Introductory Tutorial: Theory for Non-Theoreticians

Benjamin Doerr
École Polytechnique, CNRS
Laboratoire d'Informatique (LIX)
Palaiseau, France
lastname@lix.polytechnique.fr

http://gecco-2019.sigevo.org/

GECCO '19 Companion, Prague, Czech Republic

ISBN 978-1-4503-6748-6/19/07 $15.00
DOI 10.1145/3319619.3323373

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 both of problem-specific algorithms and of randomized search heuristics like evolutionary algorithms. Major contributions to the latter include runtime analyses for evolutionary algorithms and ant colony optimizers, as well as the further development of the drift analysis method, in particular, multiplicative and adaptive drift. In the young area of black-box complexity, he proved several of the current best bounds.
- 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 Anne Auger, he edited the book *Theory of Randomized Search Heuristics*.

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, we try to answer these questions independent of a particular subarea of theory
- However, to not overload you with definitions and notation, we focus mostly on *evolutionary algorithms* for *discrete search spaces*
- Hence we intentionally omit examples from
  - genetic programming, ant colony optimizers, swarm intelligence, …
  - continuous optimization
- As said, this is for teaching purposes only. There is strong theory research in all these areas. All answers this tutorial give are equally valid for these areas

A Final Word 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

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:** Current Hot Topics in the Theory of EAs
  - EDAs (new), dynamic&noisy optimization (new), dynamic/adaptive parameter choices
- **Part V:** Concluding Remarks
- **Appendix:** glossary, references

Part I:

What is *Theory of EC*

- Definition of *theory of EC*
- Other notions of theory
- 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 with the EA
  - 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's 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

All this has its price… Possible drawbacks of theory results include:

- 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

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

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

(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

A mathematically proven result

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

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

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

a hidden all-quantifier: we claim the result for all $w_1, \dots, w_n \in \mathbb{R}$

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

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

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

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

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*: if the w_i are all positive, then "flipping a 0 to a 1 always increases the fitness" (monotonicity).
    - Are monotonic functions easy to optimize for a EAs (because you only need to collect 1s)?
    - No! Exponential runtimes can occur [DJS+13].
  - *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:
  - For all linear functions f, the Hamming distance $H(x, x^*)$ of x to the optimum x^* measures very well the quality of the search point x:
  - If the current search point is x, then the expected number $E[T_x]$ of iterations to find the optimum satisfies

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

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 $w_1 \ge w_2 \ge \dots \ge w_n > 0$
- 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 $1 + \frac{1}{n}$ to 2
  - 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

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

→ 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

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

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

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

"Drift analysis": Translate *expected progress* into *expected (run-)time*

- On the previous slide, this theorem was used with
  - $\delta = 1/4en$
  - $X_t = \tilde{f}(x^*) - \tilde{f}(x^{(t)})$
  - and the estimate $X_0 \le 2n$.
- **Bibliographical notes:** Artificial fitness functions very similar to this $\tilde{f}$ were already used in Droste, Jansen, and Wegener [DJW02] (conference version [DJW98]). 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 just shown
- Two types of 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 time (= number of fitness evaluations)

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

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

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

  → 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]
- We have some results on dynamic and noisy optimization

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 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
  - still an 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
  - 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 mislead the future development of the field
- 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
  - if $f(x)$ is not maximal, then by flipping a single bit of x you can get a better solution
- Misconception: Such functions are easy to optimize…
  - "because all you need is flipping single bits"
- Truth: There are functions $f: \{0,1\}^n \to \mathbb{R}$
  - without local optima, but
  - where all reasonable EAs with high probability need time exponential in n to find even a reasonably good solution [HGD94,Rud97,DJW98]
- 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* if the fitness increases whenever you flip a 0-bit to 1
  - special case of "no local optima": *each* neighbor with additional ones is better
- Misconception: Such functions are easy to optimize for standard EAs…
  - because already a simple hill-climber flipping single bits (randomized local search) does the job in time $O(n \log n)$
- Truth: There are (many) monotonically strictly increasing functions such that with high probability the (1+1) EA with mutation probability $16/n$ needs exponential time to find the optimum [DJS+13]
  - Lengler, Steger [LS18]: the $16/n$ can be lowered to $2.13\ldots/n$]
  - Lengler [Len18]: Essentially the same result holds for a broad class of mutation-based algorithms (independent of population sizes)

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 Designing EAs

- When designing an EA, you have to decide between a huge number of design choices: the basic algorithm, the operators and representations, and the parameter settings.
- Theory can help you with deep and reliable analyses of scenarios similar to yours
  - The question "what is a similar scenario" remains, but you have the same difficulty when looking for advice from experimental research
- Examples:
  - fitness-proportionate selection
  - edge-based representations in graph problems

    → more on these 2 on the next slides
  - when to use crossover (or not)
  - good values for mutation rate, population size, etc.

