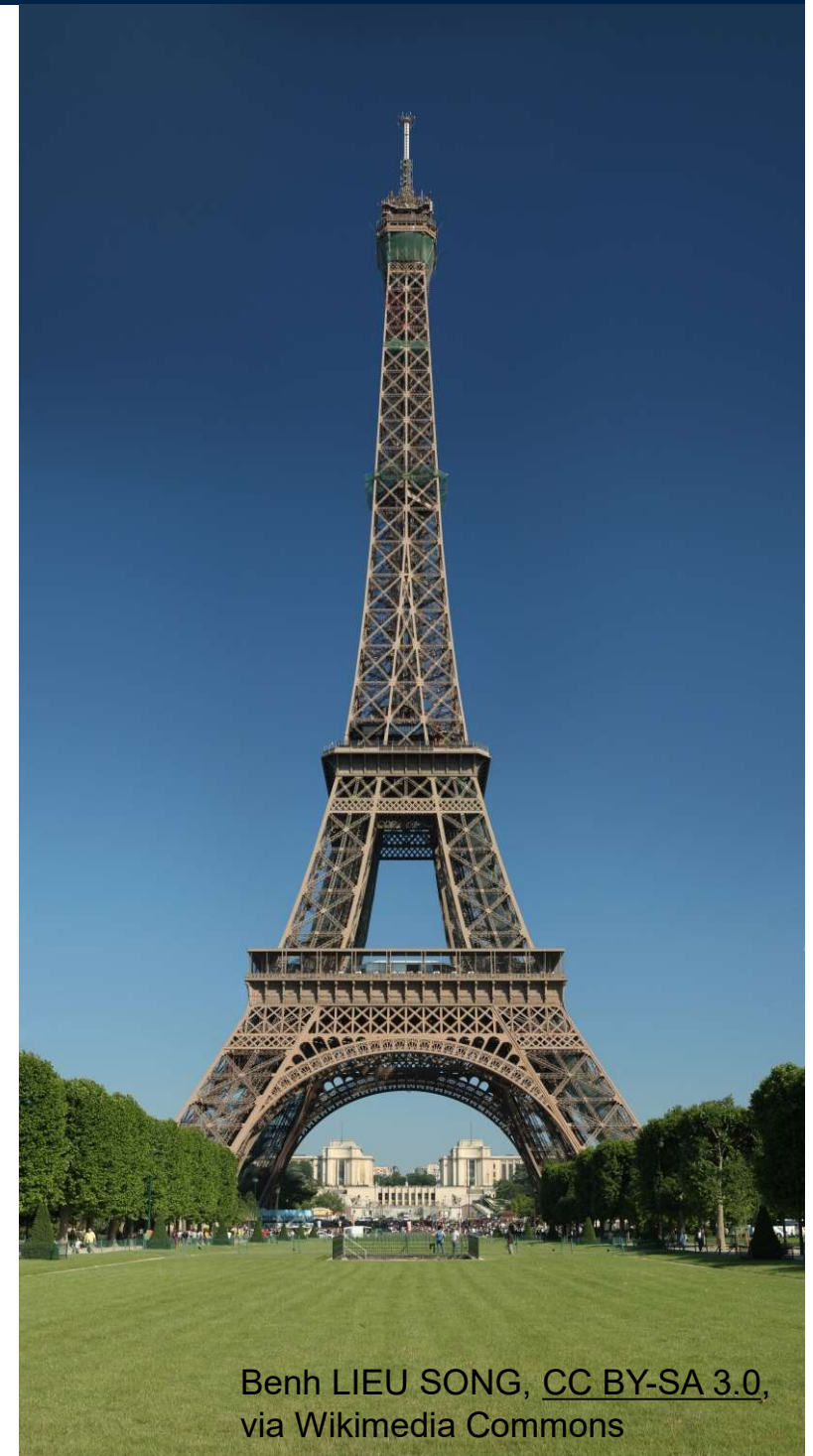




# A Gentle Introduction to Theory (for Non- Theoreticians)

*Benjamin Doerr*  
École Polytechnique



Benh LIEU SONG, [CC BY-SA 3.0](#),  
via Wikimedia Commons

# Instructor: Benjamin Doerr

- **Benjamin Doerr** is a full professor at the French École Polytechnique.
- He received his diploma (1998), PhD (2000) and habilitation (2005) in mathematics from the university of Kiel (Germany). His research area is the theory of both problem-specific algorithms and randomized search heuristics like evolutionary algorithms. Major contributions to the latter include runtime analyses for existing evolutionary algorithms, the determination of optimal parameter values, and the theory-guided design of novel operators, on-the-fly parameter choices, and whole new evolutionary algorithms.
- Together with Frank Neumann and Ingo Wegener, Benjamin Doerr founded the theory track at GECCO and served as its co-chair 2007-2009, 2014, and 2023-2024. He is a member of the editorial boards of several journals, among them *Artificial Intelligence*, *Evolutionary Computation*, *Natural Computing*, and *Transactions on Evolutionary Computation*. Together with Frank Neumann, he edited the book *Theory of Evolutionary Computation – Recent Developments in Discrete Optimization* (Springer 2020).

# This Tutorial: A *Real* Introduction to Theory

- GECCO, CEC, PPSN always had a good number of theory tutorials.
- They did a great job in educating the theory community.
- However, not much was offered for those attendees which
  - have little experience with theory,
  - but want to understand what the theory people are doing (and why).
- This is the target audience of this tutorial. We try to **answer those questions which come before the classic theory tutorials.**

# Questions Answered in This Tutorial

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

# Focus: EAs for Discrete Search Spaces

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

# The Most Important Point Before We Start

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

# Outline of the Tutorial

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

# Part I:

## What is *Theory of EC*

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



# What Do We Mean With *Theory*?

- Definition (for this tutorial):  
By theory, we mean **results proven with mathematical rigor**.
- Mathematical rigor:
  - make precise the evolutionary algorithm (EA) you regard
  - make precise the problem you try to solve
  - formulate a precise statement how this EA solves this problem
  - **prove this statement**
- **Example:**  
Theorem: The (1+1) EA generates the optimum of the OneMax function 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**: Use mathematical notation to describe, measure, or 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**: Use mathematical notation to describe, measure, or quantify observations.
  - Example: fitness-distance correlation, schema theory, ...
- **“Theories”**: Unproven claims that (mis-)guide our thinking.
  - Example: building block hypothesis

# Why Do Theory? Because of the **Results!**

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

# Why Do Theory? Because of the **Approach!**

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

# Limitations of Theoretical Research

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

# Part II:

## A Guided Walk Through a Famous Theory Result

We use a simple but famous theory result

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

# A Famous Result

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

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

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

## **Reference:**

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

- Famous paper (500+ citations, maybe the most-cited pure EA theory paper)
- Famous problem (20+ papers working on exactly this problem, many highly useful methods were developed in trying to solve this problem)



# Reading This Result

should be made precise in the paper to avoid any ambiguity

A mathematically proven result

(1+1) evolutionary algorithm to maximize  $f: \{0, 1\}^n \rightarrow \mathbb{R}$ :

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

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

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

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

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

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

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

# Why is This a Good Result?

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

→ more on these 3 items  
on the next slides

# Non-Trivial:

# Non-Trivial: Hard to Prove & Hard to Explain

## Why it Should be True

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

# Surprising: Same Runtime For Very Different Fitness Landscapes

- **Example 1: OneMax**, the function counting the number of 1s in a string,  $OM: \{0,1\}^n \rightarrow \mathbb{R}, (x_1, \dots, x_n) \mapsto \sum_{i=1}^n x_i$ :
  - 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  $BV: \{0,1\}^n \rightarrow \mathbb{R}, (x_1, \dots, x_n) \mapsto \sum_{i=1}^n 2^{n-i} x_i$ :
  - unique global maximum at  $(1, \dots, 1)$
  - very low fitness-distance correlation:
    - $BV(10 \dots 0) = 2^{n-1}$ , distance to optimum is  $n - 1$ ,
    - $BV(01 \dots 1) = 2^{n-1} - 1$ , distance to optimum is 1.

# Insight in Working Principles

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

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

**independent of  $f$ .**

# A Glimpse on a Modern Proof

- **Theorem [DJW12]:** For all problem sizes  $n$  and all linear functions  $f: \{0,1\}^n \rightarrow \mathbb{R}$  with  $f(x) = w_1x_1 + \dots + w_nx_n$  the (1+1) EA finds the optimum  $x^*$  of  $f$  in an expected number of at most  $4en \ln(2en)$  iterations.
- 1<sup>st</sup> proof idea: Without loss, we can assume that  $w_1 \geq w_2 \geq \dots \geq w_n > 0$ .
- 2<sup>nd</sup> proof idea: Regard an artificial fitness measure!
  - Define  $\tilde{f}(x) = \sum_{i=1}^n \left(2 - \frac{i-1}{n}\right) x_i$  “artificial weights” from 2 down to  $1 + \frac{1}{n}$
  - Key lemma: Consider the (1+1) EA optimizing the original  $f$ . Assume that some iteration starts with the search point  $x$  and ends with the random search point  $x'$ . Then
$$E[\tilde{f}(x^*) - \tilde{f}(x')] \leq \left(1 - \frac{1}{4en}\right) (\tilde{f}(x^*) - \tilde{f}(x)).$$

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

# Multiplicative Drift Theorem


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

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

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

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

- On the previous slide, this theorem was used with
  - $\delta = 1/4en$ ,
  - $X_t = \tilde{f}(x^*) - \tilde{f}(x^{(t)})$ ,
  - and the estimate  $X_0 \leq 2n$ .
- **Bibliographical notes:** Artificial fitness functions very similar to this  $\tilde{f}$  were already used in Droste, Jansen, and Wegener [DJW02] (conference version [DJW98b]). Drift analysis (“translating progress into runtime”) was introduced to the field by He and Yao [HY01] to give a simpler proof of the [DJW02] result. A different approach was given by Jägersküpfer [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.



