Tutorial: Theory for Non-Theoreticians

Benjamin Doerr
Ecole Polytechnique
Palaiseau, France
lastname@lix.polytechnique.fr

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

GECCO '17 Companion, Berlin, Germany
 978-1-4503-4939-0/17/07...$15.00
http://dx.doi.org/10.1145/3067695.3067713

Link to the Latest Version

- You can always find the latest version online at http://people.mpi-inf.mpg.de/~doerr/gecco17_tutorial_theory.pdf

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 Kiel University. 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 theory tutorials

History/Evolution of This Tutorial:

- GECCO 2013: Anne Auger and me proposed a real beginner's theory tutorial, but is was not accepted.
- PPSN 2014: Anne and me did a real beginner's theory tutorial
  - covering both discrete and continuous optimization
- GECCO 2015: The beginner's tutorial (proposed by Carola and me) was not accepted
- GECCO 2016 & CEC 2016: Carola and me did the beginner's tutorial
  - only covering discrete search spaces
- PPSN 2016: I did a beginner's tutorial
- This tutorial:
  - 20% overlap with PPSN 2014
  - 60% overlap with GECCO/CEC 2016
  - 85% overlap with PPSN 2016

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 (and which not)?
- What are current "hot topics" in the theory of EC?

Focus: EAs with Discrete Search Spaces

- We try to answer these questions independent of a particular subarea of theory
- However, to not overload you with definitions and notation, we focus on *evolutionary algorithms* on *discrete search spaces*
- Hence we intentionally omit examples from
  - genetic programming, estimation of distribution algorithms, ant colony optimizers, swarm intelligence, …
  - all subareas of 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
- This is still a young tutorial. To further improve it, 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

Outline of the Tutorial

- **Part I:** What We Mean by *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 examples showing how theory can have an impact
- **Part IV:** Current Hot Topics in the Theory of EAs
  - in particular: dynamic/adaptive parameter choices
- **Part V:** Concluding Remarks
- **Appendix:** glossary, references

Part I:

What We Mean by "Theory of EC"

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
  - make precise a statement on the performance of the EA solving this problem
  - prove this statement
- **Example:**
  Theorem: The (1+1) EA finds the optimum of the OneMax test 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: some parts of landscape analysis
- **"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: some parts of landscape analysis
- **"Theories":** Unproven claims that (mis-)guide our thinking
  - example: building block hypothesis

Why Do Theory? Because of Results

- Absolute guarantee that the result is correct
  - your 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
- Trigger for new ideas
  - clarifying nature of mathematics
  - playful nature of mathematicians

The Price for All This

All this has a certain a 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 get a weaker guarantee for all problem instances instead of a stronger one for the instances that show up in your real-world 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

Theory and Experiments: Complementary Results

THEORY	EXPERIMENTS
▪ cover all problem instances of arbitrary sizes → guarantee!	▪ only a finite number of instances of bounded size → have to see how representative this is
▪ proof tells you the reason	▪ only tells you numbers
▪ only models for real-world instances (realistic?)	▪ real-world instances
▪ limited scope, e.g., (1+1) EA	▪ everything you can implement
▪ limited precision, e.g., $O(n^2)$	▪ exact numbers
▪ implementation independent	▪ depends on implementation
▪ finding proofs can be difficult	▪ can be cheap (well, depends…)

→ Ideal: Combine theory and experiments. Difficulty: Get good theory people and good experimental people to talk to each other…

Part II:

A Guided Walk Through a Famous Theory Result

Outline for This Part

- 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 discuss typical shortcomings of theory results

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 very 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

What is Cool About This 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 it in one page now, but only because we developed deep analysis techniques (artificial fitness functions, drift analysis)
- **No "easy" explanation**
  - *monotonicity* is not enough: if the w_i are all positive, then "flipping a 0 to a 1 always increases the fitness" (monotonicity).
    - easy explanation that is not true: monotonic functions are easy to optimize for an EA – disproved in [DJS+13]
  - *separability* is not enough
    - a linear function can be written as a sum of functions f_i such that the f_i depend on disjoint sets of bits
    - not true that the optimization time of such a sum is not much more than the worst optimization time of the summands (because the independent f_i are optimized in parallel) – disproved in [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 or BV 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. Example:
    - $\mathrm{BV}(10 \dots 0) = 2^{n-1}$, distance to optimum is $n-1$
    - $\mathrm{BV}(01 \dots 1) = 2^{n-1} - 1$, distance to opt. 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:
  - If the current search point of the (1+1) EA is x, then the optimum is found within an expected number $E[T_x]$ of iterations that satisfies

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

independent of f

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 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 $1 + \frac{1}{n}$ 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

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

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 $E[X_{t+1}] \leq (1-\delta) E[X_t]$.
  Let $T := \min\{t \in \mathbb{N} \mid X_t = 0\}$. Then $E[T] \leq \frac{1 + \ln X_0}{\delta}$.
- On the previous slide, this theorem was used with
  - $\delta = 1/4en$
  - $X_t = \tilde{f}(x^*) - \tilde{f}(x^{(t)})$
  - and the estimate $X_0 \leq 2n$.