Designing EAs: Fitness-Proportionate Selection

- Fitness-proportionate selection has been criticized (e.g., because it is not invariant under re-scaling the fitness), but it is still used a lot.
- Theorem [OW15]: If you use
  - the Simple GA as proposed by Goldberg [Gol89] (generational GA, fitness-proportionate selection)
  - to optimize the OneMax test function $f: \{0,1\}^n \to \mathbb{R}; x \mapsto x_1 + \cdots + x_n$
  - with a population size $n^{0.2499}$ or less

  then with high probability the GA in any polynomial number of iterations does not create any individual that is 1% better than a random individual
- Interpretation: Most likely, fitness-proportionate selection makes sense only in rare circumstances in generational GAs
  - more difficulties with fitness-proportionate selection: [HJKN08, NOW09]

Designing EAs: 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 Designing EAs

- By analyzing rigorously 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 is 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 1: What is the right way to do mutation?
  - A thorough analysis how EAs optimize jump functions suggests a heavy-tailed mutation operator (instead of a binomial one) [best-paper award in the GECCO 2017 Genetic Algorithms track]
- Example 2 (maybe omitted for reasons of time): The $(1 + (\lambda, \lambda))$ GA
  - Invent an algorithm that profits from inferior search points

Example 1: Invent a New Mutation Operator

- **Short storyline:** The recommendation to flip bits independently with probability $1/n$ might be overfitted to ONEMAX or other easy functions.
- **Longer storyline** of this (longer) part:
  - A first-year Master project asks what is the best mutation rate to optimize *jump functions* (which are not "easy")
  - Surprise: for jump size m, the right mutation rate is m/n and this speeds-up things by a factor of roughly $(m/e)^m$
  - But: missing this optimal mutation rate by a small factor of $(1 \pm \varepsilon)$ increases the runtime by a factor of at least $\frac{1}{6}\, e^{m\varepsilon^2/5}$
  - Solution: design a parameter-less mutation operator where the Hamming distance of parent and offspring follows a power-law → solves all problems

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 general insights hold for other discrete search spaces.
- **General belief:** A good way of doing mutation is *standard-bit mutation*, that is, flipping each bit independently with some probability p ("mut. rate")
  - global: 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 $1/n$

Informal Justifications for $1/n$

- If you want to flip a particular single bit, then
  - a mutation rate of $1/n$ is the one that maximizes this probability
  - 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 a $1/n$ Mut. Rate

- Optimal mutation rates for (1+1) EA:
  - $\approx \frac{1}{n}$ for OneMax [Müh92; Bäc93]
  - $\approx \frac{1.59}{n}$ for LeadingOnes [BDN10]
  - $\approx \frac{1}{n}$ for all linear functions [Wit13]
  - monotonic functions [Jan07, DJSWZ13, LS18]:
    - $p = \frac{c}{n}, 0 < c < 1$, gives a $\Theta(n \log n)$ runtime on all monotonic functions with unique optimum,
    - $p = \frac{1}{n}$ gives $O(n^{1.5})$,
    - $p \geq \frac{2.13\ldots}{n}$ gives an exponential runtime on some monotonic functions.
- When $\lambda \leq \ln n$, then the optimal mutation rate for the $(1+\lambda)$ EA optimizing OneMax is $\approx \frac{1}{n}$ [GW17].

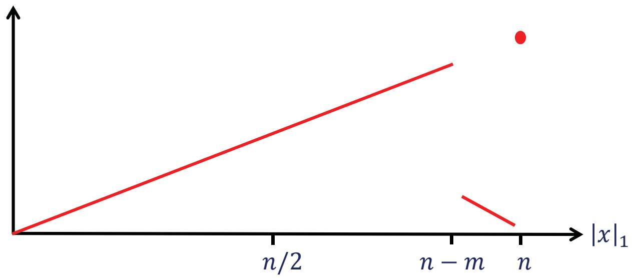
Theory supports using standard-bit mutation with mutation rate around $1/n$

Really?

