



Tutorial: Theory of Evolutionary Computation

Anne Auger (INRIA Saclay)
&
Benjamin Doerr (École Polytechnique)

PPSN 2014, Ljubljana, Slovenia

Target for This Tutorial

- Give an introduction to EA theory aimed at an audience that is not working in theory already.
 - What is theory?
 - Why do theory?
 - How can the general EA community profit from theory?

- Feel free to ask questions at any time!

- We're happy to receive all kinds of feedback.

Outline

- [b10:35] Introduction: What do we mean by theory? Why? Why not?
- [b10:50] How theory can help understand EC: 3 examples
 - Warning: What looks easy might be difficult
 - Choosing good representations, operators, ...
 - Do we need crossover?
- [a11:10] Basics: from discrete to continuous optimization
 - “interesting” theoretical questions and their relationship to practice
- [a11:30] Linear convergence of adaptive algorithms
 - illustrate benefits and limitation of theory wrt experiments
- [a11:50] Information geometry perspective
 - where theory sheds new light on “old” algorithms and gives new perspectives for algorithm design
- [b12:10] The F-words in theory

What Do We Mean With *Theory*?

- Definition (for this tutorial): By theory, we mean **results proven with mathematical rigor** and nothing else.
- Mathematical rigor:
 - make precise the 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: The (1+1) EA finds the optimum of the OneMax test function $f: \{0,1\}^n \rightarrow \mathbb{R}; x \mapsto \sum_{i=1}^n x_i$ in an expected number of at most $en \ln(n)$ iterations.
Proof: blah, blah, ...

Theory, Theories, and Everything Else

- **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: parts of landscape analysis
- **“Theories”**: Unproven claims guiding our thinking
 - example: building block hypothesis

Theory, Theories, and Everything Else

- 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: parts of landscape analysis
- “Theories”: Unproven claims guiding our thinking
 - example: building block hypothesis

Why Do Theory? – Results

- **Absolute guarantee** that the result is correct
 - for yourself
 - for reviewers and journal
 - for the reader of your papers: anyone with moderate maths skills can fully check your result

- **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? – 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 results, you are automatically pointed led to the questions that need more thought
- Trigger for new ideas
 - clarifying nature of mathematics
 - playful nature of mathematicians

The Price for all This

Theory results are very true, very trustworthy, very exact. This comes at a price... Possible drawbacks include:

- **Restricted scope:** So far, mostly simple algorithms (e.g., (1+1) EA) on simple problems (test functions, “easy” graph problems) could be analyzed
- **Less precise results:** constants are not tight, or not explicit as in “ $O(n^2)$ ” = “less than cn^2 for some unspecified constant c ”
- **More general than you want:** You get a weaker statements for all problem instances instead of a strong one for the practically more relevant ones
- **Theory results can be very difficult to obtain**
 - the proof might be short and easy to read, but finding it took long hours

Theory and Experiments: Complementary Results

THEORY

- cover all problem instances of arbitrary sizes
→ guarantees
- proof tells you the reason
- only models for real-world instances, and even this is hard
- limited scope
- limited precision
- implementation independent
- finding proofs can be difficult

EXPERIMENTS

- only a finite number of instances of bounded size
→ have to hope that this is representative
- only tells you numbers
- real-world instances
- everything you can implement
- exact numbers
- depends on implementation
- often cheap to do

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

What Can Theory Do For You?

3 Discrete Examples

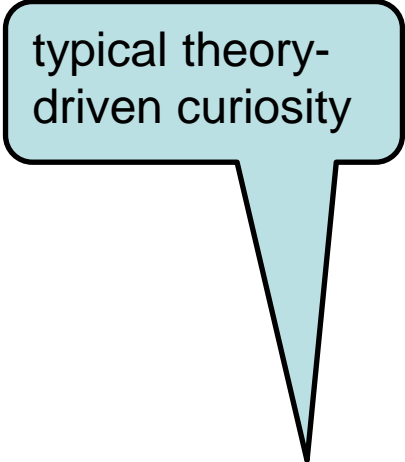
- Debunk misconceptions: What looks easy can be hard
- Give advice how to design EAs
- Contribute to the discussion how useful crossover is