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



# Limitations of the Linear Functions Result

- An **unrealistically simple EA**: the (1+1) EA.
- Linear functions are “trivial” **artificial test function**.
- **Not a precise result**, but
  - only  $O(n \log n)$  in [DJW02]
  - or a most likely significantly too large constant in the [DJW12] result.
  
- Two 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  $(1+\lambda)$  EA optimizes any linear function in expected time (= number of fitness evaluations)

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

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

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

→ 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<sup>+</sup>09],
  - minimum spanning trees [NW07, DJ10, Wit14],
  - knapsack [WPN16, NS18, NS19, XNNS21 and many more],
  - 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<sup>+</sup>09, OHY09], maximum cliques [Sto06], graph coloring [SZ10, BS19].
- We have some results on dynamic and noisy optimization (→ part V).

# Limitation 3: Only Asymptotic Results

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

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

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

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

# Summary “Guided Tour”

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

# Part III:

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

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

# Contribution 1: Debunk Misconceptions

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

# Misconception 1: Functions Without Local Optima are Easy to Optimize

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



# Misconception 2: Monotonic Functions are Easy to Optimize for EAs

- A function  $f: \{0,1\}^n \rightarrow \mathbb{R}$  is *monotonically strictly increasing (monotonic)* if the fitness increases whenever you flip a 0-bit into 1.
  - strong version of “no local optima”: *each* neighbor with additional ones is better
- Misconception: Such functions are easy to optimize for standard EAs...
  - because already simple hill-climbers flipping single bits (e.g., randomized local search) do the job in time  $O(n \log n)$ .
- Truth: **There are (many) monotonic functions such that with high probability the (1+1) EA with mutation probability  $16/n$  needs exponential time to find the optimum [DJS<sup>+</sup>13].**
  - The  $16/n$  can be lowered to  $\approx 2.13/n$  [LS18].
  - Same result for many mutation-based algorithms [Len20].
  - For any  $c > 0$  there is a  $\mu = O(n)$  and a monotonic  $f$  such that the  $(\mu+1)$  EA with mutation rate  $c/n$  needs super-polynomial time to optimize  $f$  [LZ21].

# Summary Misconceptions

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

# Contribution 2: Help With Design Choices

- When designing an EA, you have to choose between a huge number of design alternatives: the **basic algorithm**, the **operators** and **representations**, the **parameters**, ....
- **Theory can guide you with deep and reliable analyses of scenarios similar to yours.**
  - The question “what is a similar scenario” remains, but you have the same problem when looking for advice from experimental research.
- **Examples:**
  - use of fitness-proportionate selection
  - representations in graph problems
  - use of crossover: [JW02,SW04,FW04,FW05,JW05,Sud05,WJ07,RWP08,DT09,NT10,LY11,KST11,DJK+11,DHK12a,DJK+13,DFK+16,Sud17,DFK+18,CO18,CO20,OSW20,Sut21,FKR+22,LM24,DEJK24]
  - parameters: [Müh92,Bäc93,GKS99,JW00,Prü04,JJW05,Wit06,JS07,BDN10,Leh10,Leh11,LY12,Sud13,Wit13,RS14,DK15,GW17,DLMN17,ADFH18,ADY19,AD20,BBD21a,AD21,Doe21]



→ more on these 2  
on the next slides

# Design Choices:

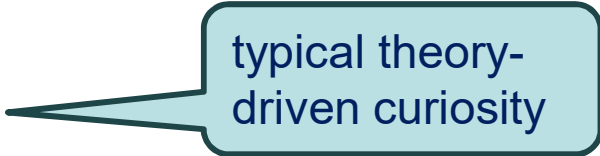
## Fitness-Proportionate Selection

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

### Algorithm (SGA)

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

# Design Choices: Representations

- Several theoretical works on shortest path problems [STW04, DHK07, BBD<sup>+</sup>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**? 
  - individuals are set of edges (forming reasonable paths)
  - mutation: add a random edge (and delete the one made obsolete)
- **Result:** All previous algorithms become faster by a factor of  $\approx \frac{|V|^2}{|E|}$ 
  - [JOZ13]: edge-based representation also preferable for vertex cover
- **Interpretation:** While there is no guarantee for success, it may be useful to think of an edge-based representation for graph-algorithmic problems

# Summary Design Choices

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

# Contribution 3: Invent New Operators and Algorithms

- Theory can also, both via the **deep understanding gained from proofs** and by “**theory-driven curiosity**” invent new operators and algorithms.
- Example: **What is the right way to do mutation [DLMN17]?**
- Outline (of the next 10+ slides):
  - What is “known” about mutation
  - A thorough analysis how simple EAs optimize the jump benchmark
  - Some unexpected conclusions [**best-paper award in the GECCO 2017 Genetic Algorithms track**]
- 2<sup>nd</sup> example [not shown]: Design of the  $(1 + (\lambda, \lambda))$  GA based on black-box complexity insight [DDE13, GP14, DDE15, DD15a, DD15b, Doe16, BD17, DD18, KAD19, ADK19, BB19, ABD20, AD20, BB20, FS22, ABD22, ADK22, ABD24]

# General Belief on Mutation

- **Note:** We only deal with *bit-string representations*, that is, the search space is  $\{0,1\}^n$  for some  $n$ .
  - [Similar results hold for other discrete search spaces, e.g., permutations [DGI24]]
- **General belief:** *The right way of doing mutation is standard bit mutation*, that is, flipping each bit independently with some probability  $p$  (“mut. rate”).
  - Global operator: from any parent you can generate any offspring (possibly with very small probability).
    - *Algorithms cannot get stuck forever in a local optimum.*
- **General recommendation:** *Use a small mutation rate like  $p = 1/n$ .* See, e.g., [Bäc96, BFM97, Och02].



# Informal Justifications for $p = 1/n$

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

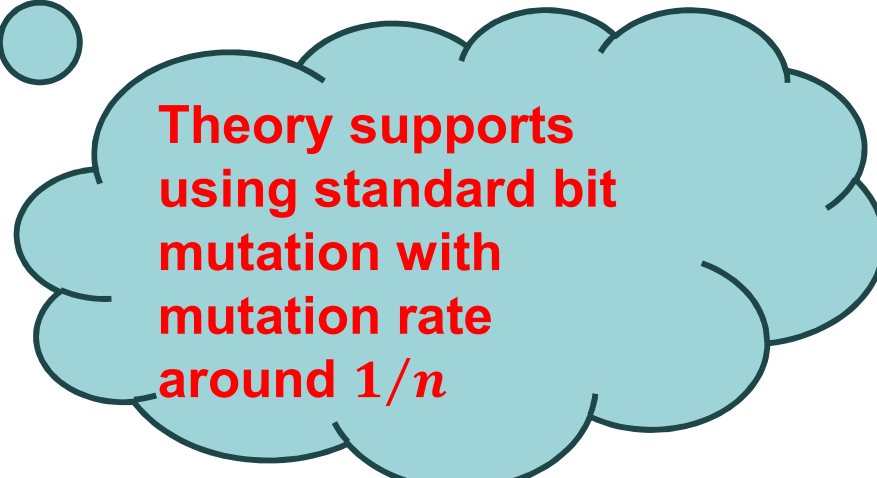
$O(c)$  = at most  $\gamma c$  for some constant  $\gamma$

$\Omega(c)$  = at least  $\delta c$  for some constant  $\delta > 0$

$\Theta(c)$  = both  $O(c)$  and  $\Omega(c)$

# Proven Results Supporting $p = 1/n$

- Optimal mutation rates for (1+1) EA:
  - $\approx \frac{1}{n}$  for OneMax [Müh92; Bäck93, GKS99]
  - $\approx \frac{1.59}{n}$  for LeadingOnes [BDN10]
  - $\approx \frac{1}{n}$  for all linear functions [Wit13]
  - monotonic functions [Jan07, DJSWZ13, LS18, LMS19]:
    - $p = \frac{c}{n}, 0 < c < 1$ , gives a  $\Theta(n \log n)$  expected runtime on all monotonic functions with unique optimum,
    - $p = \frac{c}{n}, c \in [1, 1 + \varepsilon]$  for some  $\varepsilon > 0$  gives  $O(n \log^2 n)$ ,
    - $p \geq \frac{2.13\dots}{n}$  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].



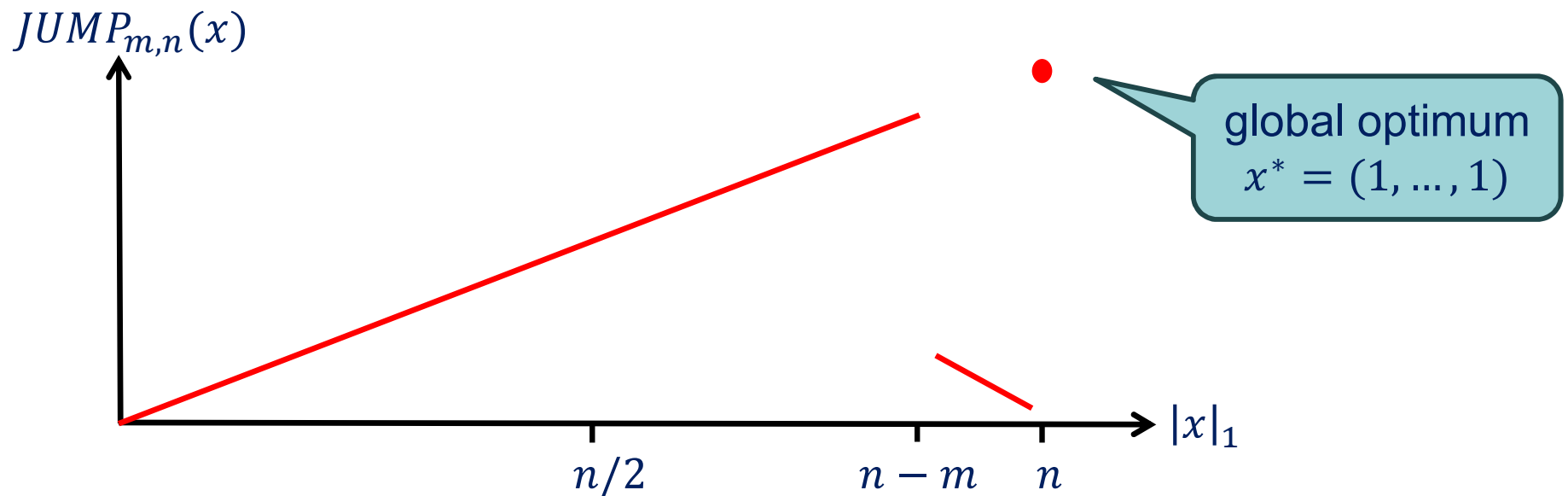
**Theory supports using standard bit mutation with mutation rate around  $1/n$**

# Really?

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

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

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



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

# Runtime Analysis

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

- Theorem: For all  $2 \leq m \leq n/2$  and  $p \leq 1/2$ ,

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

- Let  $T_{opt} = T_{opt}(m, n) := \inf\{T_p(m, n) \mid p \in [0, 1/2]\}$ .