- Can we really say that $1/n$ is good (at least "usually")?
- More provocative: Can we really say that *standard-bit mutation* 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
- Plan for the next few slides:
  - regard $JUMP_{m,n}$ functions (not unimodal)
  - observe something very different
  - design a new mutation operator
  - show that it is pretty good for many problems

Jump Functions [DJW02]

- $JUMP_{m,n}$: fitness of an n-bit string x is the number $|x|_1$ of ones, except if $|x|_1 \in \{n-m+1, \dots, n-1\}$, then the fitness is the number of zeroes.



- Observations:
  - All x with $|x|_1 = n-m$ form an easy to reach *local optimum*.
  - From there, only flipping (the right) m bits gives an improvement.
  - The unique *global optimum* is $x^* = (1, \dots, 1)$.

Runtime Analysis

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

$$T_p(m,n) = \Theta(p^{-m}(1-p)^{n-m}).$$

here and later: all implicit constants indep. of n and m

- Corollary (speed-up at least exponential in m): For any $p \in [2/n, m/n]$,
$$T_p(m,n) \le 6e^2\, 2^{-m}\, T_{1/n}(m,n).$$

- → **Clearly, here $1/n$ is not a very good mutation rate!**

- Proof of theorem uses standard theory methods:
  - upper bound: classic fitness level method
  - lower bound: argue that the runtime is essentially the time it takes to go from the local to the global optimum

Optimal Mutation Rates

- Theorem: Let $T_{opt}(m,n) \coloneqq \inf\{T_p(m,n) \mid p \in [0, 1/2]\}$. Then:
  - $T_{opt}(m,n) = \Theta(T_{m/n}(m,n))$.
  - If $p \ge (1+\varepsilon)(m/n)$ or $p \le (1-\varepsilon)(m/n)$, then
$$T_p(m,n) \ge \frac{1}{6}\exp\left(\frac{m\,\varepsilon^2}{5}\right) T_{opt}(m,n).$$
- In simple words: m/n is essentially the optimal mutation rate, but a small deviation increases the runtime massively.
- → Dilemma: To find a good mutation rate, you have to know how many bits you need to flip ☹
- Reason for the dilemma: When flipping bits independently at random (standard-bit mutation), then the Hamming distance $H(x,y)$ of parent and offspring is strongly concentrated around the mean
  - → exponential tails of the binomial distribution
- **→ Maybe standard-bit mutation is not the right thing to do?**

Solution: Heavy-tailed Mutation

- 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 (otherwise we do bad on easy functions)
- 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 distrib.]
  - 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.)

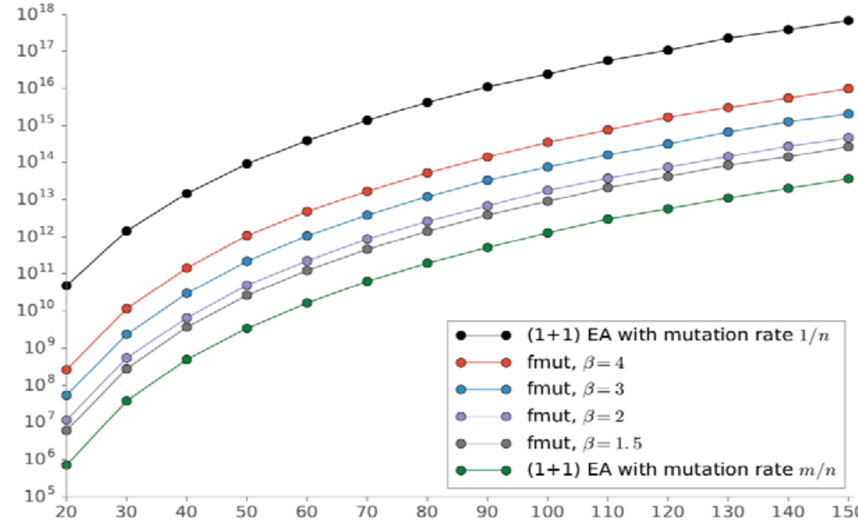
Heavy-tailed Mutation: Results

- 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 (for this m) optimal mutation rate!**
  - 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 $\varepsilon = \frac{2}{2m+1}$.
  - 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
  - $O(m^2 \log(n w_{max}))$ 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 [DJW02 (Trap); Wit05 (minimum makespan scheduling)]
  - $O(n^\beta\, 2^n)$ 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.