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

- **Bibliographical notes:** Artificial fitness functions very similar to this $\tilde{f}$ were already used in [DJW02] (conference version [DJW98]). Drift analysis ("translating progress into runtime") was introduced to the field in [HY01] to give a simpler proof of the [DJW02] result. A different approach was given by [Jäg08]. The multiplicative drift theorem [DJW12] (conference version [DJW10]) proves the [DJW02] result in one page and is one of the most-used drift theorems today.

What is Uncool About The Linear Functions Result?

- An unrealistically simple EA: the (1+1) EA
- Linear functions are artificial test function only
- 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

→ We discuss these points on the following 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.

Maybe Uncool: Only the Simple (1+1) EA

- This was, at that time (2002), the absolute maximum that was possible when asking for a proven result.
- Today, we know (a little bit) more. E.g., 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 the theory community will make progress towards more complicated EAs

Maybe Uncool: Only Linear Functions

- Again, this was the starting point. Today, we know how the (1+1) EA (and some other algorithms) compute
  - Eulerian cycles [Neu04,DHN06,DKS07,DJ07]
  - shortest paths [STW04,DHK07,BBD+09]
  - minimum spanning trees [NW07,DJ10,Wit14]
  - and many other "easy" 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 are optimistic that we will enlarge the set of problems we understand. However, like in many fields, it is also clear that "theory will always be behind"; that is, it will take quite some time until theoretical analyses become available for typical algorithms used in practice and realistic real-world problems

Maybe Uncool: $O(n \log n)$, Large Constants

- Having only asymptotic results is a typical price for proven results (also in the classic algorithms field).
- There is the general experience that often a proven "$O(n \log n)$" in fact means "roughly $cn \log n$" for a small constant c, which can, e.g., be obtained from experiments
- We know more now [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 hard 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 strengths and limitations of theory results

Part III:

How Theory Can Contribute to a Better Understanding of EAs

Outline for This Part

3 ways how theory can help understanding and improving 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 EA, 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" or for the problems we just have in mind, 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: 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 already a simple hill-climber flipping single bits (randomized local search) does the job
- Truth: There are functions $f: \{0,1\}^n \to \mathbb{R}$ without local optima 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: 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" where each neighbor with more 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 $O(n \log n)$
- [DJS+13]: There is a monotonically strictly increasing function such that with high probability the (1+1) EA with mutation probability $16/n$ needs exponential time to find the optimum
  - very different from linear functions with positive weights: $O(n \log n)$ time

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 distinguish between what is a rule of thumb and what is a proven fact
  - 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
- 2 examples:
  - fitness-proportionate selection
  - edge-based representations for optimization problems in graphs

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] (fitness-proportionate selection, comma 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 a polynomial number of iterations does not create any individual that is 1% better than a random individual
- Interpretation: Most likely, fitness-proportionate selection and comma selection together make sense only in rare circumstances
  - 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

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 that we should use mutation operators such that the Hamming distance between parent and offspring follows a heavy-tailed distribution (and not a binomial one)
    → this GECCO, best-paper nominee in the GA track
- Example 2: The $(1 + (\lambda, \lambda))$ GA
  - Invent an algorithm that truly profits also 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 unimodal functions
- **Longer storyline** of this (longer) part:
  - 4 young researchers ask themselves what is the right mutation rate to optimize jump functions (which are not unimodal)
  - surprise: for jump size m, the right mutation rate is m/n and this speeds-up things by a factor of $(m/e)^m$
  - but: missing this optimal mutation rate by a factor of $(1 \pm \varepsilon)$ increases the runtime again by a factor of at least $\frac{1}{6}\, e^{m\varepsilon^2/5}$
    - reason: With standard-bit mutation, the Hamming distance between parent and offspring is strongly concentrated
  - solution: design a mutation operator where this Hamming distance follows a power-law (not strongly concentrated)