- Theorem: If  $m \leq n/4$ , then  $T_{opt} = T_{m/n} (1 \pm o(1)) \approx \left(\frac{e}{m}\right)^m T_{1/n}$  and  $p = m/n$  is essentially the only optimal mutation rate.

- The right mutation rate is much higher than the usual  $1/n$  and it gives a huge speed-up!**

# Missing the Optimal Mutation Rate

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

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

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

# From This Analysis to a New Mutation Operator

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

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

# Heavy-tailed Mutation: Results

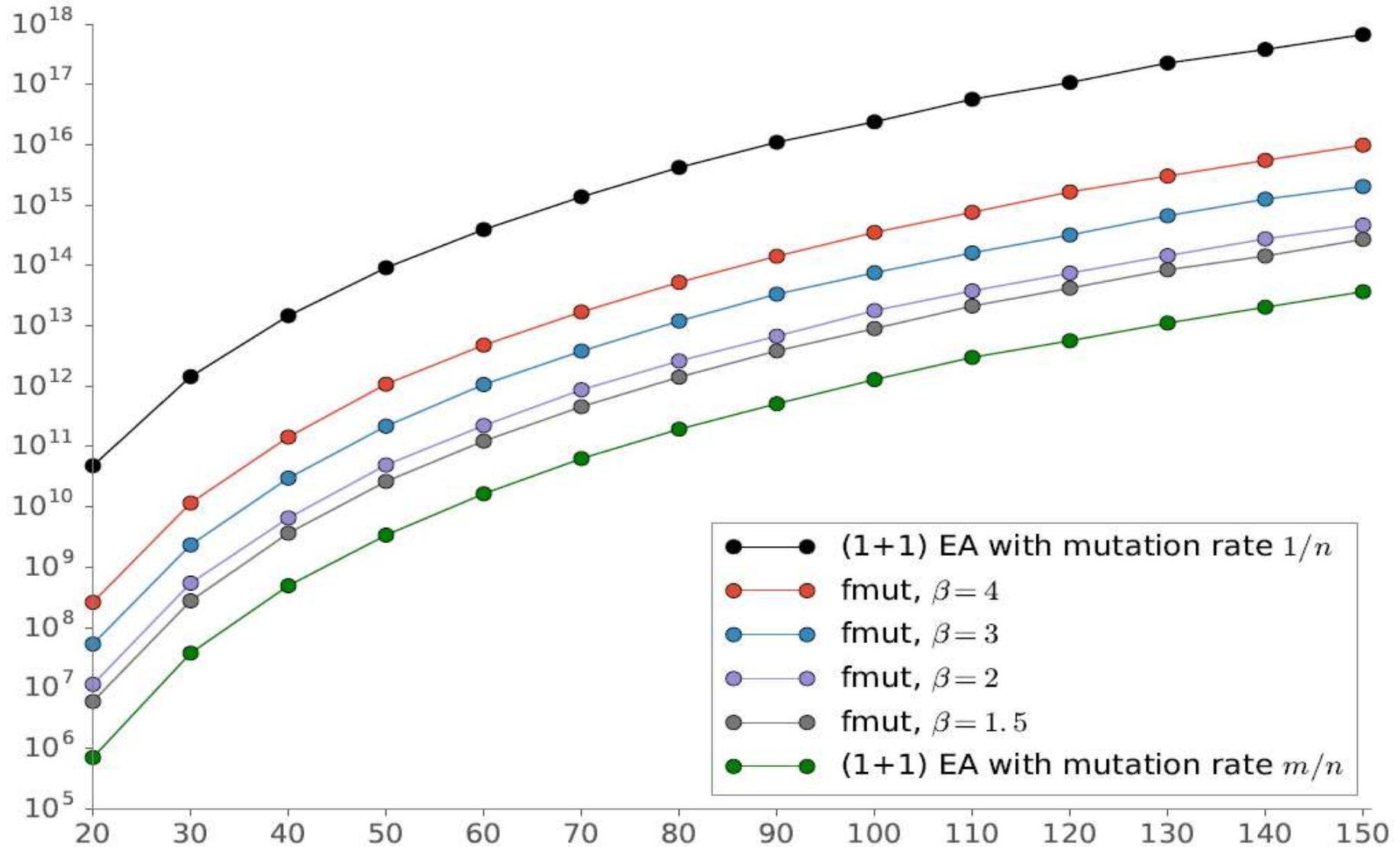
- Theorem: The (1+1) EA with heavy-tailed mutation ( $\beta > 1$ ) has an expected optimization time on  $JUMP_{m,n}$  of

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

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



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



Runtime of the (1+1) EA on  $JUMP_{8,n}$  (average over 1000 runs). To allow this number of experiments, the runs were 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  $(1 + o(1)) e \zeta(\beta) \left(\frac{e}{m}\right)^m m^{\beta-0.5} T_{n,\varepsilon}$ .
- 2<sup>nd</sup> example: **Vertex cover** in bipartite graphs (details omitted).

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

# 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
  - $O(m^2 \log(nw_{\max}))$  runtime on MinimumSpanningTree [NW07]
  - and many others...
- The **largest expected runtime** that can occur on an  $f: \{0,1\}^n \rightarrow \mathbb{R}$  is ...
  - $\Theta(n^n)$  for the classic (1+1) EA: Trap function [DJW02], minimum makespan scheduling [Wit05]
  - $O(n^\beta 2^n)$  for the heavy-tailed (1+1) EA

# Working Principle of Heavy-Tailed Mutation

- Reduce the probability of a 1-bit flip slightly (say from 37% to 32%)
- Distribute this free probability mass in a **power-law** fashion on all other  $k$ -bit flips
  - increases the prob. for a  $k$ -bit flip from roughly  $\frac{1}{e \cdot k!}$  to roughly  $k^{-\beta}$
  - reduces the waiting time for a  $k$ -bit flip from  $e \cdot k!$  to  $k^\beta$
- This redistribution of probability mass is a good deal, because we usually spend much more time on finding a good multi-bit flip
  - $JUMP_{m,n}$ : spend  $\Theta(n \log n)$  time on all 1-bit flips, but  $\binom{n}{m}$  time to find the one necessary  $m$ -bit flip
- These elementary observations are a good reason to believe that heavy-tailed mutation is advantageous for a wide range of multi-modal problems.
  - Other theory works: [FQW18, FGQW18, WQT18, ABD20a, ABD20b, DZ21, QGWF21, BBD23, DQ23a, ZD24, ABD24]

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

# Heavy-Tailed → “Fast”

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

# Summary: Fast Mutation on $\{0, 1\}^n$

## – A Theory-Guided Invention

- By rigorously analyzing the performance of a simple mutation-based EA on the non-unimodal JUMP fitness landscape, we observe that
  - higher mutation rates are useful to leave local optima
  - standard bit mutation with a fixed rate is sub-optimal on most problems
- Solution: Use standard bit mutation, but with a random mutation rate sampled from a power-law distribution
  - $m^{\Theta(m)}$  factor speed-up for  $JUMP_{m,n}$  and many other problems
- Does this work in practice? First results are promising 😊
  - Neumann, Xie, Neumann [NXN22]: Knapsack with stochastic profits
  - D, Krejca, Vu [DKV24] (GECCO'24: BP nominee ECOM): Target set selection

# Summary Part 3

Theory has contributed to the understanding and use of EAs by

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

# Part IV:

## How Theory Can Help YOU: Theory-Style Thinking



# How Theory Can Help YOU

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

# An Example of *Theory-Style Thinking*

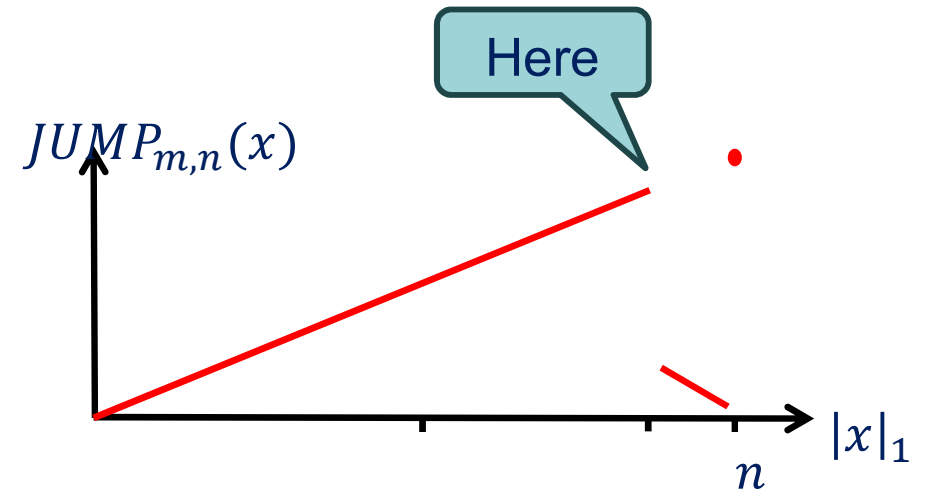
- Problem: You run your favorite evolutionary algorithm on your favorite problem and you feel that it **takes too long to leave local optima**.
- You try without success all your tricks:
  - parameter tuning
  - landscape analysis
  - asking colleagues that are true experts in experimental work
  - etc.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