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 – 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 promising

Example 2: Invent New Algorithms (1/3)

- Theory can also, both via the deep understanding gained from proofs and by "theory-driven curiosity" invent new operators and algorithms. Here is one recent example:
- Theory-driven curiosity: Explain the following dichotomy!
  - the theoretically best possible black-box optimization algorithm $\mathcal{A}^*$ for OneMax (and all isomorphic fitness landscapes) needs only $O(n/\log n)$ fitness evaluations
  - all known (reasonable) EAs need at least $n \cdot \ln n$ fitness evaluations
- One explanation (from looking at the proofs): $\mathcal{A}^*$ profits from all search points it generates, whereas most EAs gain significantly only from search points as good or better than the previous-best
- Can we invent an EA that also gains from inferior search points?
  - YES [DDE13,GP14,DD15a,DD15b,Doe16,BD17], see next slides

New Algorithms (2/3)

- A simple idea to exploit inferior search points (in a (1+1) fashion):
  1. create λ mutation offspring from the parent by flipping λ random bits
  2. select the best mutation offspring ("mutation winner")
  3. create λ crossover offspring via a biased uniform crossover of mutation winner and parent, taking bits from mutation winner with probability $1/\lambda$ only
  4. select the best crossover offspring ("crossover winner")
  5. elitist selection: crossover winner replaces parent if not worse
- Underlying idea:
  - If λ is larger than one, then the mutation offspring will often be much worse than the parent (large mutation rates are destructive)
  - However, the best of the mutation offspring may have made some good progress (besides all destruction)
  - Crossover with parent repairs the destruction, but keeps the progress

New Algorithms (3/3)

- Performance of the new algorithm, called (1+(λ,λ)) GA:
  - solves OneMax in time (=number of fitness evaluations) $O\left(\frac{n \log n}{\lambda} + \lambda n\right)$, which is $O(n\sqrt{\log n})$ for $\lambda = \sqrt{\log n}$
  - the parameter λ can be chosen dynamically imitating the 1/5th rule, this gives an $O(n)$ runtime
  - experiments:
    - these improvements are visible already for small values of λ and small problem sizes n
    - [GP14]: good results for satisfiability problems
- Interpretation: Theoretical considerations can suggest new algorithmic ideas. Of course, much experimental work and fine-tuning is necessary to see how such ideas work best for real-world problems.

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:

Current Topics of Interest in the Theory of EC

- Estimation-of-distribution algorithms
- Dynamic and noisy optimization
- Dynamic/adaptive parameter choices

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).
  - Hope: more powerful algorithms by more expressive representations.
- Contrast: A parent x in the (1+1) EA corresponds to the frequency vector τ with $\tau_i = 1/n$ if $x_i = 0$ and $\tau_i = 1 - 1/n$ otherwise.
  - The (1+1) EA only admits the models $\{1/n, 1 - 1/n\}^n$.

Do We Profit From This Model Building?

- Problem: 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 step:
  - $\tau_i^{new} = \tau_i + 1/K$ 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.
- Common solution: Artificially cap the frequencies so that at all times $\tau_i \in [1/n, 1 - 1/n]$
  - Problem remains: If frequencies are mostly at the artificial boundary values, then our probabilistic model is not richer than for the (1+1) EA

Frequencies At Boundaries

- For a neutral bit, it takes an expected number of $O(K^2)$ iterations to reach a random boundary value (Zheng, Yang, D. [ZYD18])
  - → the problem of random movements is real!
- Witt [Wit17], Lengler, Sudholt, Witt [LSW18]: When optimizing OneMax, there are three regimes.
  - When K is small, then many frequencies reach the boundary values, but it is easy to bring them to the right boundary value (since the $1/K$ changes move the frequencies quickly) → $O(n \log n)$ runtime
  - When K is large, then the random movements of the frequencies are slow. The fitness moves the frequencies in parallel into the right direction → $O(n \log n)$ runtime
  - When K is "in the middle", then some frequencies reach boundaries, but it is costly to move them to the right value → no $O(n \log n)$ runtime is possible
- Disclaimer: I formulated things in the cGA language, some of these results are proven only for UMDA

Can We Avoid That Frequencies Drift to the Boundaries Without Good Reason?