Example 1: Prevent Misconceptions

- **Misconception: Functions without local optima are easy to optimize**
- **Horn, Goldberg, Deb (PPSN'94), Rudolph (1997), Droste, Jansen, Wegener (PPSN'98):** There is a function $f: \{0,1\}^n \rightarrow \mathbb{R}$ such that
 - f has no local optima: If $f(x)$ is not maximal, then by flipping a single bit of x you can get a better solution
 - several common EAs need time exponential in n with high probability
- **D., Jansen, Sudholt, Winzen, Zarges (PPSN'10):** There is a function $f: \{0,1\}^n \rightarrow \mathbb{R}$ such that
 - f is strictly monotonic: if you obtain y from x by flipping any zero to one, then $f(y) > f(x)$ [super-easy, no local optima, $(1, \dots, 1)$ is unique opt.]
 - the $(1+1)$ EA with mutation probability $16/n$ needs time exponential in n with high probability

Example 2: Help in the Design of EA

- Example: Several theoretical works on shortest path problems
 - Scharnow, Tinnefeld, Wegener (PPSN'02)
 - D., Happ, Klein (CEC'07)
 - Baswana et al. (FOGA'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
- D., Johannsen (GECCO'10): 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 $\approx \frac{|V|^2}{|E|}$



typical theory-
driven curiosity

Example 2: Help in the Design of EA

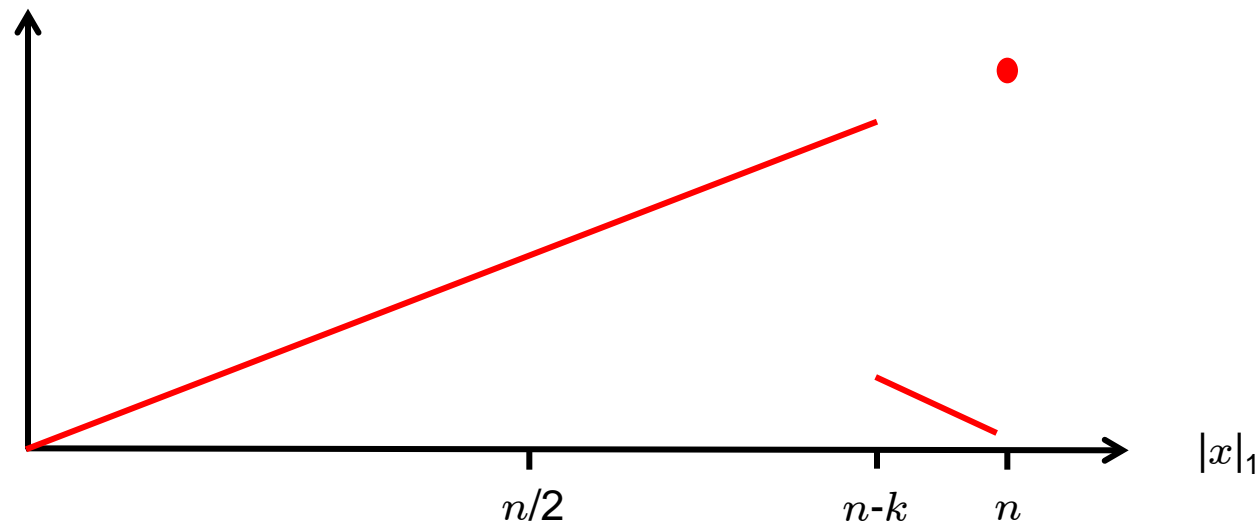
- Theory result (proven, can be made precise): Going from a vertex-based to an edge-based representation speeds up several existing algorithms.
- Message beyond this particular result: Possibly, edge-based representations can be superior also for other graph problems
- Many more theory results giving advice/suggestions to the algorithm designer:
 - representations for the Eulerian cycle problem (where a permutation of edges is asked for)
 - optimal mutation probabilities
 - different selection mechanisms
 - ...

Example 3: Do We Need Crossover?

