliques in sparse semi-random graphs by simple randomized search heuristics. *Theoretical Computer Science*, 386:114–131, 2007.

[Sto08] Tobias Storch. On the choice of the parent population size. *Evolutionary Computation*, 16:557–578, 2008.

[STW04] Jens Scharnow, Karsten Tinnefeld, and Ingo Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3:349–366, 2004.

[Sud05] Dirk Sudholt. Crossover is provably essential for the Ising model on trees. In *Genetic and Evolutionary Computation Conference, GECCO 2005*, pages 1161–1167. ACM, 2005.

[Sud13] Dirk Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17:418–435, 2013.

[Sud17] Dirk Sudholt. How crossover speeds up building block assembly in genetic algorithms. *Evolutionary Computation*, 25:237–274, 2017.

[Sud21] Dirk Sudholt. Analysing the robustness of evolutionary algorithms to noise: refined runtime bounds and an example where noise is beneficial. *Algorithmica*, 83:976–1011, 2021.

[Sut21] Andrew M. Sutton. Fixed-parameter tractability of crossover: steady-state GAs on the closest string problem. *Algorithmica*, 83:1138–1163, 2021.

[SW04] Tobias Storch and Ingo Wegener. Real royal road functions for constant population size. *Theoretical Computer Science*, 320:123–134, 2004.

[SW19] Dirk Sudholt and Carsten Witt. On the choice of the update strength in estimation-of-distribution algorithms and ant colony optimization. *Algorithmica*, 81:1450–1489, 2019.

[SZ10] Dirk Sudholt and Christine Zarges. Analysis of an iterated local search algorithm for vertex coloring. In *International Symposium on Algorithms and Computation, ISAAC 2010, Part I*, pages 340–352. Springer, 2010.

[Wit05] Carsten Witt. Worst-case and average-case approximations by simple randomized search heuristics. In *Symposium on Theoretical Aspects of Computer Science, STACS 2005*, pages 44–56. Springer, 2005.

[Wit06] Carsten Witt. Runtime analysis of the $(\mu + 1)$ EA on simple pseudo-Boolean functions. *Evolutionary Computation*, 14:65–86, 2006.

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

[Wit14] Carsten Witt. Revised analysis of the (1+1) EA for the minimum spanning tree problem. In *Genetic and Evolutionary Computation Conference, GECCO 2014*, pages 509–516. ACM, 2014.

[Wit19] Carsten Witt. Upper bounds on the running time of the univariate marginal distribution algorithm on OneMax. *Algorithmica*, 81:632–667, 2019.

[WJ07] Richard A. Watson and Thomas Jansen. A building-block royal road where crossover is provably essential. In *Genetic and Evolutionary Computation Conference, GECCO 2007*, pages 1452–1459. ACM, 2007.

[WQT18] Mengxi Wu, Chao Qian, and Ke Tang. Dynamic mutation based Pareto optimization for subset selection. In *Intelligent Computing Methodologies, ICIC 2018, Part III*, pages 25–35. Springer, 2018.

[YL97] Xin Yao and Yong Liu. Fast evolution strategies. In *Evolutionary Programming*, volume 1213 of *Lecture Notes in Computer Science*, pages 151–162. Springer, 1997.

[YLL99] Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3:82–102, 1999.