- Friedrich, Kötzing, Krejca [FKK16]: "EDAs cannot be balanced and stable"
  - balanced: $E[\tau_i^{new}] = \tau_i$ when i is a neutral bit
  - stable: frequencies of neutral bits do not move quickly to boundaries
  - → if we want stability, we have to abandon balancedness
- The following algorithms are stable
  - stable-cGA [FKK16]: cGA with an artificially modified frequency update.
    - $O(n \log n)$ runtime on LeadingOnes
    - exponential runtime on OneMax (D., Krejca [DK18]).
  - sig-cGA [DK18]: regard a longer history, change frequencies only when there is sufficient evidence for it.
    - $O(n \log n)$ runtime on both LeadingOnes and OneMax
  - Binary differential evolution: Provably stable, but no fully rigorous runtime results [ZYD18]

Summary EDA-Theory

- Significant progress in the last 3 years.
- Main problem:
  - Frequencies move to boundaries in a random fashion.
  - This can lead to an undesired behavior (imitation of EAs) and to longer runtimes.
- Some suggestions for stable algorithms, but it is not clear yet how good they really are.

Hot Topic 2: 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

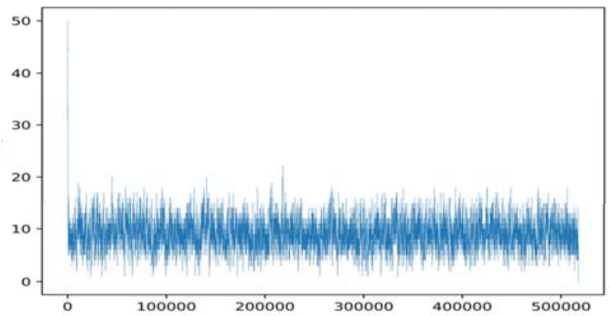
- First theory result (Droste [Dro02]):
  - OneMax function with optimum z: $OM_z(x) = |\{i \in [1..n] \mid x_i = z_i\}|$
  - Dynamic OneMax with 1-bit dynamics: in each iteration, with some small probability p the current optimum z is replaced by a random Hamming neighbor (=a random bit of z is flipped)
  - 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 $(3.77 + o(1))n^{1.39..ce+1}$ 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

Noisy Optimization

- Very roughly speaking, similar results hold for noisy optimization, see Droste [Dro04], Giessen, Kötzing [GK16], Qian, Bian, Jiang, Tang [QBJT17], Dang-Nhu et al. [DNDD+18], Sudholt [Sud18]

- Additional aspect: We can tolerate higher noise levels by
  - resampling (Akimoto, Astete-Morales, Teytaud [AMT15], Qian et al. [QBJT17], 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])

Summary Dynamic and Noisy Optim.

- 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 3: 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 $\log n$
- → First (artificial) example proving that dynamic parameter choices can be beneficial.

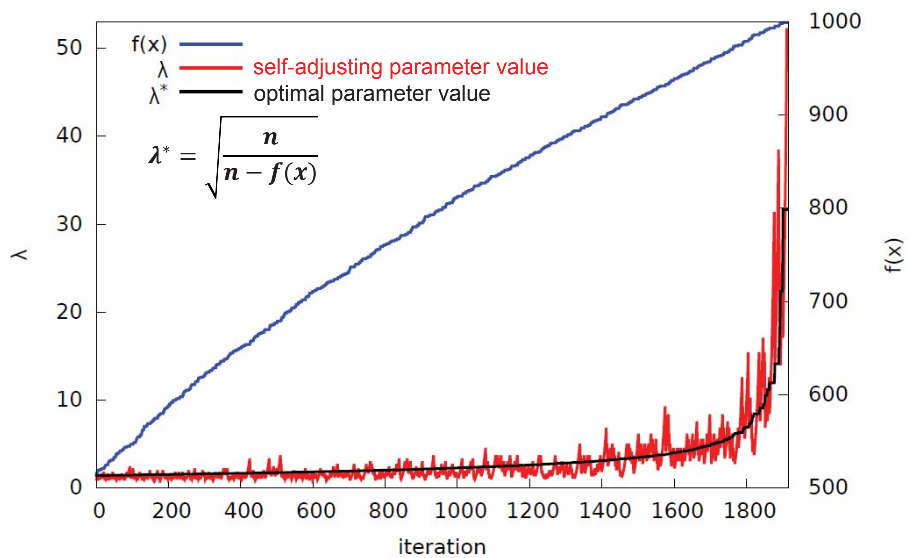
Theory for Dynamic Parameter Choices: Depending on the Fitness

