Theory of Randomized Search Heuristics

Benjamin Doerr
École Polytechnique
Institut Polytechnique de Paris

Outline

- Part 1: What a typical result in the theory of randomized search heuristics can look like
- Part 2: AofA meets heuristics
- Part 3: Simple problems that are difficult
- Part 4: State of the art

Part 1: Example for a Theoretical Work on Evolutionary Algorithms

- Plan for this part of the talk: Show to you how a useful theory result in this area can look like (but not all results have to follow this scheme).
  - Analyze how a very simple heuristic solves a very simple problem (proven results).
    → Simplicity of the setting necessary for a strong math. analysis.
  - From the result and the proof, obtain insights that could be true in more complex settings.
    → Here the clarifying nature of the maths is crucial!
  - Follow-up work (by others): Validate these insights on real-world settings.

A Very Little Background on Evolutionary Algorithms

- Evolutionary algorithms (EAs) solve problems by setting up an evolutionary process with solution candidates as individuals.
  - Hope: After some time, the individuals in the process represent good solutions to the problem one wants to solve.

- Very successful in practice – not because mimicking evolution gives some mysterious computational advantage, but because practitioners find it easy to design good algorithms in this paradigm.

How Does an EA Look Like?

- Initialization: Put μ random solutions in the parent population P_0.
- For $t = 1,2,3, \dots$ do
  - From suitably selected* parents from P_{t-1}, create λ offspring via
    - mutation: small random modification of a parent;
    - crossover: random mix of two parents.
  - From the μ parents and λ offspring, select* μ individuals as next parent population P_t.
  - If *terminate*, then output the best solution ever seen and STOP

- By setting the parameters and choosing suitable selection, mutation, and crossover operators, the EA can be adjusted to the problem to be solved.

*) Selections may depend on the solution quality. By this the EA "sees" the problem.

Research Question Discussed Now: What is the Right Way of Doing Mutation?

1	0	0	1	0	1

- We only regard *bit-string representations*:
  - Solutions are described via bit-strings of length n.
  - The most common representation for EAs.

- **General recommendation:** *Bit-wise mutation*
  - Obtain the offspring by flipping each bit of the parent independently with some probability p ("mutation rate").
  - Global operator: from any parent you can generate any offspring → Algorithms cannot get stuck forever in a local optimum.

- **General recommendation:** Use a small mutation rate like $p = 1/n$.
  - → In expectation, you flip one bit.

Informal Justifications for $p = 1/n$

- Imitate local search / hill-climbing: A mutation rate of $1/n$ maximizes the probability $p(1-p)^{n-1}$ to flip a single bit.

- Mutation is destructive: If your current search point x has a Hamming distance $H(x, x^*)$ of less than $n/2$ from the optimum x^*, then the offspring y has (in expectation) a larger Hamming distance and this increase is proportional to p:
  - $E[H(y, x^*)] = H(x, x^*) + p(n - 2H(x, x^*))$

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

Proven Results Supporting $p = 1/n$

- For some very simple test-cases, a mutation rate of $1/n$ (or close to that) was proven to give the asymptotically optimal expected runtime.
  - (1+1) EA optimizes OneMax.
  - (1+1) EA optimizes LeadingOnes (optimal mutation rate $1.59/n$).
  - (1+1) EA optimizes a pseudo-Boolean linear functions.
  - (1+1) EA optimizes a monotonic function.
  - (1+λ) EA with $\lambda \leq \ln n$ optimizes OneMax.

- No convincing result that proves a different mutation rate to be preferable.

Our Work*

- Previous state of the art:
  - Strong belief that bit-wise mutation with rate $1/n$ is the best way to mutate solutions.
  - Based on informal considerations and mathematical proofs on very simple problems (without local optima).

