Objectives of the Tutorial

- This is a tutorial on black-box complexity. This is currently one of the hottest topics in the theory of randomized search heuristics.
- I shall try my best to:
  - tell you on an elementary level what black-box complexity is and how it shapes our understanding of randomized search heuristics
  - give an in-depth coverage of two topics that received much attention in the last few years
    - stronger upper bounds and the connection to guessing games
    - alternative black-box models
  - sketch several open problems
- Don’t hesitate to ask questions whenever they come up!

Bio-Sketch

- Benjamin Doerr is a senior researcher at the Max Planck Institute for Informatics and a professor at Saarland University.
- He received his diploma (1998), PhD (2000) and habilitation (2005) in mathematics from Kiel University.
- Together with Frank Neumann and Ingo Wegener, he founded the theory track at GECCO and served as its co-chair 2007-2009.
- He is a member of the editorial boards of Evolutionary Computation and Information Processing Letters.
- His research area includes theoretical aspects of randomized search heuristics, in particular, run-time analysis and complexity theory.

Agenda

- Part 1: Black-box complexity: A complexity theory for randomized search heuristics (RSH)
  - Introduction/definition
  - Lower bounds for all RSH (example: needle functions)
  - Thorn in the flesh: Are there better RSH out there? (example onemax)
  - Different black-box models – what is the right difficulty measure?
- Part 2: Tools and techniques (in the language of guessing games)
  - From black-box to guessing games
  - A general lower bound
  - How to play Mastermind
  - A new game
- Summary, open problems
**Timeline**

- **2002** Droste, Jansen, Tinnefeld, Wegener. A new framework for the valuation of algorithms for black-box optimization. FOGA
- **2009** Anil, Wiegand. Black-box search by elimination of fitness functions. FOGA
- **2010** Lehre, Witt. Black-box search by unbiased variation. GECCO
- **2011** Doerr, Winzen. Towards a Complexity Theory of Randomized Search Heuristics: Ranking-Based Black-Box Complexity. CSR
- **2011** Rowe, Vose. Unbiased black box search algorithms. GECCO
- **2012** Doerr, Kötzing, Winzen. Too Fast Unbiased Black-Box Algorithms. GECCO
- **2012** Doerr, Winzen. Playing Mastermind with constant-size memory. STACS
- **2012** Doerr, Winzen. Reducing the arity in unbiased black-box complexity. GECCO

---

**Part 1: Complexity Theory for RSH**

- Why a complexity theory for RSH?
  - Understand problem difficulty!
- How?
  - Black-box complexity!
- What can we do with that?
  - General lower bounds, thorn in the flesh
- Different notions of black-box complexity

---

**Why a Complexity Theory for RSH?**

- Understand problem difficulty!
  - Randomized search heuristics (RSH) like evolutionary algorithms, genetic algorithms, ant colony optimization, simulated annealing, … are very successful for a variety of problems.
  - Little general advice which problems are suitable for such general methods
  - Solution: Complexity theory for RSH
- Take a similar successful route as classical algorithmics!
  - Algorithmics: Design good algorithms and analyze their performance
  - Complexity theory: Show that certain things are just not possible
  - The interplay between the two areas proved to be very fruitful for the research on classic algorithms

---

**Algorithms vs. Complexity Theory for RSH – An Example**

- **Algorithm Analysis:** Prove how a certain algorithm solves a particular problem.
  - The (1+1) EA finds a minimum spanning tree with an expected number of $O(m^2 \log(m w_{max}))$ fitness evaluations.
- **Complexity Theory:** What can the best possible algorithm for this problem do or not.
  - No RSH can solve the Needle problem in an expected number of less than $(2^n+1)/2$ fitness evaluations.
- Bottom line: Spanning tree is easy for RSH, the Needle problem not.
Reminder: Classic Complexity Theory

- General approach: Complexity (difficulty) of a problem := Performance of the best algorithm on the hardest problem instance
- Example: “Sorting \( n \) numbers needs \( \Theta(n \log(n)) \) pair-wise comparisons.”
  - Problem: “Sorting an array of \( n \) numbers”
  - Instance (input to algorithm): An (unsorted) array of \( n \) numbers
  - Algorithms: All that run on a Turing machine
  - Performance (cost) measure: Number of pair-wise comparisons
    - \( T(A, I) \) = number of comparisons performed when algorithm \( A \) runs on instance \( I \)
  - Theorem: “Complexity of sorting = \( \min_A \max_I T(A, I) = \Theta(n \log(n)) \).”
- How does this work for RSH?
  - Algorithms = RSHs, Performance = number of fitness evaluations, …

Complexity Theory for RSH

- Algorithms: Randomized search heuristics (RSH)
  - may generate solutions and query their fitness
  - no explicit access to the problem description
  - \( \Rightarrow \) black-box optimization algorithm
- Performance measure \( T(A, I) \) = expected number of fitness evaluations until algorithm \( A \) running on instance \( I \) queries an optimum of \( I \)
- Black-box complexity: Expected number of fitness evaluations the best black-box algorithm needs to query the optimum of the hardest instance.
  - \( \min_A \max_I T(A, I) \)

BBC: What Can We Do With It?