- *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 $\boldsymbol{p = \frac{1}{f(x)+1}}$ gives $\approx 0.68\, n^2$ (optimal dynamic rate)
- *Fitness-dependent offspring pop. size* for the $(1 + (\lambda, \lambda))$ GA [DDE15]:
  - if you choose $\boldsymbol{\lambda = \sqrt{\frac{n}{n-f(x)}}}$, then the optimization time on OneMax drops from roughly $n\sqrt{\log n}$ to $O(n)$
- → 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 [DD15a,DD18a] ($F > 1$ 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 ($n = 1000$)

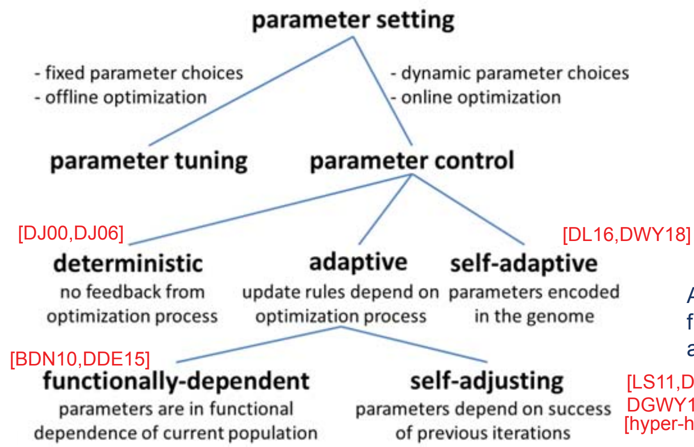


Theory for Dynamic Parameter Choices: Self-Adaptation

- In all dynamic parameter choices discussed so far, we added an extra mechanism onto the EA to control 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
- Dang, Lehre [DL16]: First proof that this can work (artificial example)
- D., Witt, Yang [DWY18]: Proof that self-adaptation can find the right mutation rate for the (1+λ) EA on OneMax (classic benchmark)
- → 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: D. and Doerr [DD18b]



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

Part V:

Conclusion

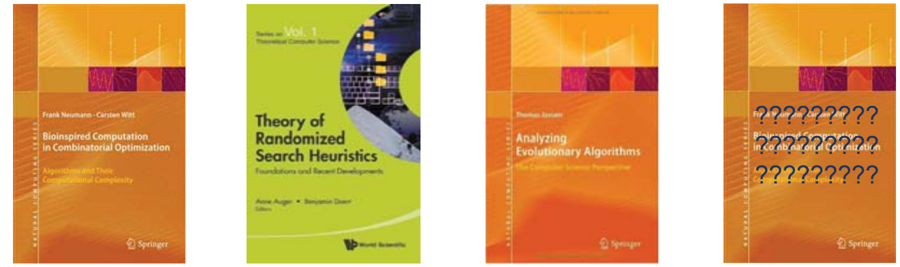
Summary

- 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

How to Use Theory in Your Work?

- Try to read theory papers (or listen to the talks in one of the 4 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 thinking”: take a 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

Recent 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 (2019?). Theory of Discrete Optimization Heuristics, Springer
  → Most chapters are already on the arxiv ←

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

Glossary of Terms Used in This Tutorial

- Big-Oh notation
- Optimization, global and local optima
- Discrete, pseudo-Boolean
- Runtime of an EA

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 $O(.)$

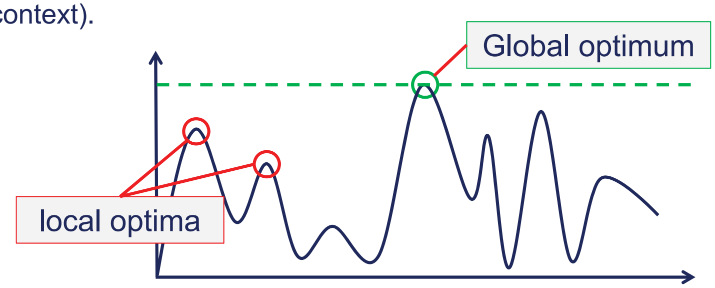
- Let us continue to use the example of the expected runtime $T(n) > 0$ of some algorithm on some problem that is defined for all problems sizes n (e.g., the expected runtime of the (1+1) EA on the n-dimensional ONEMAX function.
- Big-Oh notation allows to describe the *asymptotic behavior* of the runtime, 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 $O\big(f(n)\big)$ for some function $f: \mathbb{N} \to \mathbb{R}_{>0}$ if there is a constant c such $T(n) \le cf(n)$ 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 $T(n) \le cf(n)$ for all $n \in \mathbb{N}$.
- Asymptotic lower bound:
  - $T = \Omega(f)$ if there is a constant $c \ge 0$ such $T(n) \ge cf(n)$ 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 $\lim_{n\to\infty} \frac{T(n)}{f(n)} = \infty$