- Our work: Conduct a mathematical runtime analysis on a classic benchmark that has local optima and see what is the best mutation rate.
  - Algorithm: (1+1) EA [next slide]
  - Benchmark problem: Jump functions [slide after next slide]

*) Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, Ta Duy Nguyen: Fast genetic algorithms. Genetic and Evolutionary Computation Conference (GECCO) 2017, pages 777-784. ACM

(1+1) Evolutionary Algorithm

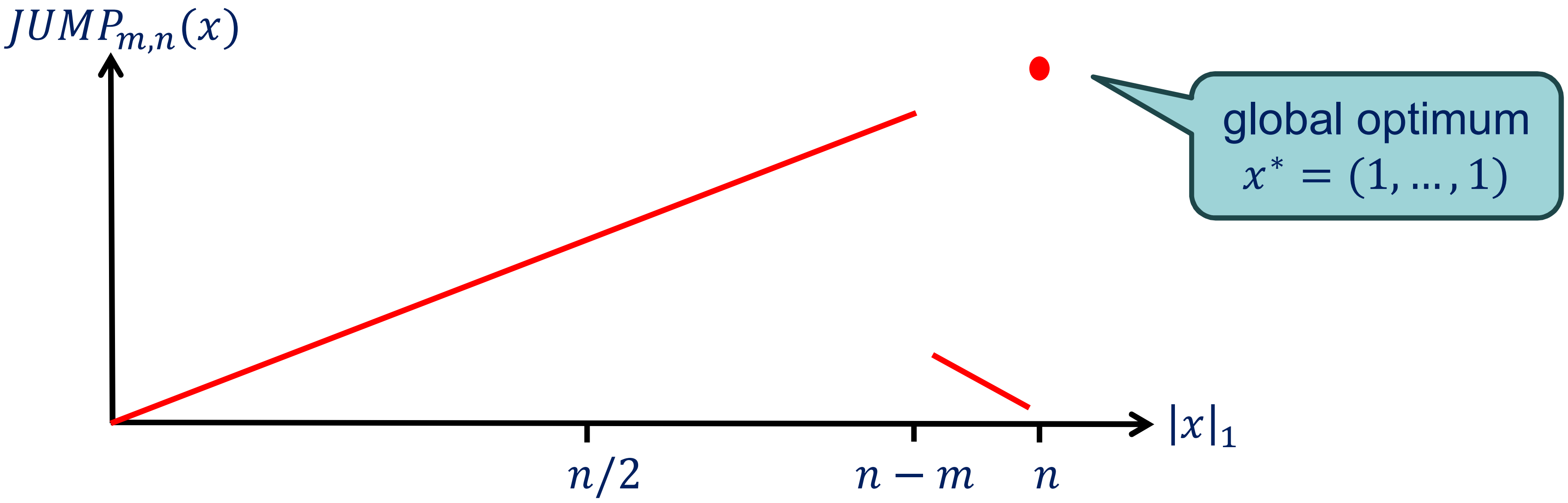
- **(1+1) EA with mutation rate $\boldsymbol{p}$, maximizing $\boldsymbol{f} : \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}} \to \mathbb{R}$**
- Choose $x \in \{0,1\}^n$ uniformly at random
- For $t = 1,2,3, \ldots$ do
  - $y \coloneqq \mathrm{mutate}(x)$ % flip each bit of x independently with prob. p
  - if $f(y) \geq f(x)$ then $x \coloneqq y$

- A very simple algorithm…
  - to enable a mathematical analysis,
  - to study mutation in isolation.

- Runtime: First time t at which an optimum of f is generated.

Jump Functions Benchmark

- $JUMP_{m,n}$: fitness $f(x)$ of an $x \in \{0,1\}^n$ is the number $|x|_1$ of ones, except if $|x|_1 \in \{n-m+1, \dots, n-1\}$, then $f(x) = n - |x|_1$



- 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 runtime 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 $p \leq 1/2$,

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

- Proof of the upper bound:
  - The local optimum is reached in expected time $\frac{2\ln\frac{n}{m}}{p(1-p)^{n-1}}$ [details omitted]
  - The probability that a solution on the local optimum (having m zeroes and $n-m$ ones) is mutated into the global optimum is $p^m(1-p)^{n-m}$.
  - Hence another $\frac{1}{p^m(1-p)^{n-m}}$ iterations (in expectation) to find the optimum.

Optimal Mutation Rate

- Theorem: If $m \le n/4$, then the optimal mutation rate p_{opt} satisfies
$$p_{\text{opt}} = \big(1 \pm o(1)\big)\frac{m}{n}$$
and give an expected runtime of
$$T_{\text{opt}} := T_{p_{\text{opt}}} = \big(1 + o(1)\big) T_{m/n}.$$

- → **The optimal mutation rate is very different from $\boldsymbol{1/n}$!**
- → **We are not talking about peanuts. For $\boldsymbol{m = o(n)}$:**
  - $\boldsymbol{T_{1/n} = \big(1 + o(1)\big)\, e\, n^m}$,
  - $\boldsymbol{T_{m/n} = \big(1 + o(1)\big) \left(\frac{e}{m}\right)^m n^m}$.