# 1<sup>st</sup> Step: Take a Really Simple Example Situation

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

# 2<sup>nd</sup> Step: Analyze Your Example Precisely and in Full Generality

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



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

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

# 3<sup>rd</sup> Step: Generate Useful Data

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

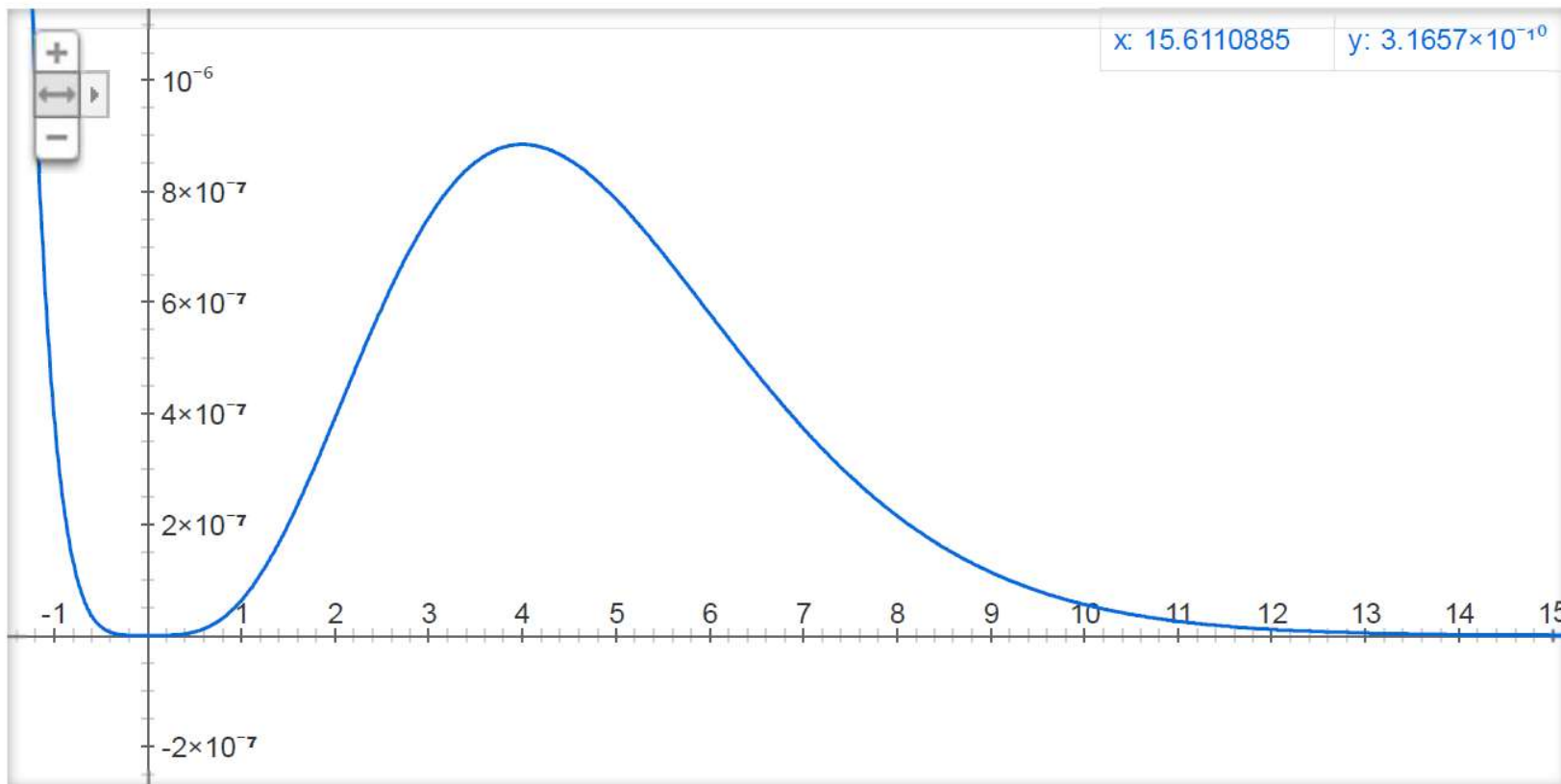
$$P(\alpha) = \left(\frac{\alpha}{50}\right)^4 \left(1 - \frac{\alpha}{50}\right)^{46}, \alpha \in [0.05, 20]$$

- [type “ $y = (x/50)^4*(1-x/50)^{46}$ ” into Google]

# 3<sup>rd</sup> Step: Generate Useful Data

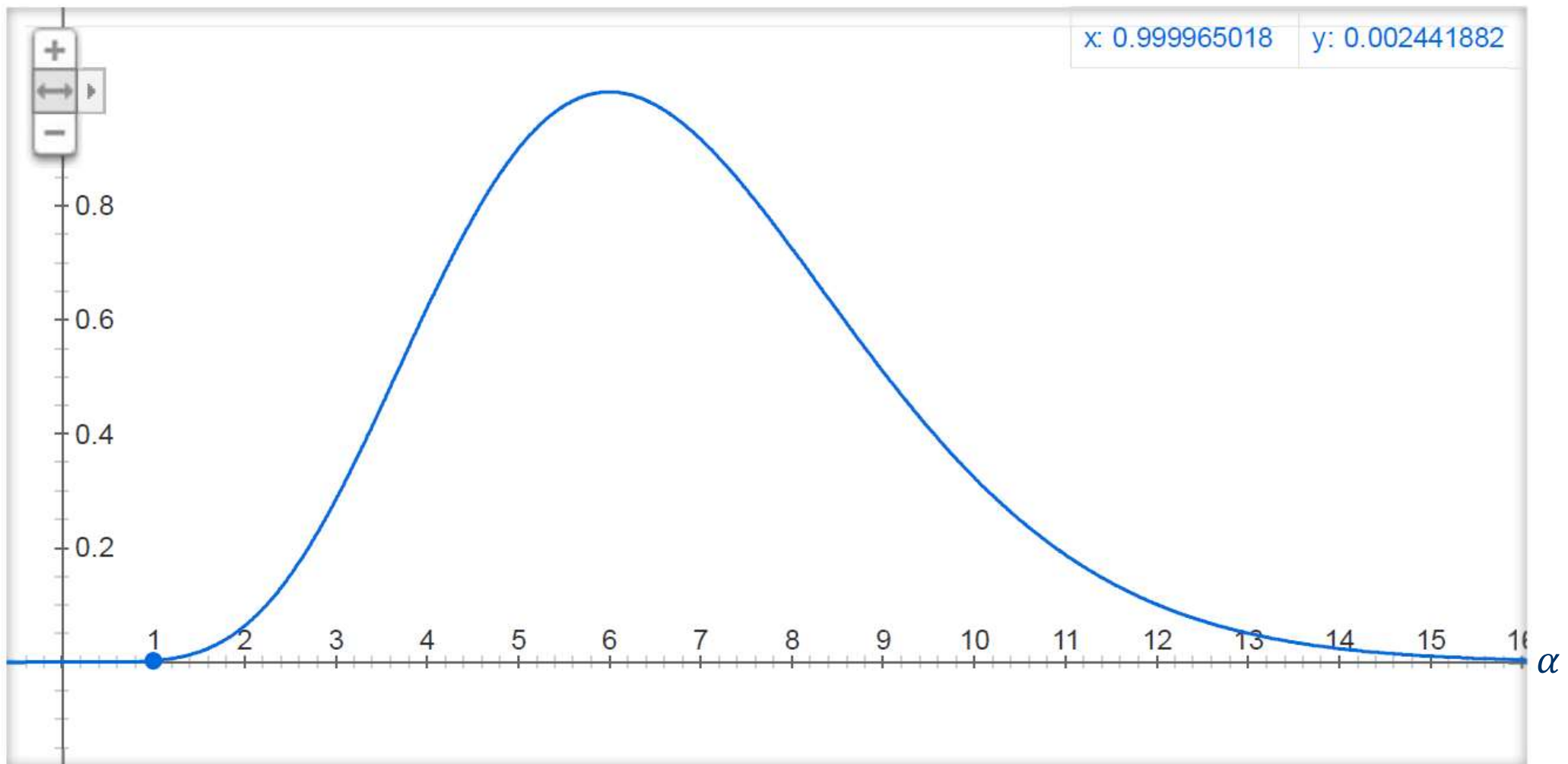
- We plot the function  $p \mapsto P(p) = p^m(1-p)^{n-m}$ 
  - a moderate problems size  $n = 50$ , a small jump size  $m = 4$ ;
  - for the mutation rate  $p$ , we recall that the standard choice is  $1/n$ . So let's use the scaling  $p = \alpha/n$  and take  $\alpha \in [0.05, 20]$ :