Optimization

- Optimization means that we try to find an optimum (maximum or minimum, depending on context) of a given function $f: S \to \mathbb{R}$.
  - x^* is a maximum of f if $f(x^*) \geq f(x)$ for all $x \in S$.
  - x^* is a minimum of f if $f(x^*) \leq f(x)$ for all $x \in S$.
- In practice, we often resort to finding a solution x with $f(x) \approx f(x^*)$.
- A local optimum is a solution $x \in S$ that is an optimum of f restricted to a small neighborhood around x (where "neighborhood" depends on the context).



Discrete and Pseudo-Boolean Optimization

- Discrete optimization: The search space S is a finite set.
  - Note: In principle, this allows to find an optimum by computing $f(x)$ for all $x \in S$. Naturally, we aim at more efficient algorithms. Still, the theoretical possibility to find a global optimum is a crucial difference to continuous optimization, where (generally) only approximations to global optima can be found.
- When $S = \{0,1\}^n$ and $f: \{0,1\}^n \to \mathbb{R}$, we call f a *pseudo-Boolean function.*
  - These are very common in evolutionary computation, since there are natural variation operators (mutation, crossover) for this representation.

Runtimes of Evolutionary Algorithms

- To make statements on the performance of an evolutionary algorithm (EA) in an implementation-independent manner, we regard as *runtime* (or *optimization time*) the number of fitness evaluations that the EA used until it queries for the first time an optimal solution.
  - This models that fact that in many EAs, the fitness evaluations are the most costly part.
- All EAs are *randomized algorithms*, i.e., they take random decisions during the optimization process. Consequently, the runtime (and almost everything) are *random variables*.

Definition: Runtime of an EA

- Let A be an EA, let f be a function to be maximized, and let $x^1, x^2, \ldots$ be the series of search points evaluated by A in a run when optimizing f (the x^i are also random variables). Then the runtime T of A on the problem f is defined by
$$T := \min\left\{ i \in \mathbb{N} \;\middle|\; f(x^i) = \max_{y \in \{0,1\}^n} f(y) \right\}.$$
- Several features of this random variable are interesting. We mostly care about the *expected runtime of an EA*. This number is the average number of function evaluations that are needed until an optimal solution is evaluated for the first time.
- Caution: sometimes runtime is stated in terms of *generations*, not *function evaluations.* Hence this runtime is smaller than ours by a factor equal to the number of search points evaluated per iteration.

Expected Runtimes – Caution!

- Caution: Regarding the expectation only can be misleading. For this reason, it is desirable to obtain more information about the runtime, e.g., its concentration behavior around the expectation.

- Misleading expectation: The expected runtime is large, when
  - occasionally the EA takes very very long,
  - but usually the EA is very efficient.
- In this case, the expectation does not tell you the full truth. For example, the EA with a restart strategy or with parallel runs is very efficient for this problem
- Example: The DISTANCE function regarded in [DJW02], see next slide

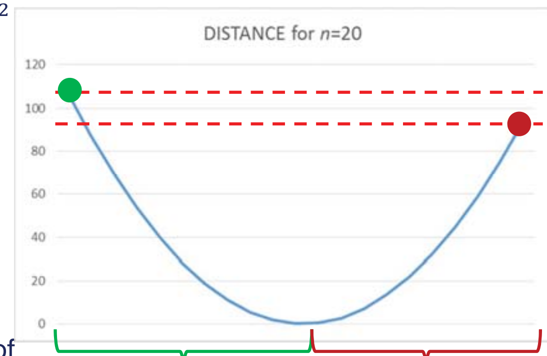
Expected Runtimes – Caution!

Formally,

$$\text{Distance}(x) := \left(\sum_{i=1,\dots,n} x_i - \frac{n}{2} - \frac{1}{3} \right)^2$$

We regard a simple hill climber (Randomized Local Search, RLS) which is

- initialized uniformly at random,
- flips one bit at a time,
- always accepts search points of best-so-far fitness



With probability (almost) 1/2, the algorithm has optimized DISTANCE after $O(n \log n)$ steps

With probability ~1/2 it does not find the optimum at all, thus having an infinite expected optimization time

Appendix B

List of References

References

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

[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.

[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.

[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. Full version available at http://arxiv.org/abs/1704.04366.