Missing the Optimal Mutation Rate

- Theorem: If $p \geq (1+\varepsilon)(m/n)$ or $p \leq (1-\varepsilon)(m/n)$, then
$$T_p(m,n) \geq \frac{1}{6} \exp\left(\frac{m\,\varepsilon^2}{5}\right) T_{\mathrm{opt}}(m,n).$$

- Very bad news: In a practical application, you cannot tell what is the "m". The estimates above show that guessing it wrong is very costly.

- (Obvious) math. reason for the dilemma: When flipping bits independently, the Hamming distance $H(x,y)$ of parent x and offspring y is strongly concentrated around the mean.

- **→ Maybe bit-wise mutation is a bad idea?**

Solution: A New Mutation Operator

- Recap: What do we need?
  - No strong concentration of $H(x, y)$
  - Larger numbers of bits flip with reasonable probability
  - 1-bit flips occur with constant probability (→ easy hill-climbing)

- Solution: *Heavy-tailed mutation* (with parameter $\beta > 1$, say $\beta = 1.5$).
  - Choose $\alpha \in \{1, 2, \dots, n/2\}$ randomly with $\Pr[\alpha] \sim \alpha^{-\beta}$ [power-law].
  - Perform bit-wise mutation with mutation rate α/n.

- Some maths:
  - The probability to flip k bits is $\Theta\left(k^{-\beta}\right)$. → No strong concentration ☺
  - $\Pr[H(x, y) = 1] = \Theta(1)$, e.g., ≈32% for $\beta = 1.5$ (≈37% for classic mut.)

Heavy-tailed Mutation: Results

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

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

- One size fits all: Without "knowing" m, this mutation operator leads to almost the optimal runtime.
  - Price for "one size fits all": a factor of $m^{\beta-0.5}$. Small compared to the losses from choosing a sub-optima mutation rate.
  - Still a speed-up by a factor of $m^{\Theta(m)}$ compared to the classic mutation.

- Good news: This works in practice. Olivier Teytaud (Meta Research, Paris) is a big fan of this mutation operator.

Lower Bound on the *Price for one-size fits all*

- The loss of slightly more than $\Theta(m^{0.5})$ – by taking $\beta = 1 + \varepsilon$ – is unavoidable in a one-size fits all solution.

- Theorem: Let n be sufficiently large. Let D be any distribution on the mutation rates in $[0, 1/2]$. Let $T_D(m,n)$ be the expected runtime of the (1+1) EA which in each iteration samples its mutation rate from D, on $JUMP_{m,n}$.
- Then there is an $m \in [2..n/2]$ such that $T_D(m,n) \geq \sqrt{m}\, T_{opt}(m,n)$.

- Proof: Clever averaging argument.

Summary: How Can a Theory Result on Heuristics Look Like?

- We did a precise mathematical runtime analysis of a synthetic scenario (which was rather elementary here).
- From the result and the maths behind, we saw a general problem (the optimal mutation rate is very problem-specific) and developed a solution (heavy-tailed mutation operator).
- We proved that it solves the problem in the synthetic setting and, again from understanding the maths, we are confident that this solution also works in practice.

Part 2: AofA Meets Heuristics Theory

- While not the typical result in the theory of randomized search heuristics, AofA-style technique have been used to analyze randomized search heuristics.

- Hsien-Kuei Hwang, Alois Panholzer, Nicolas Rolin, Tsung-Hsi Tsai, and Wei-Mei Chen. Probabilistic analysis of the (1+1)-evolutionary algorithm. *Evolutionary Computation*, 26:299–345, 2018.
  - Very precise analysis of how the (1+1) EA optimize the OneMax problem via "matched asymptotics".
  - Very precise analysis of how the (1+1) EA optimizes the LeadingOnes problem via generating functions and recurrence relations.