Graph for  $(x/50)^4(1-x/50)^{46}$



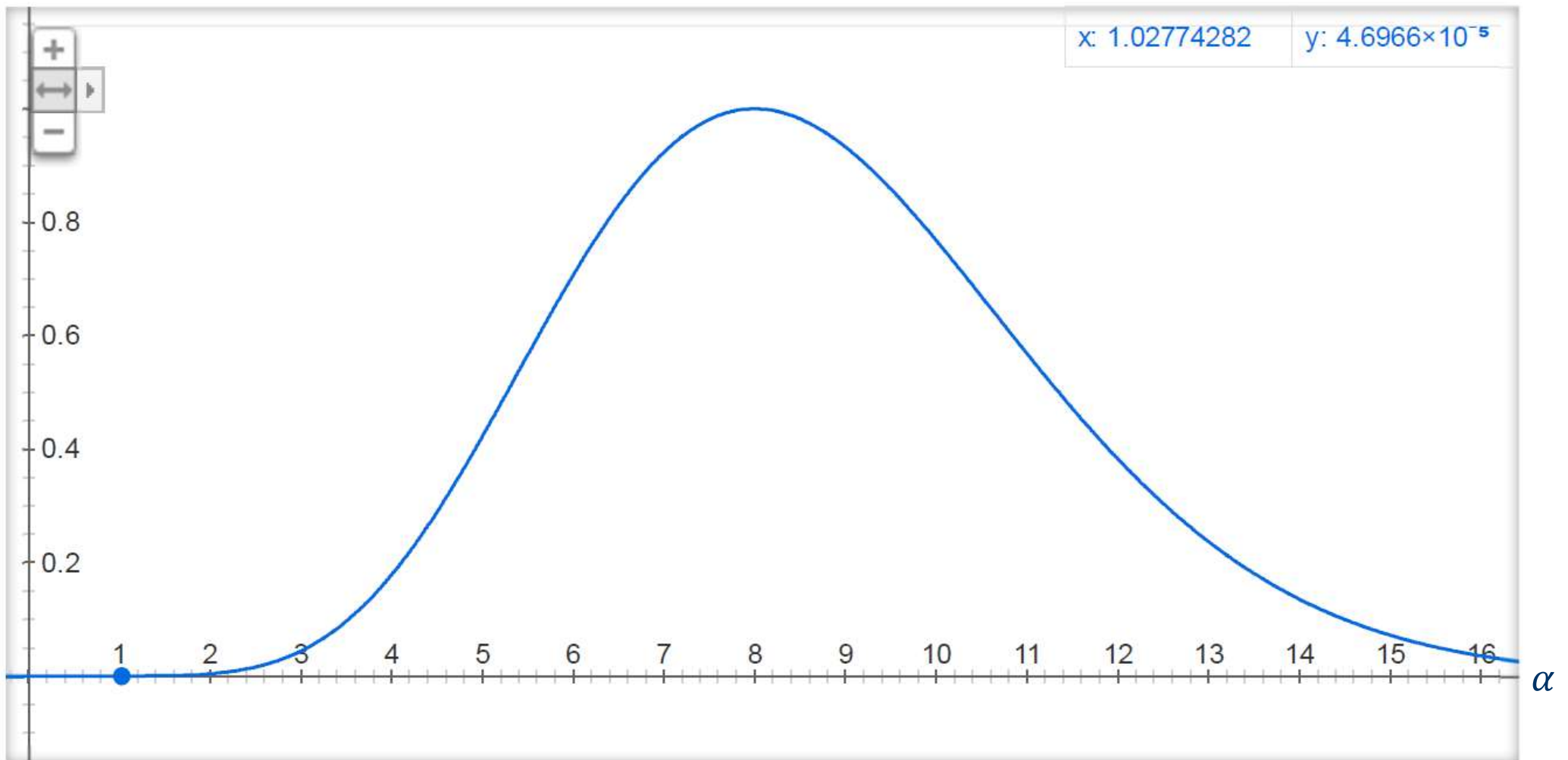
$$P(\alpha) = \left(\frac{\alpha}{n}\right)^m \left(1 - \frac{\alpha}{n}\right)^{n-m}, n = 50, m = 6$$

Graph for  $(x/50)^6 \cdot (1-x/50)^{44} / ((6/50)^6 \cdot (1-6/50)^{44})$



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

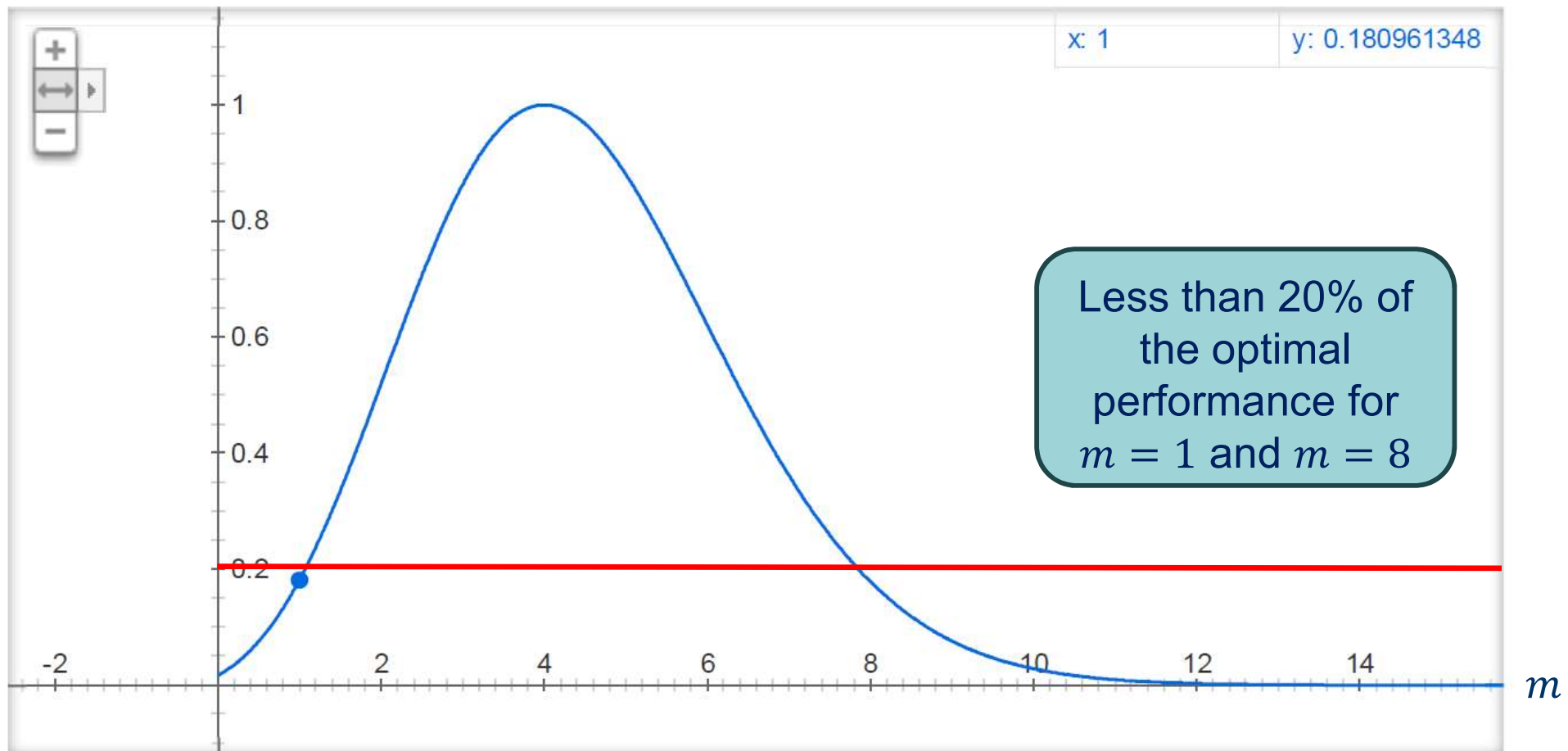
Graph for  $(x/50)^8 \cdot (1-x/50)^{42} / ((8/50)^8 \cdot (1-8/50)^{42})$





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

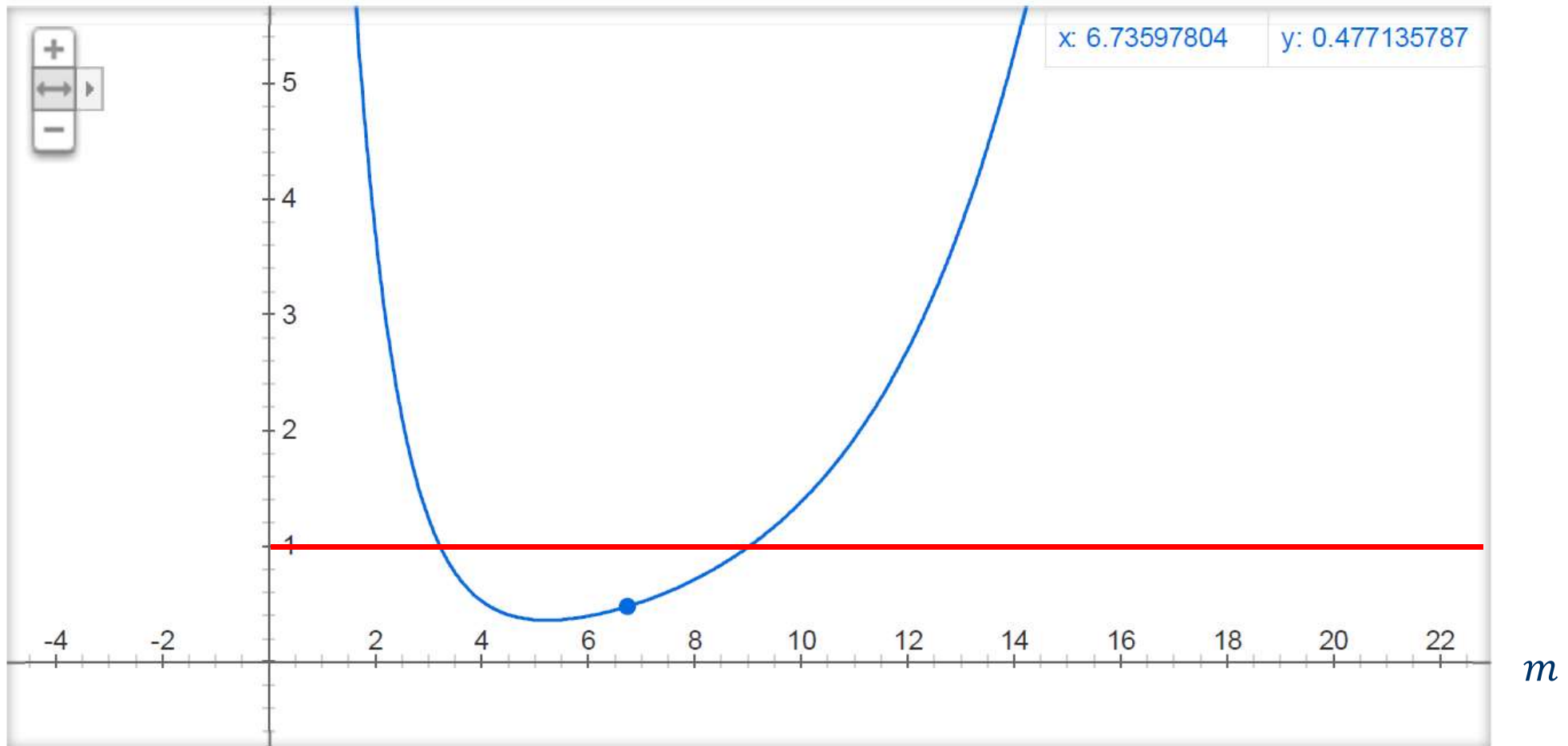
Graph for  $(4/50)^x \cdot (1-4/50)^{(50-x)} / ((x/50)^x \cdot (1-x/50)^{(50-x)})$