- Applied work: Using crossover and mutation together seems to work well.
- Holland's (1975) building block hypothesis (BBH):
 - A genetic algorithm works by combining short, low-order, highly fit schemata (“building blocks”) into fitter higher order schemata.
- Forrest, Mitchell (1993): Disprove BBH experimentally *
 - design a simple optimization problem that perfectly fulfills the BBH
 - experiments: a simple hill-climber is much faster than a standard crossover-based algorithm □
- What can theory add to this debate?
 - Possible first tiny step: Find a problem and a reasonable EA such that the EA with crossover performs better than without

First Theory Contribution

- Jansen&Wegener (1999): Jump functions.
 - fitness of an n -bit string x is the number of ones, except if this is in $\{n-k, \dots, n-1\}$, then the fitness is the number of zeroes.



- Hope: One quickly finds many bit-strings with $n - k$ ones, then crossover combines two of these into the optimum $x^* = 11\dots 1$
- Analysis: Mutation needs $\Theta(n^k)$ iterations, with uniform crossover $O(n^2 \log n)$ suffice, **but only for very small crossover rates ($\leq 1/n$)**

More Theory on Crossover

- Sequence of follow-up works
 - Fischer&Wegener (GECCO 2004)
 - Storch&Wegener (TCS 2004)
 - Sudholt (GECCO 2005)
 - Jansen&Wegener (Disc. Appl. Math. 2005)
- ...but all regard quite artificial problems
 - “It will take many major steps to prove rigorously that crossover is essential for typical applications.” (Jansen&Wegener (2005))
- Insight: If it is so hard to find a single convincing problem where crossover provably helps, maybe crossover is not so important?

Good News: Crossover Can Work

- First classic optimization problem where crossover provably gives a speed-up (of around a factor of \sqrt{n}): All-pairs shortest path problem ☺
[D., Happ, Klein (Gecco'08), D., Theile (Gecco'09), D., Johannsen, Kötzing, Neumann, Theile (PPSN'10)]
 - proof also reveals why crossover works here: crossover finds some good solutions, mutation “fills the gaps” and finds the rest
 - not primarily “putting together of building blocks”
- More recent works:
 - Sudholt (Gecco'12): Crossover can even help for the simple OneMax problem (constant factor improvement)
 - D., Doerr, Ebel (Gecco'13): $\sqrt{\log n}$ factor improvement for OneMax
 - working principle: crossover as repair mechanism (known from ES, first time in discrete optimization)

Example 3: Summary

Theory work on crossover:

- It was hard to find a single convincing example where crossover was provably helpful. Possibly, **crossover is not so important, or it gives improvements too small to be visible with theory methods** (e.g., constant factors improvements or lower order terms)
- The few examples where crossover works indicate that **other principles than the building block hypothesis are more relevant.**

The F-Words in Theory

- Besides (hopefully) short- or medium-term useful results, theory also asks and tries to answer questions that are

FFUNDAMENTAL

- Besides being difficult, mathematical, tricky, full of negative surprises, theory can be

FFUN

- I'll now show you how to bring the two together

A Fundamental Question

- How difficult is a problem for evolutionary methods?
- Since “evolutionary methods” is hard to define precisely (and we do theory), let us broaden the scope to black-box optimization.
- **Black-box optimization:** you do not have access to an explicit problem description, but all you can do is evaluate solution candidates.
 - evolutionary algorithms
 - ant colony optimization
 - local search, random search, ...
- How difficult is a problem for black-box optimization = what is the performance of the best black-box algorithm for this problem?
 - black-box complexity (BBC) of the problem

Example: BBC of Needle Functions

- Needle functions: For all $z \in \{0,1\}^n$, let

$$f_z : \{0,1\}^n \rightarrow \{0,1\}; x \mapsto \begin{cases} 1, & x = z \\ 0, & x \neq z \end{cases}.$$

- How many fitness evaluations do you need to find the maximum of *any* of these functions?
- First answer: You can try all bit-strings one after the other. After 2^n fitness evaluations, you surely found the optimum.
 $\rightarrow BBC \leq 2^n$.