General Belief on Mutation

- **Disclaimer:** In this part, we only deal with bit-string representations, that is, the search space is $\{0,1\}^n$ for some n.
- **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 ("convergence")
- **Recommendation:** Use a small mutation rate like $1/n$
  - nature-inspired
  - reduces the destructive effect of mutation: since mutation is undirected, in average it moves you towards search points with Hamming distance $n/2$ from the optimum

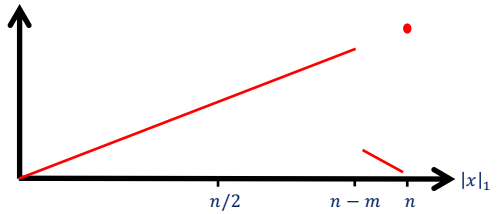
Proven Results for Mutation

- The mutation rate of $1/n$ is the one that maximizes the probability to flip a single bit → it minimizes the waiting time for "finding the last bit", that is, flipping the missing bit when one is at a Hamming neighbor of the optimum.
- The optimal mutation rate for the (1+1) EA optimizing OneMax or linear functions [Wit13] is $\frac{1+o(1)}{n}$.
- When $\lambda \leq \ln n$, then the optimal mutation rate for the (1+λ) EA optimizing OneMax is $\frac{1+o(1)}{n}$ [GW15].
- The optimal mutation rate for the (1+1) EA optimizing the LeadingOnes test function is $\frac{1.59\ldots+o(1)}{n}$ [BDN10].

Object of This Study: Jump Functions

Jump functions [DJW02]:

- $JUMP_{m,n}$: fitness of an n-bit string x is the number $|x|_1$ of ones, except if $|x|_1 \in \{n-m+1, \ldots, n-1\}$, 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 $x^* = (1 \ldots 1)$.

Runtime Analysis

- In the following, we always regard how the (1+1) EA optimizes the jump function $JUMP_{m,n}$.
  - Motivation: This is the most simple mutation-based algorithm, so we feel that this is the cleanest way to study different ways to do mutation.
  - Side remark: From the proofs, it is easy to see that most results also hold for many other elitist mutation-based algorithms.
- Theorem: Let $T_p(m,n)$ denote the expected optimization time of the (1+1) EA optimizing $JUMP_{m,n}$ with mutation rate $p \leq 1/2$. Then
$$T_p(m,n) = \Theta(p^{-m}(1-p)^{n-m}).$$
- Corollary (speed-up at least exponential in m): For any $p \in [2/n, m/n]$,
$$T_p(m,n) \leq 6e^2\, 2^{-m}\, T_{1/n}(m,n).$$
- → **Clearly, $1/n$ is not the optimal mutation rate!**

Optimal Mutation Rates

- Theorem: Let $T_{opt}(m,n) := \inf\{T_p(m,n) \mid p \in [0,1/2]\}$.
  - Then $T_{opt}(m,n) = \Theta\left(T_{m/n}(m,n)\right)$.
  - 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. A small deviation from it increases the runtime massively.
- Dilemma: To find the optimal mutation rate, we need to know beforehand which numbers of bits we need to flip during a typical run of the EA ☹ [not a realistic hope for any real problem]

Understanding the Dilemma

- Reason for the dilemma: When flipping bits independently at random, then the Hamming distance of parent and offspring is strongly concentrated around the mean
  - → exponential tails of the binomial distribution
- We made this dilemma very precise for jump functions, but of course it is the same for all other optimization problems where, at some time, a certain (larger) number of bits needs to be flipped.
- It seems that
  - previous theory work has not detected this, since it mostly regarded unimodal functions (where flipping single bits is usually best)
  - previous experimental work has not detected this … (I don't know why. Lack of the "clarifying nature of mathematics"?)