# 4<sup>th</sup> Step: Interpret the Data and Find a Solution

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

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



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

# Summary: Theory-Style Thinking

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

# Part V:

## Current Topics of Interest in the Theory of EC

- Populations
- Estimation-of-distribution algorithms (EDAs)
- Dynamic and noisy optimization
- Dynamic/adaptive parameter choices
- Fine-grained runtime analysis: fixed budget/target, parameterized complexity
- No slides:
  - Co-evolution: Per Kristian Lehre is the expert
  - **Evolutionary Multi-objective Optimization (EMO)**: see the Gecco EMO tutorial by Joshua Knowles and Weijie Zheng

# Hot Topic 1: Populations

- While most EAs in practice use non-trivial populations, EA theory has not been very successful in understanding why this is good (but some interesting results exist).
- Elitist mutation-based algorithms:
  - Larger **offspring population size** [JJW05, DK15, GW17, GW18]:
    - Allow parallel implementations (faster).
    - Usually no speed-up w.r.t. the total number of fitness evaluations.
    - Research question: Up to which pop. size you have a linear speed-up, that is, the total runtime does not increase?
  - Larger **parent population size**: Rather slows down things, but by surprisingly little [Wit06, ADFH18, AD20].
  - **Both can provably give robustness against noise and dynamic changes of the problem** [JS05, GK16, LW16, DJL17, LM24, ADI24].

# Populations (2)

- Non-elitist mutation-based algorithms: **Need sufficiently large populations** [JS07, NOW09, Leh10, Leh11, RS14, DL16a, CDEL18, DK19, Doe20]
  - Small offspring pop: **you lose good solutions too quickly** and cannot really optimize (exponential runtimes).
  - Large offspring pop: you usually generate a copy of a good parent and thus **imitate an elitist algorithm**.
  - Inside the phase transition: strange things happen [ADY19].
  - **Problem: Not too much argument for non-elitism in theory so far!**  
*Example: for no choice of the population sizes, the  $(\mu, \lambda)$  EA shows an interesting speed-up over the  $(\mu + \lambda)$  EA on jump functions [Doe22]*
- Large parent population plus diversity mechanism: The diversity mechanism can force the population to spread out, this can aid leaving local optima [FHN07, Sto08, FHN09, DFK<sup>+</sup>16, DFK<sup>+</sup>18, CS18, OSZ19].

# Populations (3)

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



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

- Example: **compact Genetic Algorithm (cGA)** of Harik, Lobo, and Goldberg [HLG99] with hypothetical pop. size  $K \in 2\mathbb{N}$  to maximize  $f: \{0,1\}^n \rightarrow \mathbb{R}$

- 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  $\tau$  in the cGA).
  - much richer representation of knowledge

# What Can EDAs Do That EAs Can't?

- **Robustness to noise:**
  - The cGA can cope well with normally distributed additive posterior noise [FKKS17]
  - The UMDA can cope well with 1-bit prior noise [LN19b]
  - (similar result for ACO found earlier [DHK12b, FK13, ST12])
- **Leaving local optima:** EDAs can optimize multimodal functions faster than many classic EAs [HS18, DK20c, Doe21, WZD21, BBD21b, Wit23]
  - different finding: on CLIFF, the cGA is (most likely) slower than the best EAs [NSW22] (but still faster than many standard EAs)
- **Model building = representing many good solutions at once:**
  - MIMIC can build a probabilistic model that allows to sample a huge number of distant good solutions (experimental) [DK20a]

# Difficulty: Genetic Drift

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

# Quantifying Genetic Drift

- Good news: From many previous works specifically targeting genetic drift [Sha02, Sha05, Sha06, FKK16] and many runtime analyses coping with genetic drift [Dro06, DLN19, LN17, LN18, HS18, SW19, Wit19, KW20, Doe21, LSW21], we now understand genetic drift well.

- **Theorem (stated for the cGA only):** Assume that the cGA optimizes a function with a neutral bit.
  - The first time the frequency of this bit is at the boundary, is  $O(K^2)$ .
  - The first time this frequencies leaves  $[0.25,0.75]$  is  $\Omega(K^2)$ .
  - The probability that this frequency leaves the interval  $[0.5 - \varepsilon, 0.5 + \varepsilon]$  in the first  $T$  iterations, is at most

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

- **Advice to the practitioner:** If you want to use a cGA for  $T$  iterations, set the parameter  $K$  somewhat larger than  $\sqrt{T}$ .

# Overcoming Genetic Drift

## Automated ways to set the parameters right:

- Parallel runs with diverse parameter values [Doe21]
- Smart restarts: Restart with a larger  $K$  when the theorem on the previous slide says that genetic drift could have occurred [DZ20a,ZD23a]

## EDA variants trying to avoid genetic drift outright:

- stable-cGA [FKK16]: cGA with an artificially modified frequency update.
  - $O(n \log n)$  runtime on LeadingOnes
  - exponential runtime on OneMax [DK20b].
- sig-cGA [DK20b, WZD21]: regards a longer history and changes frequencies only when there is sufficient evidence for it
  - $O(n \log n)$  runtime on OneMax, LeadingOnes, and DeceivingLeadingBlocks

# Summary: EDA Theory

- Significant progress in the last 8 years:
  - many runtime analyses, many strong methods
  - understanding genetic drift.
- Cool particular results:
  - EDAs work well in the presence of noise
  - EDAs can leave local optima quicker than most EAs
- Many open problems:
  - Tight runtime bounds for classic problems (OneMax)
  - Theory for multi-variate EDAs (no result yet)

# Hot Topic 3:

## Dynamic and Noisy Optimization

- **Dynamic optimization:** The problem to be solved changes over time
- **Noisy optimization:** Access to the problem data stochastically disturbed
  
- **General belief:** Due to their randomized and problem-unspecific 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 instance  $OM_z$  is replaced by  $OM_y$ , where  $y$  is a random neighbor of  $z$  (=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 problem in  $O(n^{ce+1+o(1)})$  iterations (in expectation).
- Bit-wise dynamics:  $y$  is obtained from  $z$  by flipping each bit independently with probability  $p' = c \frac{\ln n}{n^2}$  (expected distance  $H(y, z)$  same as above).
  - Droste [Dro03]: Runtime on OneMax is  $O(n^{4ce+1} \ln n)$ .
  - Kötzing, Lissovoi, Witt [KLW15]: Improved to  $22 n^{1.76ce+2} \ln n$ .
  - Dang-Nhu et al. [DNDD<sup>+</sup>18]: Improved to  $(3.77 + o(1))n^{1.39ce+2}$ .

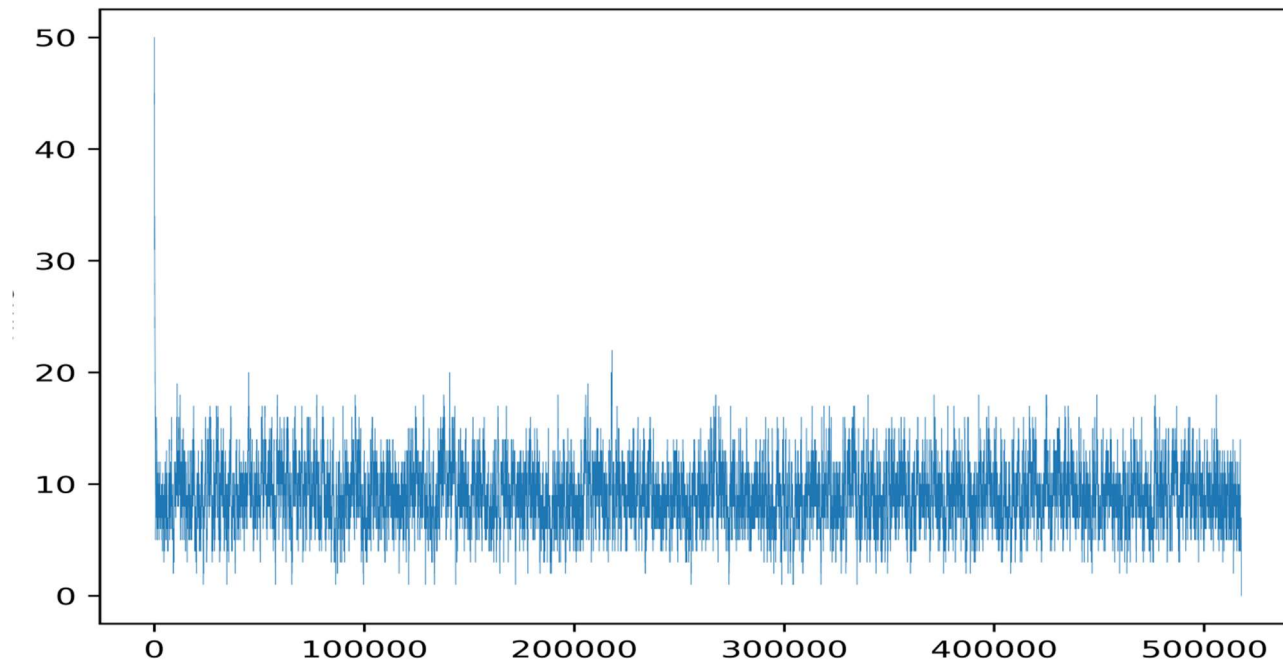


# Interpretation of These Results

- EAs are surprisingly robust to dynamically changing problem instances!
- Example: 1-bit dynamics with  $p = c \frac{\ln n}{n}$ .
  - In average, every  $\frac{n}{c \ln n}$  iterations the optimum moves to a neighbor.
  - → 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<sup>+</sup>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



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

A plot of a typical run (fitness distance over time) for  $n = 100$  and 1-bit dynamic with  $p = \frac{\ln n}{n}$ .