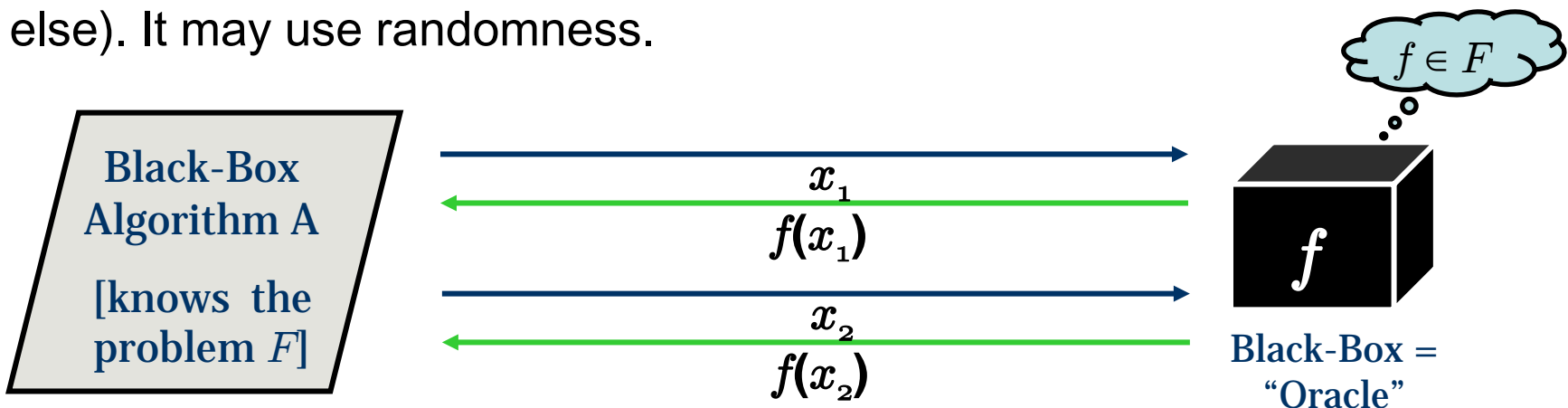
Definition of BBC

- This example made precise: For all $z \in \{0,1\}^n$, let

$$f_z : \{0,1\}^n \rightarrow \{0,1\}; x \mapsto \begin{cases} 1, & x = z \\ 0, & x \neq z \end{cases} \quad \text{“instance”}.$$

Let $F := \{ f_z \mid z \in \{0,1\}^n \}$ “problem”.

- A **black-box algorithm A for F** takes any $f_z \in F$ (blindly) as input. It may evaluate any $x \in \{0,1\}^n$ in a black-box fashion (it learns $f_z(x)$, but nothing else). It may use randomness.



Definition of BBC (2)

- This example made precise: For all $z \in \{0,1\}^n$, let

$$f_z : \{0,1\}^n \rightarrow \{0,1\}; x \mapsto \begin{cases} 1, & x = z \\ 0, & x \neq z \end{cases} \quad \text{“instance”}.$$

Let $F := \{ f_z \mid z \in \{0,1\}^n \}$ “problem”.

- A **black-box algorithm A for F** takes any $f_z \in F$ (blindly) as input and tries to find the optimum by evaluating search points.
 - $T(A, f_z) :=$ **Expected** number of evaluations until $OPT(f_z)$ is evaluated
 - $T(A, F) := \max_{f_z \in F} T(A, f_z)$
 - $BBC(F) := \min_A T(A, F) =$ “**min number of fitness evaluation to solve F** ”
- $BBC(F) \leq 2^n$, because “ask all $x \in \{0,1\}^n$ in some fixed order” for any $f_z \in F$ needs at most 2^n fitness evaluations to query the optimum z of f_z

Lower Bounds

- Definition (reminder):
 - $T(A, f_z) :=$ **Expected** number of evaluations until $OPT(f_z)$ is evaluated
 - $T(A, F) := \max_{f_z \in F} T(A, f_z)$
 - $BBC(F) := \min_A T(A, F) =$ “**min number of fitness evaluation to solve F** ”

- To prove a lower bound “ $BBC(F) \geq xxx$ ”, we need to show that *any black-box algorithm* A needs at least xxx fitness evaluations (for some f_z)