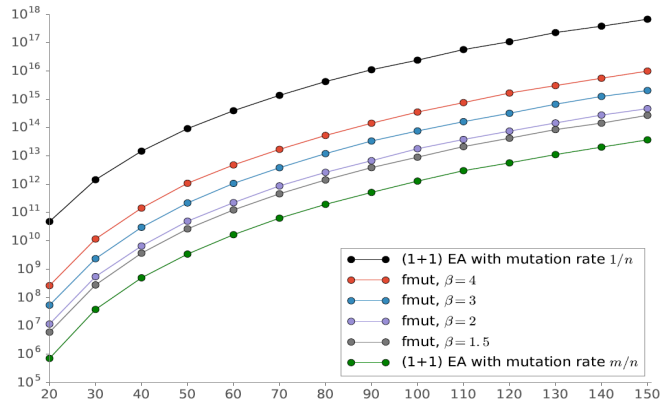
Solution: Heavy-tailed Mutation

- What do we need?
  - Larger numbers of bits flip with reasonable probability
  - No strong concentration
  - 1-bit flips occur with constant probability (we don't want a massive slow-down for easy problems)
- NEW: Heavy-tailed mutation (with parameter $\beta > 1$):
  - 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 $\Theta\left(k^{-\beta}\right)$
  - → no exponential tails around the mean ☺
  - $\Pr[1 \text{ bit flips}] = \Theta(1)$, e.g., ≈32% when $\beta = 1.5$ (≈37% for classic mut.)

Heavy-tailed Mutation: Results

- Theorem: The expected optimization time on $JUMP_{m,n}$ satisfies
  $$O\left(m^{\beta-0.5}\, T_{opt}(m,n)\right).$$
- In simple words: We're at most 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 $m^{\Theta(m)}$.

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

- The "only" reason for these speed-ups is that we increase the probability for a k-bit flip from roughly $\frac{1}{e \cdot k!}$ to roughly $k^{-\beta}$. Hence it is fair to suspect that similar advantages are also observed for other problems where multi-bit flips are useful.
- Example: 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 $m \in \mathbb{N}$ 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 $T_{n,\varepsilon}$ iterations, where $T_{n,\varepsilon}$ is some number in $\Theta(n^{2m+2})$.
  - The (1+1) EA with heavy-tailed mutation does the same in expected time $\big(1+o(1)\big)\, e\, \zeta(\beta) \left(\frac{e}{m}\right)^m m^{\beta-0.5}\ T_{n,\varepsilon}$.

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

Summary Heavy-tailed Mutation

- Key working principle:
  - 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 1-bit flips, but $\binom{n}{m}$ time to find the one necessary m-bit flip
- There is good reason to believe that heavy-tailed mutation is useful for a wide range of multi-modal problems.

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], 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" does not mean "easy"
- giving hints how to choose parameters, representations, operators, and algorithms
  - e.g., how useful is crossover when we hardly find an example where is provably improves things?
- inventing new representations, operators, and algorithms; this is fueled by the deep understanding gained in theoretical analyses and "theory-driven curiosity"

Part IV:

Current Topics of Interest in Theory of EC

What We Currently Try to Understand

- Precise runtime guarantees
- Dynamic/adaptive parameter choices
- Population-based EAs
- Dynamic optimization, noisy environments
- Non-elitism
- Black-box complexity

- Examples for all will be given on the next slides.

- Parallel to these topics, we study also methodical questions (e.g., *drift analysis*), but these are beyond the scope of this tutorial

Precise Runtime Guarantees

- Theory results can give advice on how to chose the parameters of an EA
  - Example: the discussion on optimal mutation rates in part III

- The more precisely we know the runtime (e.g., upper *and* lower bounds for its expected value), the more precise recommendations we can give for the right parameter choice (e.g., m/n instead of $\Theta(m/n)$)
  - in practice, constant factors matter ☺

- Challenge: For such precise runtime bounds often the existing mathematical tools are insufficient
  - in particular, tools from classic algorithms theory are often not strong enough, because in that community (for several good reasons) there is no interest in bounds more precise than $O(\dots)$.

Dynamic Parameter Choices

- Instead of fixing a parameter (mutation rate, population size, …) once and forever (*static* parameter choice), it might be preferable to use parameter choices that change
  - depending on time
  - depending on the current state of the population
  - depending on the performance in the past

- Hope:
  - different parameter settings may be optimal early and late in the optimization process
  - with self-adjusting parameters, we do not need to know the optimal parameters beforehand, but the EA finds them itself

- 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 [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 $\log n$

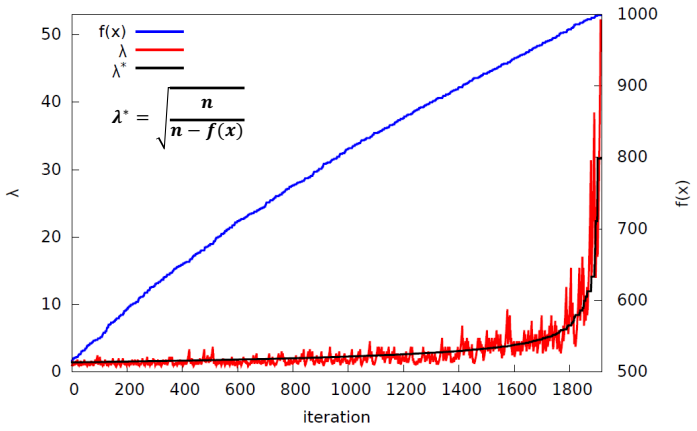
Theory for Dynamic Parameter Choices: Depending on the Fitness