# Noisy Optimization

- Much more research on noisy optimization (started again by Droste [Dro04]).
- Typical setting: **Noisy access to the fitness function**, that is, some fitness evaluations return a wrong value.
  - **Prior noise models:** Instead of  $f(x)$ , the EA learns  $f(x')$  for some stochastically disturbed version  $x'$  of  $x$ .
    - *1-bit noise:* With probability  $p$ , you learn the fitness of a random neighbor  $x'$  of  $x$ .
    - *Bit-wise noise:* Obtain  $x'$  from  $x$  by flipping each bit independently with probability  $q$  and return  $f(x')$ .
  - **Posterior noise:** Return a disturbed version of  $f(x)$ , e.g.  $f(x) + D$  where  $D$  follows a normal distribution.

# Noisy Optimization: Results for the (1+1) EA

- **OneMax** (Droste [Dro04], Gießen, Kötzing [GK16], Dang-Nhu et al. [DNDD<sup>+</sup>18]):
  - $O(n \log n)$  when  $p \leq \frac{\beta \ln \ln}{n}$  (1-bit noise),  $q \leq \frac{\gamma \ln l}{n^2}$  (bit-wise noise)
  - polynomial time when  $p = O\left(\frac{\log n}{n}\right)$ ,  $q = O\left(\frac{\log n}{n^2}\right)$
  - super-polynomial time for stronger noise.
- **LeadingOnes** ([GK16], Qian, Bian, Jiang, Tang [QBJT19], Sudholt [Sud21]):
  - $O(n^2)$  when  $p = O(n^{-2})$ ,  $q = O(n^{-3})$
  - polynomial when  $p = O\left(\frac{\log n}{n^2}\right)$ ,  $q = O\left(\frac{\log n}{n^3}\right)$
  - super-polynomial for stronger noise.
- Bottom line: 1-bit and bit-wise noise behave similar when  $p = nq$ .
  - EA is somewhat robust, but not very robust; worse for LeadingOnes.

# Larger Populations: Additional Robustness

- $(\mu+1)$  EA on OneMax [GK16]: 1-bit noise with strength  $p$ ,  $\mu \geq 12 \frac{\log 15n}{p}$   
→ *noise-less runtime* of  $O(\mu n \log n)$  fitness evaluations.
- $(1+\lambda)$  EA on OneMax:
  - [GK16]: 1-bit noise with strength  $p$ ,  $\lambda \geq 24 \frac{n \log n}{p}$   
→ runtime  $O(n^2 \lambda p^{-1})$  fitness evaluations.
  - Antipov, D., Ivanova [ADI24] For bit-wise noise with any rate  $q = O(n^{-1})$ , a population size of  $\Omega(\log n)$  suffices to obtain the *noise-less runtime* of  $O\left(n\lambda \frac{\log \log}{\log} + n \log n\right)$  f-evals.
- $(1+\lambda)$  EA on LeadingOnes [GK16, Sud21]: 1-bit noise with strength  $p$  or bit-wise noise with strength  $q \leq 1/n$ . For  $\lambda = \Omega(pn)$  resp.  $\lambda = \Omega(qn^2)$ ,  $\lambda \geq 3.42 \ln n$ , and  $\lambda = O(n)$ , we have the *noise-less runtime* of  $O(n^2)$ .
- → **Already moderate population sizes massively increase the robustness to noise!**

# Additional Robustness from Resampling

- Resampling: Evaluate a solution several times and estimate the true fitness from the obtained values (usually by averaging).
- Akimoto, Astete Morales, Teytaud [AAT15]: Additive posterior noise can be overcome with averaging over sufficiently many re-evaluations.
- Qian et al. [QBJT19]:
  - More precise quantitative results for posterior noise.
  - First results for prior noise, e.g., the (1+1) EA resampling each search point  $k = 10n^4$  times can optimize LeadingOnes under 1-bit noise with rate  $p = 0.5$  in expected time  $O(n^6)$ .
- D. and Sutton [DS19]): Use the median instead of the average!
  - Example: To optimize LeadingOnes in the presence of constant noise rates (below 1/2), averaging needs  $\Omega(n/\log n)$  resamples. With the median,  $O(\log n)$  suffice.

# Other Results on Robustness To Noise

- **Ant-colony optimizers** and **estimation-of-distribution algorithms** are relatively robust to noise: Sudholt and Thyssen [ST12], Doerr, Hota, Kötzing [DHK12b], Friedrich, Kötzing, Krejca, Sutton [FKKS17], Lehre, Nguyen [LN19b], Zheng, D. [ZD23a].
  - Reason: The cautious up-date of the probabilistic model reduces the negative impact of wrong decisions stemming from noise.
- Multi-objective EAs can be relatively robust to noise: Dinot, D., Hennebelle, Will [DDHW23], Dang, Opris, Salehi, Sudholt [DOSS23a].
  - Reason: The larger population together with the implicit diversity mechanism reduces the negative impact of noise.

# Summary Dynamic and Noisy Optimization

- Due to their randomized nature, EAs without any specific adjustments cope well with moderate levels of noise and with moderate changes of the problem instance.
  - Larger population sizes help.
  - ACO, EDAs are more robust.
- Resampling can give additional robustness (at the price of more function evaluations per search point).



# Hot Topic 4: Dynamic Parameter Choices

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

# Theory for Dynamic Parameter Choices: *Deterministic Schedules*

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

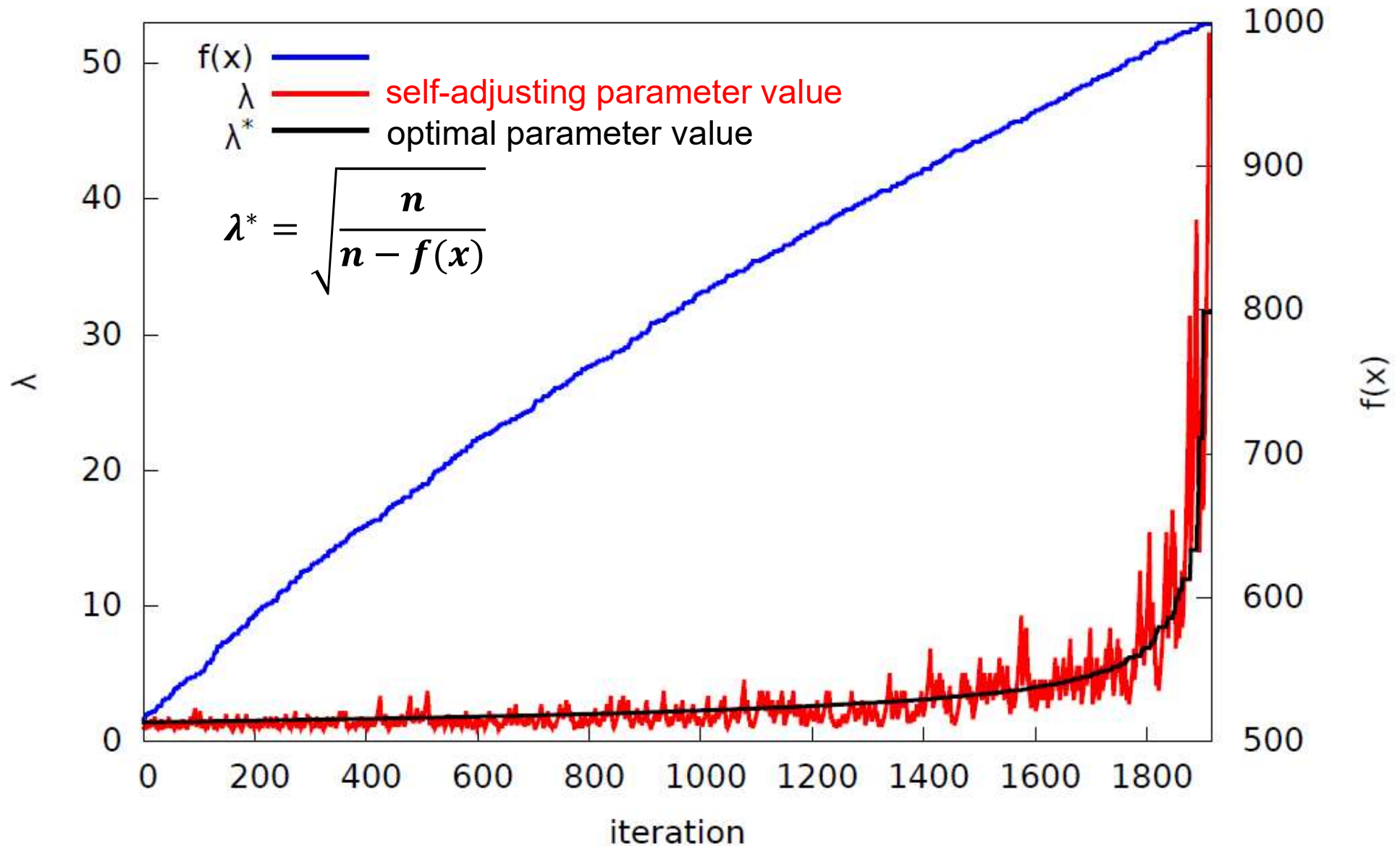
# Theory for Dynamic Parameter Choices: *Depending on the Fitness*