- For needle functions: $BBC(F) \geq (2^n + 1)/2 \approx 2^{n-1}$.
 - very informal argument: can't do better than trying random search points (without repetition)
 - Corollary: **Any EA, GA, ACO algorithm, swarm intelligence method etc. needs at least $(2^n + 1)/2 \approx 2^{n-1}$ fitness evaluations to solve the Needle problem. Universal lower bound 😊**

Lower Bound For Needle (Formal)

- Reminder: To prove a lower bound “ $BBC(F) \geq xxx$ ”, we need to show that any black-box algorithm needs at least xxx fitness evaluations (for some f_z)
- Theorem: For needle functions, we have $BBC(F) \geq (2^n + 1)/2$.
- Proof:
 - Let A be any black-box algorithm for needle functions.
 - Let x_1, x_2, x_3, \dots be the sequence of search points A evaluates. Note that this does not depend on f_z (except that we stop when $x_i = z$). Hence we may assume that A chooses (possibly randomly) an enumeration x_1, x_2, x_3, \dots of the search space $\{0,1\}^n$ and then evaluates x_1, x_2, x_3, \dots in that order until the optimum is found.
 - Some maths: For any random sequence x_1, x_2, x_3, \dots there is an f_z such that the expected position of z in the sequence is at least $(2^n + 1)/2$.

Example 2: OneMax Functions

- OneMax functions: For all $z \in \{0,1\}^n$, let

$$f_z : \{0,1\}^n \rightarrow \{0,1\}; x \mapsto eq(x, z) := |\{i \mid x_i = z_i\}| \quad \text{“\# bits in which } x, z \text{ agree”}$$

[hence $f_{(1,\dots,1)}$ is the classic OneMax function counting the 1s in the bit-string]

- What is the BBC of the OneMax functions?
- Upper bound 1: The (1+1) evolutionary algorithm optimizes any f_z in $en \ln(n)$ iterations $\rightarrow BBC \leq en \ln(n)$.
- Upper bound 2: You can learn the bits of z one after the other. E.g. if $f_z(1x_2 \dots x_n) \geq f_z(0x_2 \dots x_n)$, then $z_1 = 1$, else $z_1 = 0$. $\rightarrow BBC \leq n + 1$.
- Theorem [Erdős, Rényi '63]: $BBC(OneMax) = \Theta\left(\frac{n}{\log(n)}\right)$.**

Example 2: OneMax Functions

- **Theorem [Erdős, Rényi '63]:** $BBC(OneMax) = \Theta\left(\frac{n}{\log(n)}\right)$.
 - There is a black-box algorithm finding the optimum z of any OneMax function $f_z : \{0,1\}^n \rightarrow \{0,1\}; x \mapsto eq(x, z) := |\{i \mid x_i = z_i\}|$ with only $2n/\log_2 n$ fitness evaluations.
 - There is no algorithm doing better than $n/\log_2 n$.

- Note: There are papers presenting faster algorithms for OneMax
 - these algorithms cannot work for all OneMax functions (**because of the theorem above**)
 - typically, these algorithms are designed for the “counting 1s” function only
 - in a sense, they derive their better performance by exploiting the fact that they know the optimal solution $(1, \dots, 1)$ already

Example 2: OneMax Functions

- Theorem [Erdős, Rényi '63]: $BBC(OneMax) = \Theta\left(\frac{n}{\log(n)}\right)$.
- ER'63-algorithm:
 - Evaluate $2n/\log_2 n$ random search points x_1, x_2, \dots and store their fitnesses y_1, y_2, \dots
 - With high probability, there is only one f_z such that $f_z(x_i) = y_i$ for all x_i
 - Then z is the optimal solution 😊
- Comments:
 - This is an artificial algorithm and not useful for any real problem
 - But this is OK for now, because it tells us the black-box complexity of the OneMax problem.
 - We could use this as a trigger and ask ourselves if there are better EAs for OneMax (indeed, there are)