- *Fitness-dependent mutation rate* [BDN10]: When optimizing the LeadingOnes test function $LO\colon \{0,1\}^n \to \{0,\dots,n\}$ 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 dyn. mut. rate)
- *Fitness-dependent offspring pop. size* for the $(1+(\lambda,\lambda))$ GA [DDE13]:
  - 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 $O(n)$
- Interpretation: Fitness-dependent parameters can pay off. It is hard to find the optimal dependence, but others give improvements as well (→ proofs)

Theory for Dynamic Parameter Choices: Success-based Dynamics

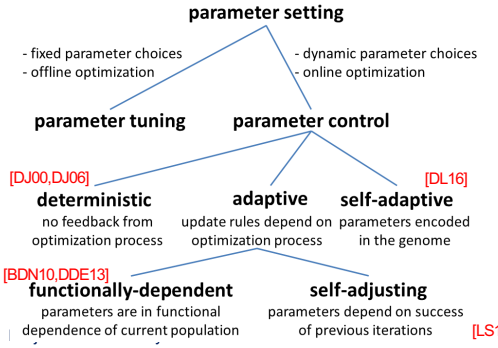
- *Success-based* choice of island number: You can reduce of the parallel runtime (but not the total work) of an island model when choosing the number of islands dynamically [LS11]:
  - double the number of islands after each iteration without fitness gain
  - half the number of islands after each improving iteration
- A *success-based* choice (1/5-th rule) of λ in the (1+(λ,λ)) GA automatically finds the optimal mutation strength [DD15a]
  - $\lambda := \sqrt[4]{F}\,\lambda$ after each iteration without fitness gain, $F > 1$ a constant
  - $\lambda := \lambda/F$ after each improving iteration
  - Important that F is not too large and that the fourth root is taken (→ 1/5-th rule). The doubling scheme of [LS11] would not have worked
- Simple mechanisms to automatically find the current-best parameter setting (note: this is great even when the optimal parameter does not change over time)

Example Run Self-Adjusting $(1+(\lambda,\lambda))$ GA



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
  - NEW tutorial "Non-static parameter choices" (C. Doerr) at this GECCO



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

Population-Based EAs

- Population-based: using a non-trivial (≥ 2) population of individuals
- In practice, non-trivial populations are often employed
- In theory,
  - no convincing evidence (yet) that larger populations are generally beneficial (apart from making the algorithm easy to run on parallel machines)
  - the typical result is "up to a population size of …, the total work is unchanged, for larger population sizes, you pay extra"
  - some evidence (on the level of artificially designed examples) that populations help in dynamic or noisy settings
  - not many methods to deal with the complicated population dynamics
- Big open problem: Give rigorous advice how to profitably use larger populations (apart allowing parallel implementations)
  - and devise methods to analyze such algorithms

Dynamic Optimization

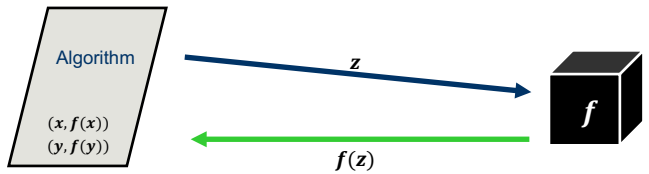
- Dynamic optimization: Optimization under (mildly) changing problem data
- Question: How well do EAs find and track the moving optimum?
- First theory result [Dro02]: dynamic version of OneMax where the optimum changes (by one bit) roughly every K iterations
  - If $K = n/\log n$ or larger, then a polynomial number of iterations suffices to find or re-find the current optimum
    - K can be quite a bit smaller than the usual $en \ln n$ runtime!
  - First indication that EAs do well in dynamic optimization
- More recent results: Many (artificial) examples showing that populations, diversity mechanisms, island models, or ant colonies help finding or tracking dynamically changing optima [JS05,KM12,OZ15,LW14,LW15,…]
- Two main open problems: (i) What are realistic dynamic problems?
  - (ii) What is the best way to optimize these?