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

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

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

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



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

- Previous success-based dynamics:
  - Work well when the characteristics of the landscape changes slowly  
→ good parameter values “follow” the changes of the landscape.
  - Not much known for abrupt changes, e.g., jump functions.
- Very recent idea: “Stagnation detection” [RW21a, RW21b, RW22, DR23]
  - When for too long (details omitted) no improvement happened, increase the mutation strength (because there’ll be no improvement close by).
  - Can be added to all mutation-based algorithms.
  - Gives the best runtime for the (1+1) EA on jump functions.
- → **Very simple, very promising idea!**

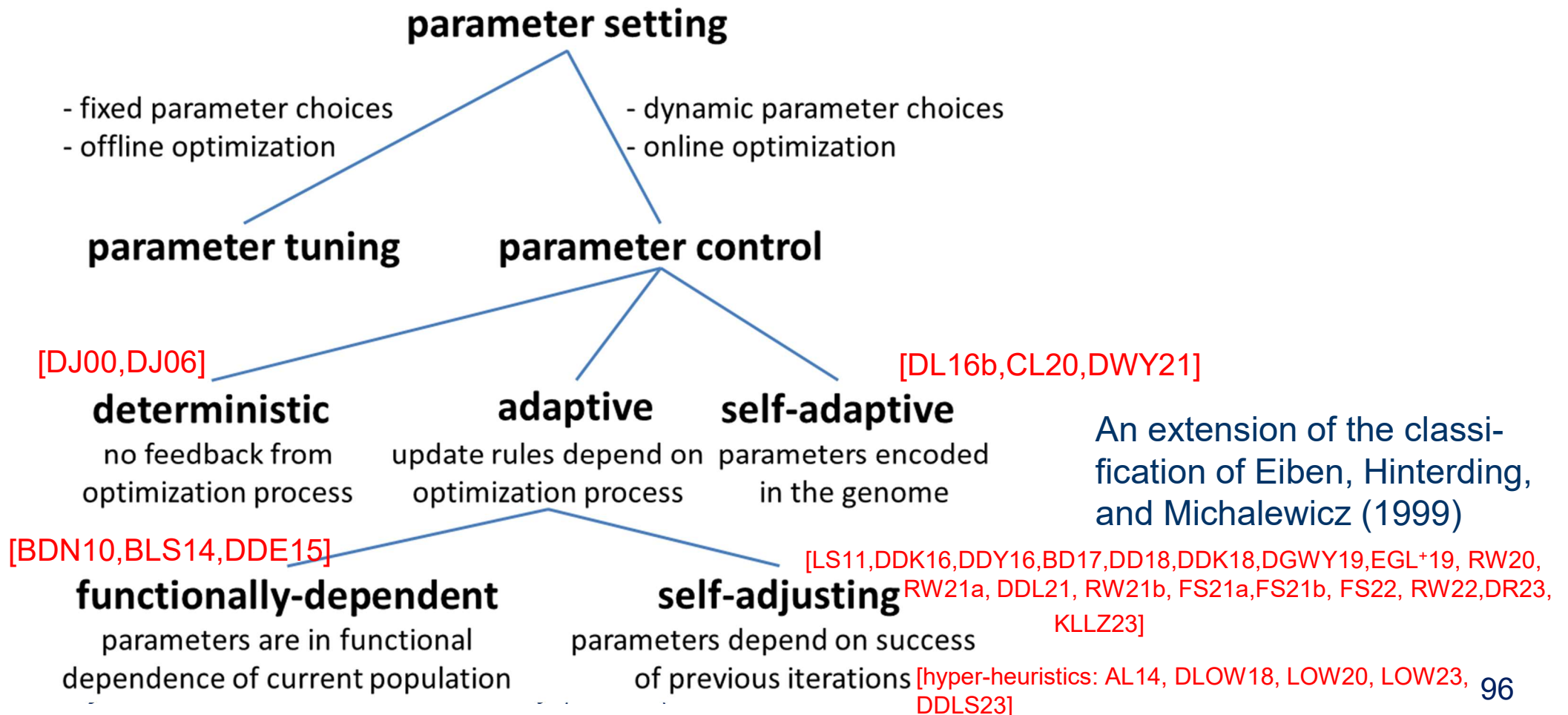
# Theory for Dynamic Parameter Choices: *Self-Adaptation*

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



# Summary Dynamic Parameter Choices

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





# Hot Topic 5: Fine-grained Runtime Analysis

- Classic runtime analysis: Analyze the time until the optimum is found.
- Recently: Runtime notions that give more or more relevant information.
- Fixed budget perspective: Analyze the (expected) solution quality obtainable in a given time budget [JZ12,DJWZ13,JZ14a,JZ14b,LS15,NNS17,DDY20,KW20]
  - Interesting from the application perspective, but difficult to analyze!
- Fixed target analysis, starting with good solutions: Classic runtime notion extended to arbitrary starting/ target solution qualities [BDDV20,ABD20a,DK21a].
  - Classic proofs can be re-used, but different algorithms become good.
- Parameterized complexity: Analyze the runtime relative to a parameter of the input instance [Sto06,Sto07,KLNO10,SN12,KN13,SNN14,FN15,CLNP16,Sut21]
  - Hot topic in classic algorithms since 1999

# Part VI:

# Conclusion

# Summary

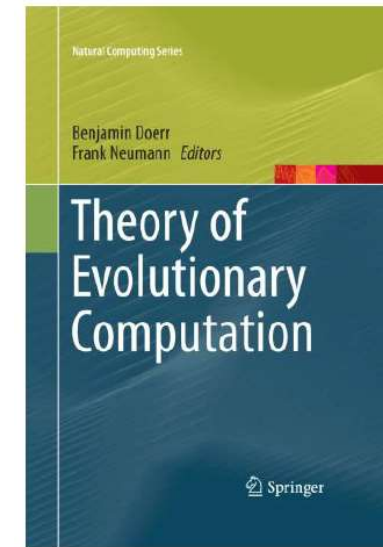
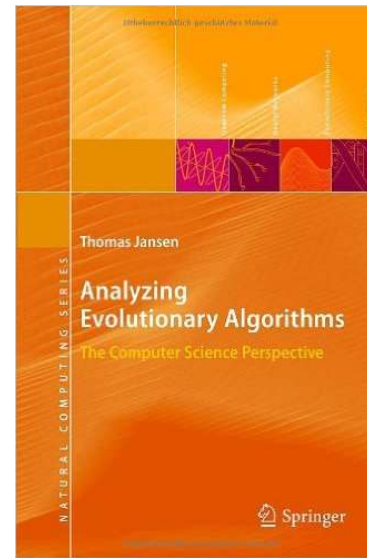
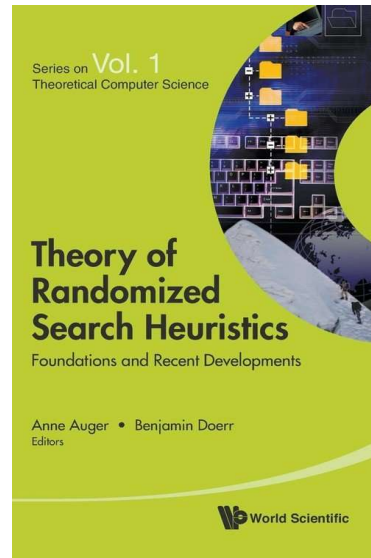
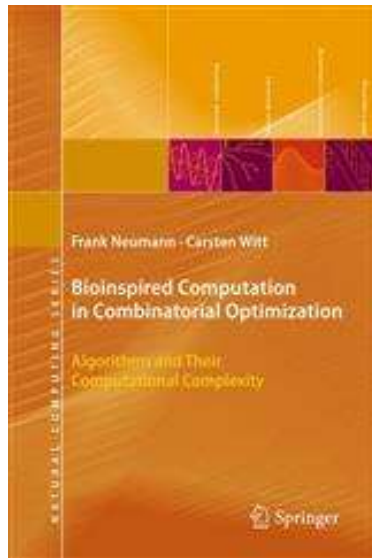
- Theoretical research gives **deep insights in the working principles** of EC, with results that are of a different nature than in experimental work
  - “**very true**” (=proven), but often apply to idealized settings only
  - **for all instances and problem sizes**, but sometimes less precise
    - often only asymptotic results instead of absolute numbers
  - proofs tell us **why certain facts are true**
- The different nature of theoretical and experimental results implies that a real understanding is best obtained from a combination of both
- **Theory-driven curiosity** and the **clarifying nature of mathematical proofs** can lead to new ideas, insights and algorithms

# Summary (2):

## How to Use Theory in Your Work?

- **Try to read theory papers** (or listen to theory talks), but don't expect more than from other papers
  - Neither a theory nor an experimental paper can tell you the best algorithm for your particular problem, but both can suggest ideas
- **Try “theory-style thinking”**: take a very very simplified version of your problem and imagine what could work and why
- **Don't be shy to talk to the theory people!**
  - they will not have the ultimate solution and their mathematical education makes them very cautious presenting an ultimate solution
  - but they might be able to prevent you from a wrong path or suggest alternatives to your current approach

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



- Neumann/Witt (2010). Bioinspired Computation in Combinatorial Optimization, Springer
- Auger/Doerr (2011). Theory of Randomized Search Heuristics, World Scientific
- Jansen (2013). Analyzing Evolutionary Algorithms, Springer
- Doerr/Neumann (2020). Theory of Evolutionary Computation – Recent Developments in Discrete Optimization, Springer. **Free electronic version at [http://www.lix.polytechnique.fr/Labo/Benjamin.Doerr/doerr\\_neumann\\_book.html](http://www.lix.polytechnique.fr/Labo/Benjamin.Doerr/doerr_neumann_book.html)**