Fun: Mastermind

- Mastermind: 2-player game
 - *CodeMaker* hides a ~~4~~ ^{n} -digit ~~6~~ ^{k} -color code C .
 - *CodeBreaker* tries to guess it using few guesses
- Guess: Some color code G
- Answer:
 - Number of positions in which C and G agree (“black answer-pegs” [here: red])
 - ~~Number of additional code letters that occur in a wrong position (“white pegs”)~~



Fun: Mastermind

- Mastermind: 2-player game
 - *CodeMaker* hides a n -digit k -color code C .
 - *CodeBreaker* tries to guess it using few guesses
- Guess: Some color code G
- Answer:
 - Number of positions in which C and G agree (“black answer-pegs” [here: red])
- Reminder: OneMax functions: For all $z \in \{0,1\}^n$, let $f_z : \{0,1\}^n \rightarrow \{0,1\}; x \mapsto eq(x, z) := |\{i \mid x_i = z_i\}|$ “# bits in which x, z agree”
- Observation: **BBC(OneMax) \triangleq n -digit 2-color Mastermind!**
 - \rightarrow You can win Mastermind with $2 n / \log_2 n$ guesses

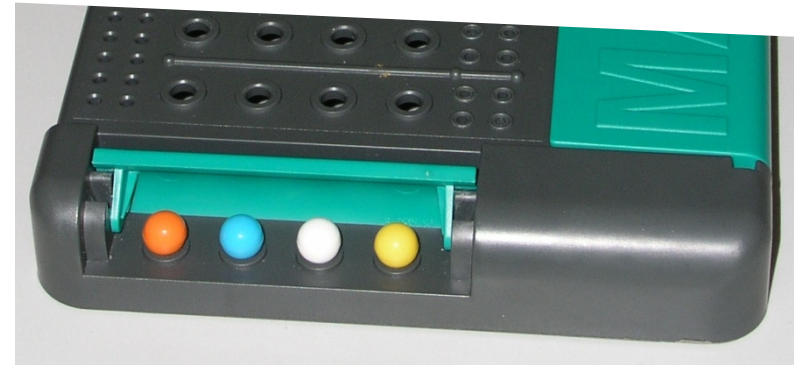


BBC and Population Size

- The OneMax functions have a surprisingly low BBC of only $2n/\log_2 n$.
 - Most evolutionary algorithms need at least $n \ln(n)$ fitness evaluations.
- Question: Can this discrepancy be explained by the fact that the ER'63 black-box algorithm stores $2n/\log_2 n$ search points? Is the black-box complexity larger when we restrict the memory to some $k < 2n/\log_2 n$.
 - Note: Use restricted BBCs to gain fundamental insight about EA parameters, here the population size
- Conjecture (Droste, Jansen, Wegener (2006)): For $k = 1$, the memory restricted BBC is of order $n \log n$.
- Theorem (Doerr&Winzen (2012)): No, even with memory $k = 1$, the BBC remains at a low $cn/\log n$, where c is some (unknown) constant.

Fun Fact: Memory-1 BBC(OneMax) = Mastermind with Only Two Rows

- Precise rules:
 - We start the game with an empty board
 - If there is an empty row, CodeBreaker can enter a guess, which will be answered by CodeMaker
 - If there is no empty row, CodeBreaker must empty one of the two rows and *forget the content*.
- Observation: Black-box algorithms with memory restriction $k = 1$ exactly correspond to strategies for Mastermind with two rows
- **Theorem: CodeBreaker can find the secret code with $O(n / \log n)$ guesses even when there are only two rows.**
- PS: There is a BBC paper at PPSN'14 (by Badkobeh, Lehre, Sudholt), tomorrow first session



Final Summary

- Theory (strict definition): Precise results proven with mathematical rigor.
- Complements well with experiments
 - Theory: safest possible assertion, reader can check the proof, “for all” statements, many problems too complicated to prove exact bounds or something at all
 - Experiments: more error-prone, less reproducibility, finite number of tests, exact numbers, real-world examples
- Achievements:
 - debunk misbeliefs
 - give advice in algorithm design
 - understand working principles of EA
- PS: We’ll put the slides on the PPSN page, sorry for being late.

Hvala!
Thanks!