Non-Elitism

- Most EAs analyzed in theory use *truncation selection,* which is an *elitist selection* = you cannot lose the best-so-far individual
- Mostly negative results on non-elitism are known. For example, [OW15] proves that the Simple Genetic Algorithm using *fitness-proportional selection* is unable to optimize OneMax efficiently [see above]
- Strong Selection Weak Mutation (SSWM) algorithm [PPHST15], inspired by an inter-disciplinary project with populations-genetics:
  - worsening solutions are accepted with some positive probability
  - for improving offspring, acceptance rate depends on the fitness gain
  - Examples are given in [PPHST15] for which SSWM outperforms classic EAs
- Black-box complexity view: there are examples where *any* elitist algorithm is much worse than a non-elitist algorithm [DL15]
- State of the art: Not much real understanding apart from sporadic results. The fact that non-elitism is used a lot in EC practice asks for more work.

Limits of EC: Black-Box Complexity

- EAs are *black-box algorithms*: they learn about the problem at hand only by evaluating possible solutions



- What is the price for such a problem-independent approach?
  →This is the main question in black-box complexity.
- In short, the black-box complexity of a problem is the minimal number of function evaluations that are needed to solve it
  - = performance of the best-possible black-box algorithm

Black-Box Complexity Insights

- Unified lower bounds: The black-box complexity is a lower bound for the runtime of *any* black-box algorithm: all possible kinds of EAs, ACO, EDA, simulated annealing, …
- Specialized black-box models allow to analyze the impact of algorithmic choices such as type of variation in use, the population size, etc.
- Example result: [LW12] proves that every *unary unbiased* algorithm needs $\Omega(n \log n)$ function evaluations to optimize OneMax
  - unary: mutation only, no crossover
  - unbiased: symmetry in
    - bit-values 0 and 1
    - bit positions 1,2,…,n

  → Result implies that algorithms using fair mutation as only variation cannot be significantly more efficient on OneMax than the (1+1) EA

Black-Box Complexity vs. Games – Where EA Theory Meets Classic CS

- Black-box algorithms are strongly related to Mastermind-like guessing games:
  - algorithm guesses a search point
  - opponent reveals the fitness
- Such guessing games have a long history in classic computer science due to applications in security and privacy
- We have several (hidden) black-box complexity publications in classic CS venues (including a paper to appear in the *Journal of the ACM*)
  - EC theory meets classic theory
  - a chance to get the classic CS community interested in our field!

Part V:

Conclusion

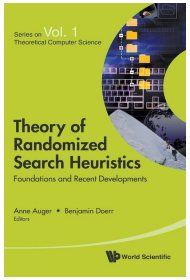
Summary

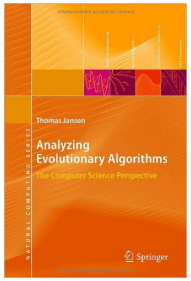
- Theoretical research gives deep insights in the working principles of EC, with results that are of a different nature than in experimental work
  - "very true" (=proven), but often apply to idealized settings only
  - *for all* instances and sizes, …, but sometimes less precise
    - often only asymptotic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

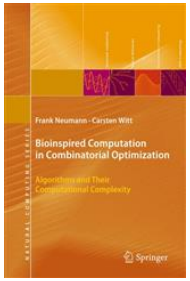
How to Use Theory in Your Work?

- Try to read theory papers, 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)







- Auger/Doerr (2011). Theory of Randomized Search Heuristics, World Scientific
- Jansen (2013). Analyzing Evolutionary Algorithms, Springer
- Neumann/Witt (2010). Bioinspired Computation in Combinatorial Optimization, Springer

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

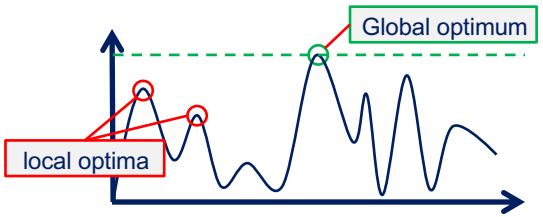
Discrete and Pseudo-Boolean Optimization

In this tutorial we are mostly interested in the optimization of problems of the type $f: \{0,1\}^n \to \mathbb{R}$