[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.

[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 $(1+(\lambda, \lambda))$ genetic algorithm on OneMax. In *Genetic and Evolutionary Computation Conference, GECCO 2015*, pages 1423–1430. ACM, 2015.

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

[DD18b] Benjamin Doerr and Carola Doerr. Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. *CoRR*, abs/1804.05650, 2018.

[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.

[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*. ACM, 2019. To appear.

[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. Optimal parameter choices via precise black-box analysis. In *Genetic and Evolutionary Computation Conference, GECCO 2016*, pages 1123–1130. ACM, 2016.

[DGWY17] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. The (1+λ) evolutionary algorithm with self-adjusting mutation rate. In *Genetic and Evolutionary Computation Conference, GECCO 2017*, pages 1351–1358. ACM, 2017. Full version available at http://arxiv.org/abs/1704.02191.

[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.

[DHK08] Benjamin Doerr, Edda Happ, and Christian Klein. Crossover can provably be useful in evolutionary computation. In *Genetic and Evolutionary Computation Conference, GECCO 2008*, pages 539–546. ACM, 2008.

[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.

[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.

[DJW98] Stefan Droste, Thomas Jansen, and Ingo Wegener. A rigorous complexity analysis of the (1 + 1) evolutionary algorithm for separable functions with boolean inputs. *Evolutionary Computation*, 6:185–196, 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 *Proceedings of the 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.

[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.

[DK18] Benjamin Doerr and Martin S. Krejca. Significance-based estimation-of-distribution algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 1483–1490. ACM, 2018.

[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.

[DL16] 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. Full version available at http://arxiv.org/abs/1703.03334.

[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.

[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. Full version available at http://arxiv.org/abs/1604.01088.

[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.

[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*. ACM, 2019. To appear.

[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.

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

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

[ELG+18] Hafsteinn Einarsson, Johannes Lengler, Marcelo Matheus Gauy, 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. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 1491–1498. ACM, 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.

[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.

[FM92] Stephanie Forrest and Melanie Mitchell. Relative building-block fitness and the building block hypothesis. In *Foundations of Genetic Algorithms, FOGA 1992*, pages 109–126. Morgan Kaufmann, 1992.

[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.

[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.

[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.

[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.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[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*. Natural Computing Series. Springer, 2013.

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

[JW99] Thomas Jansen and Ingo Wegener. On the analysis of evolutionary algorithms - a proof that crossover really can help. In *European Symposium on Algorithms, ESA 1999*, pages 184–193. Springer, 1999.

[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.

[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.

[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.

[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.

[Len18] Johannes Lengler. A general dichotomy of evolutionary algorithms on monotone functions. In *Parallel Problem Solving from Nature, PPSN 2018*, pages 3–15. Springer, 2018.

[LOW17] Andrei Lissovoi, Pietro Simone Oliveto, and John Alasdair Warwicker. On the runtime analysis of generalised selection hyper-heuristics for pseudo-boolean optimisation. In *Genetic and Evolutionary Computation Conference, GECCO 2017*, pages 849–856. ACM, 2017.

[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.

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

[LSW18] Johannes Lengler, Dirk Sudholt, and Carsten Witt. Medium step sizes are harmful for the compact genetic algorithm. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 1499–1506. ACM, 2018.

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

[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.

[NOW09] Frank Neumann, Pietro Simone 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.

[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.

[OHY09] Pietro Simone 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.

[OW15] Pietro Simone 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.

[QBJT17] 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. In *Genetic and Evolutionary Computation Conference, GECCO 2017*, pages 1399–1406. ACM, 2017.

[Rud97] Günter Rudolph. *Convergence Properties of Evolutionary Algorithms*. Verlag Dr. Kovăc, 1997.

[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.

[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.

[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.

[Sud18] Dirk Sudholt. On the robustness of evolutionary algorithms to noise: refined results and an example where noise helps. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 1523–1530. ACM, 2018.

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

[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.

[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. Fitness levels with tail bounds for the analysis of randomized search heuristics. *Information Processing Letters*, 114:38–41, 2014.

[Wit17] Carsten Witt. Upper bounds on the runtime of the univariate marginal distribution algorithm on onemax. In *Genetic and Evolutionary Computation Conference, GECCO 2017*, pages 1415–1422. ACM, 2017.

[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.

[ZYD18] Weijie Zheng, Guangwen Yang, and Benjamin Doerr. Working principles of binary differential evolution. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 1103–1110. ACM, 2018.