(1+1) EA optimizes OneMax: Difficulties

- OneMax problem: $f: \{0,1\}^n \to \mathbb{N}_0; x \mapsto \|x\|_1$ "counting the ones"

- **(1+1) EA with mutation rate $1/n$, maximizing $f: \{0,1\}^n \to \mathbb{R}$**
- Choose $x \in \{0,1\}^n$ uniformly at random
- For $t = 1,2,3,\dots$ do
  - $y := \mathrm{mutate}(x)$ % flip each bit of x independently with prob. $1/n$
  - if $f(y) \geq f(x)$ then $x := y$

- The probability to mutate an x with m zeroes into a y with $m - \ell$ zeroes, is

$$\lambda_{n,m,\ell} = \sum_{0 \leqslant j \leqslant \min\{n-m, m-\ell\}} \binom{n-m}{j} \left(\frac{1}{n}\right)^{j} \left(1 - \frac{1}{n}\right)^{n-m-j} \binom{m}{j+\ell} \left(\frac{1}{n}\right)^{j+\ell} \left(1 - \frac{1}{n}\right)^{m-j-\ell}$$

- For $m = \Theta(n)$ and $\ell = \Theta(1)$, this is $\Theta(1)$, hence relevant for the $\Theta(n)$ lower order term in $E[T] = en \ln n + c_1 n + 0.5\, e \ln n + c_2 + O(\log(n)/n)$.

More AofA-Style Results?

- So far: Only two AofA-style results on randomized search heuristics: Runtime analysis of the (1+1) EA with mutation rate $1/n$ optimizing OneMax and LeadingOnes.

- Do these methods also work for other problems? For example…

  - Runtime analysis of the (1+1) EA with the heavy-tailed mutation operator?

  - Runtime analyses for the (1+λ) EA, which generates λ offspring in each iteration and let the best of parent and offspring survive?

Part 3: Mathematically Challenging Problems

- One motivation for me to work in this field is that even very basic problem can be mathematically very challenging.
  - Sometimes, even the answer is not clear before you have a proof.

- Examples: Runtime of the (1+1) EA on…
  - Onemax [just discussed],
  - linear functions,
  - monotonic functions,
  - the minimum spanning tree problem.

Linear Functions

- Definition: Let $w_1, \dots, w_n > 0$. Then

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

  is called a (pseudo-Boolean) *linear function*.
  - "Same as OneMax, but now the bits may have different weights."

- Question: What is the runtime of the (1+1) EA on such a linear function?

- Answer: For all linear functions, $E[T] = en \ln n \pm O(n)$.
- But no intuitive reason and no simple proof.

Very Different Linear Functions

- **Example 1:** OneMax, the function counting the number of 1s in a string, $\mathrm{OM}: \{0,1\}^n \to \mathbb{R}, (x_1, \dots, x_n) \mapsto \sum_{i=1}^n x_i$.
  - Perfect fitness distance correlation: if a search point has higher fitness, then it is closer to the global optimum.

- **Example 2:** BinaryValue, the function mapping a bit-string to the number it represents in binary, $\mathrm{BV}: \{0,1\}^n \to \mathbb{R}, (x_1, \dots, x_n) \mapsto \sum_{i=1}^n 2^{n-i} x_i$.
  - Very low fitness-distance correlation:
    - $\mathrm{BV}(10 \dots 0) = 2^{n-1}$, distance to optimum is $n-1$,
    - $\mathrm{BV}(01 \dots 1) = 2^{n-1} - 1$, distance to optimum is 1.

A Glimpse on the Proof

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

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

    → expected artificial fitness distance reduces by a factor of $\left(1 - \frac{1}{4en}\right)$.
- 3rd proof idea: Translate this expected progress a runtime bound ("drift analysis").
- Note: To prove the tight $en\ln n \pm O(n)$ bound, one has to choose the artificial weights dependent on the linear function f ☹.

References

- First (and complicated) proof of an $O(n \log n)$ upper bound for all linear functions: Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.
- Proof via multiplicative drift: Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. *Algorithmica*, 64:673–697, 2012.
- Proof of this result for all mutation rates c/n, c a constant: Benjamin Doerr and Leslie A. Goldberg. Adaptive drift analysis. *Algorithmica*, 65:224–250, 2013.
- Proof the $\big(1 \pm o(1)\big)e\, n \ln n$ bound: Carsten Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability & Computing*, 22:294–318, 2013.