- Problems $f: S \to \mathbb{R}$ with finite search space S are called *discrete optimization problems*
  (in contrast to *continuous problems* $f: \mathbb{R}^n \to \mathbb{R}$ or, more generally $f: S \to \mathbb{R}$ with continuous S)
- When $S = \{0,1\}^n$ and $f: \{0,1\}^n \to \mathbb{R}$, we call f a *pseudo-Boolean function*
- Please note: don't get fooled! Even if optimizing a function $f: \{0,1\}^n \to \mathbb{R}$ may look harmless, a HUGE range of problems (even NP-hard ones like Max-SAT and many others!) can be expressed this way

What we Mean by "Optimization"

- Recall: we assume that we aim at optimizing a function $f: \{0,1\}^n \to \mathbb{R}$
- For this tutorial "optimization" = *maximization*,
  that is, we aim at finding a bit string $x = (x_1, \dots, x_n)$ such that $f(x) \geq f(y)$ for all $y \in \{0,1\}^n$
- Note in particular: we are not interested in this tutorial in identifying local optima, only the *global best solution(s)* are interesting for us



Expected Runtimes – Introduction

- All EAs are *randomized algorithms*, i.e., they use random decisions during the optimization process (for example, the *variation step*, i.e., the step in which new search points are generated, is often based on random decisions---we will discuss this in more detail below)
- Our object of interest, the *runtime of EAs*, is the number of function evaluations that an EA needs until it queries for the first time an optimal solution. Since EAs are randomized algorithms, their runtime is a random variable

Expected Runtimes – Definition

- Formally, let A be an EA, let f be a function to be optimized and let $x^1, x^2, \ldots$ be the series of search points queried by A in one run of optimizing f. The search points x^i are random and so is the series of fitness values $f(x^1), f(x^2), \ldots$. The runtime T 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 (1/2): sometimes runtime is measured in terms of *generations*, not *function evaluations*
- Caution (2/2): Regarding expectation only can be misleading (see next slide for an example), this is why we typically study also other features of the runtime, such as its concentration

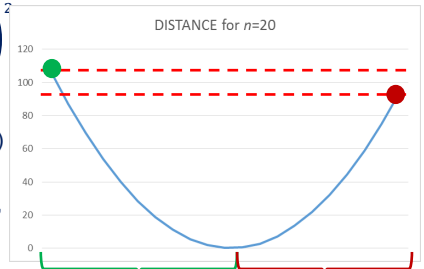
Expected Runtimes – Caution!

- The expected runtime does not always tell you the full truth:
There are functions for which the expected runtime is very large but which can be optimized in a small number of steps with a fair probability.
Example: The DISTANCE function regarded in [DJW02], see next slide

Expected Runtimes – Caution!

Formally,

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



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

Big-O Notation, aka Landau Notation (1/2)

- The "big-O" notation is used in algorithms theory to classify the order at which the running time of an algorithm grows with the size of the input problems
- In our example, it says that "The expected runtime of the (1+1) EA on any linear function with weights $\neq 0$ is $\Theta(n \log n)$."
- $\Theta(n \log n)$ means that the expected runtime of the (1+1) EA on f is
  - $O(n \log n)$, that is, there exists a constant $C > 0$ such that for all n the expected runtime is at most $Cn \log n$
  - $\Omega(n \log n)$, that is, there exists a constant $c > 0$ such that for all n the expected runtime is at least $cn \log n$
- That is, there exist constants $0 < c < C$ such that
$$cn \log n \leq E(T_{(1+1)EA,f}) \leq Cn \log n$$

Big-O Notation, aka Landau Notation (2/2)

Further frequently used notation

- $f \in o(n)$ if f grows slower than linear. Formally:
  for all constants $0 < c$ there exists a n_0 such that for all $n > n_0$: $f(n) \leq cn$
- $f \in \omega(n)$ if f grows faster than linear. Formally:
  for all constants $0 < c$ there exists a n_0 such that for all $n > n_0$: $f(n) \geq cn$

Appendix B

List of References

References (1/6)

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

[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 functions. In *Proc. of Foundations of Genetic Algorithms (FOGA)*, pages 59–66. ACM, 2009.

[BDN10] S. Böttcher, B. Doerr, and F. Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Proc. of Parallel Problem Solving from Nature (PPSN)*, 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 *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, 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 *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 1423–1430. ACM, 2015.

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

[DDY16] Benjamin Doerr, Carola Doerr, and Jing Yang. Optimal parameter choices via precise black-box analysis. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 2016. To appear.

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

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

[DHN06] Benjamin Doerr, Nils Hebbinghaus, and Frank Neumann. Speeding up evolutionary algorithms through restricted mutation operators. In *Proc. of Parallel Problem Solving from Nature (PPSN)*, volume 4193 of *Lecture Notes in Computer Science*, pages 978–987. Springer, 2006.

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

References (2/6)

[DJ10] Benjamin Doerr and Daniel Johannsen. Edge-based representation beats vertex-based representation in shortest path problems. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 758–766. 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 *Proc. of Foundations of Genetic Algorithms (FOGA)*, 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 monotonic functions. *Evolutionary Computation*, 21:1–27, 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.

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

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

[DL15] Carola Doerr and Johannes Lengler. Elitist black-box models: Analyzing the impact of elitist selection on the performance of evolutionary algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 839–846. ACM, 2015.

References (3/6)

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

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

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

[Dro03] Stefan Droste. Analysis of the (1+1) EA for a dynamically bitwise changing OneMax. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, volume 2723 of *Lecture Notes in Computer Science*, pages 909–921. Springer, 2003.

[DSW13] Benjamin Doerr, Dirk Sudholt, and Carsten Witt. When do evolutionary algorithms optimize separable functions in parallel? In *Proc. of Foundations of Genetic Algorithms (FOGA)*, pages 51–64. ACM, 2013.

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

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

[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.

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

[FW04] Simon Fischer and Ingo Wegener. The ising model on the ring: Mutation versus recombination. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, volume 3102 of *Lecture Notes in Computer Science*, pages 1113–1124. Springer, 2004.

[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 *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 785–792. ACM, 2014.

References (4/6)

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

[HGD94] Jeff Horn, David Goldberg, and Kalyan Deb. Long path problems. In *Proc. of Parallel Problem Solving from Nature (PPSN)*, LNCS 866, 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 *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 953–960. ACM, 2008.

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

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

[JOZ13] Thomas Jansen, Pietro Simone Oliveto, and Christine Zarges. Approximating vertex cover using edge-based representations. In *Proc. of Foundations of Genetic Algorithms (FOGA)*, 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 *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 841–848. ACM, 2005.

[JW99] Thomas Jansen and Ingo Wegener. On the analysis of evolutionary algorithms - A proof that crossover really can help. In *Proc. of European Symposium of Algorithms (ESA)*, volume 1643 of *Lecture Notes in Computer Science*, pages 184–193. Springer, 1999.

[JW05] Thomas Jansen and Ingo Wegener. Real royal road functions–where crossover provably is essential. *Discrete Applied Mathematics*, 149(1-3):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] G. Karafotias, M. Hoogendoorn, and A.E. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19:167–187, 2015.

[KM12] Timo Kötzing and Hendrik Molter. ACO beats EA on a dynamic pseudo-boolean function. In *Proc. of Parallel Problem Solving from Nature (PPSN)*, volume 7491 of *Lecture Notes in Computer Science*, pages 113–122. Springer, 2012.

References (5/6)

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

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

[LW14] Andrei Lissovoi and Carsten Witt. MMAS vs. population-based EA on a family of dynamic fitness functions. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 1399–1406. ACM, 2014.

[LW15] Andrei Lissovoi and Carsten Witt. Runtime analysis of ant colony optimization on dynamic shortest path problems. *Theoretical Computer Science*, 561:73–85, 2015.

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

[NOW09] Frank Neumann, Pietro Simone Oliveto, and Carsten Witt. Theoretical analysis of fitness-proportional selection: landscapes and efficiency. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, 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.

[OZ15] Pietro Simone Oliveto and Christine Zarges. Analysis of diversity mechanisms for optimisation in dynamic environments with low frequencies of change. *Theoretical Computer Science*, 561:37–56, 2015.

References (6/6)

[PPHST15] Tiago Paixao, Jorge Pérez Heredia, Dirk Sudholt, and Barbora Trubenova. First steps towards a runtime comparison of natural and artificial evolution. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 1455–1462. ACM, 2015.

[Rud97] Günter Rudolph. *Convergence Properties of Evolutionary Algorithms*. Kovac, 1997.

[Sto06] Tobias Storch. How randomized search heuristics find maximum cliques in planar graphs. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, 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 *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 2005.

[SW04] Tobias Storch and Ingo Wegener. Real royal road functions for constant population size. *Theoretical Computer Science*, 320(1):123–134, 2004.

[Wit05] Carsten Witt. Worst-case and average-case approximations by simple randomized search heuristics. In *Proc. of Symposium on Theoretical Aspects of Computer Science (STACS)*, 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. Revised analysis of the (1+1) EA for the minimum spanning tree problem. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 509–516. ACM, 2014.