Monotonic Functions

- Definition: A function $f: \{0,1\}^n \to \mathbb{R}$ is called *monotonic* if the f-value strictly increases whenever you flip a zero in the argument to one.

- Question: What is the runtime of the (1+1) EA on a monotonic function?

- Answer: We do not know. Known results for mutation rate c/n:
  - For $0 < c < 1$, the expected runtime on all monotonic fcts. is $\Theta(n \log n)$.
  - For $c = 1$, it is $\Omega(n \log n)$ and $O(n \log^2 n)$ [not known if tight].
  - For $c > 2.13 \ldots$, there are monotonic functions such that the runtime is exponential in n. SURPRISE! Proof via a randomized construction.

References

- First proof that monotonic functions can be difficult to optimize (when the mutation rate is $16/n$ or higher): Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges. Mutation rate matters even when optimizing mono tone functions. *Evolutionary Computation*, 21:1–21, 2013.
- Proof that this problem appears already for mutation rates above $\approx 2.13/n$: Johannes Lengler and Angelika Steger. Drift analysis and evolutionary algorithms revisited. *Combinatorics, Probability & Computing*, 27:643–666, 2018.
- Proof of the $O(n \log^2 n)$ bound for mutation rate $1/n$: Johannes Lengler, Anders Martinsson, and Angelika Steger. When does hillclimbing fail on monotone functions: an entropy compression argument. In *Analytic Algorithmics and Combinatorics, ANALCO 2019*, pages 94–102. SIAM, 2019.

Minimum Spanning Trees

- Let $G = ([1..n], E)$ be an undirected graph with edge weights $w: E \to \mathbb{N}$.
- A minimum spanning tree of G is a subset $F \subset E$ of edges such that $([n], F)$ is connected and has minimal weight $w(F) = \sum_{e \in F} w(e)$ among all such F.
- Using the natural correspondence between bit-string of length $|E|$ and subsets of E, and giving a high penalty for each extra connected component, this problem can be formulated as minimization problem over $\{0,1\}^{|E|}$.

- Question: What is the expected runtime of the (1+1) EA on this problem?

- Answer: We do not know. Known bounds are
  - $O(|E|^2 \log(n\, w_{\max}))$, where $w_{\max}$ is the maximum edge weight,
  - $\Omega(|E|^2 \log n)$.

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

Part 4: State of the Art

- We can analyze…
  - evolutionary algorithms with non-trivial parent and offspring populations on classic single- and multi-objective benchmarks and polynomial-time solvable combinatorial optimization problems
    - also in the presence of noise;
    - also with dynamic parameter choices;
  - ant-colony optimizers and estimation-of-distribution algorithms on many of the classic benchmarks;
  - the Metropolis algorithm and simulated annealing on some problems.

Part 4: State of the Art (2)

- We struggle with
  - weighted versions of problems (linear functions, minimum spanning trees, shortest paths);
  - understanding the precise population dynamics, and thus with showing advantages of crossover (offspring is a mix of two parents) and advantages from larger population sizes;
  - understanding how evolutionary algorithms can profit from non-elitism (forgetting the current-best solution).

Summary and Conclusion

- I have shown to you
  - what theoretical research on heuristics can look like: analyze a simplified setting with mathematical means and learn from this;
  - that AofA and heuristics theory are not disjoint;
  - that there are many “simple” problems waiting for a clever solution;
  - what is the state of the art: we can analyze certain heuristics, but we fail to explain many things the practitioner do.

- [My] conclusion: This is a young area with open problems that are both mathematically attractive and have the potential to have a broader impact.
  - Also, this is an applied area that is very open to theoretical work ☺.

谢谢! Thanks!

Further Reading

- Benjamin Doerr and Frank Neumann, editors. Theory of Evolutionary Computation—Recent Developments in Discrete Optimization. Springer, 2020. Freely available at http://www.lix.polytechnique.fr/Labo/Benjamin.Doerr/doerr_neumann_book.html.

Appendix: Multiplicative Drift Theorem

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

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

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

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

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

- Proof of the linear functions result: Use
  - $\delta = 1/4en$,
  - $X_t = \tilde{f}(x^*) - \tilde{f}\left(x^{(t)}\right)$,
  - and the estimate $X_0 \leq 2n$.