- Black-box complexity: Expected number of fitness evaluations the best black-box algorithm needs to query the optimum of the hardest instance.
  - \( \min_A \max_I T(A, I) \)
- 3 uses:
  - Measure for problem difficulty [that’s how we designed the definition]
  - Universal lower bounds [next slide]
  - A thorn in the flesh [next to next slide]

BBC: Universal Lower Bounds

- Black-box complexity: Expected number of fitness evaluations the best black-box algorithm needs to query the optimum of the hardest instance.
  - \( \min_A \max_I T(A, I) \)
- Follows right from the definition: The black-box complexity is a lower bound on the performance of any RSH!
  - \( \text{BBC} := \min_A \max_I T(A, I) \leq \max_B T(B, I) = \text{performance of } B \)
- Example:
  - Theorem [DJTW’02]: The black-box complexity of the needle function class is \( (2^n+1)/2 \).
  - Consequence: No RSH can solve the needle problem in sub-exponential time.
  - One simple proof replaces several proofs for particular RSH 😊
BBC: A Thorn in the Flesh

- If the black-box complexity is lower than what current best RSH achieve, you should wonder if there are better RSH for this problem!
- Example: OneMax functions
  - for all "bit-strings" \( z \in \{0,1\}^n \) let \( f_z : \{0,1\}^n \to \{0,\ldots,n\} ; x \mapsto \text{"number of positions in which } x \text{ and } z \text{ agree"} \)
  - all \( f_z \) have a fitness landscape equivalent to the classic OneMax function (counting the number of ones in a bit-string).
- Theorem [many, see later]: The black-box complexity of the class of all OneMax functions is \( \Theta(n / \log(n)) \).
- But: All standard RSH need at least \( \Omega(n \log(n)) \) time!
- Are there better RSH that we overlooked?
- Same motive as in classical theory: \( n \times n \) matrix multiplication can be done in time \( O(n^{2.372}) \), only lower bound is \( \Omega(n^2) \).

Alternative Black-box Models

- Previous slide: “Are there better RSH?”
- Alternative answer: The black-box model allows too powerful (unnatural) algorithms.
- Next \( x \) slides: Discuss alternative black-box models
  - very active research area in the last 3 years
  - no definitive answer
- Common theme: Instead of allowing all black-box optimization algorithms, only regard a restricted class!
  - restricted class should include most classic RSH

Alternative 1: Unbiased BBC

- Lehre&Witt (GECCO’10 theory track best paper award):
  - allow only unbiased variation operators: treat all bit-positions \( (1, \ldots, n) \) and the two bit-values \( (0, 1) \) equally!
  - equivalent: if \( \sigma \) is an automorphism of the hypercube, then the probability that \( y \) is an offspring of \( x_1, \ldots, x_k \) must be equal to the probability that \( \sigma(y) \) is an offspring of \( \sigma(x_1), \ldots, \sigma(x_k) \)
- Observation: Most RSH are unbiased
  - exception: one-point crossover
- Result: The unbiased, mutation-only BBC of OneMax is \( \Theta(n \log(n)) \)
  - as observed for random local search, (1+1) EA, …
- Anti-result [DKW’11]: Also the TRAP \( k \) function has an unbiased, mutation-only BBC of \( \Theta(n \log(n)) \).
  - contrasts the \( \Omega(n^4) \) performance of all classic RSH
  - Interesting [DJKLW’11]: Unbiased 2-ary BBC of OneMax: \( O(n) \).

Alternative 2: Ranking-Based BBC

- D&Winzen (CSR’11), suggested by Niko Hansen: ranking-based
  - do not regard the absolute fitness values, but make all decisions dependent only on how fitnesses of search points compare!
- Observation: Many RSH follow this scheme
  - exception: fitness-proportionate selection
- Bad news: OneMax has a ranking-based BBC of \( \Theta(n / \log(n)) \)
  - Good news: For BinaryValue…
    - BBC: \( \log(n) \)
    - ranking-based BBC: \( \Omega(n) \)
    - many RSH: \( \Theta(n \log n) \)
- Open problem: Partition…
  - BBC: \( O(n) \), heavily exploits absolute fitness values
  - Unbiased BBC: Maybe exponential?
Alternative 3: Memory-Restricted BBC

  - suggest to restrict the memory: store only a fixed number of search points and their fitness
  - inspired by bounded population size
  - conjecture: with memory one, the BBC of OneMax becomes the desired $\Theta(n \log(n))$

- D&Winzen (STACS’12): Disprove conjecture.
  - Even with memory one, the BBC of OneMax is $\Theta(n / \log(n))$.
    [I’ll give a proof in the second part of the tutorial]

Summary Alternative BBC Models

- Different models:
  - unrestricted (classic)
  - unbiased
  - ranking-based
  - memory-restricted

- None is yet “the ultimate complexity notion” for RSH

- Each expanded our understanding
  - what makes a problem hard
  - what makes a RSH powerful

- Many open problems…

Summary Part 1

- Black-box complexity (BBC): “Minimum number of search points that have to be evaluated to find the optimum”
  - Expected number of fitness evaluations the best black-box algorithm needs to query the optimum of the hardest instance.

- Min, max, $T(A, I)$

- Uses:
  - Measure of problem difficulty
  - Universal lower bounds
  - Thorn in the flesh

- Particular problem: What is the most useful class of black-box algorithms to be regarded?

Part 2: Tools and Techniques

Plan for the 2nd part of this tutorial:

- Explain, why BBC and guessing games are almost the same

- Use the language of guessing games to demonstrate some techniques
  - Random guessing: The BBC of OneMax or “how to play Mastermind with two colors?”
  - A simple “information theoretic” lower bound
  - Clever guessing:
    - Mastermind with $n$ colors
    - [intermediate summary “tools and techniques”]
    - Memory-restricted BBC of OneMax = Mastermind with 2 rows

- A game derived from BBC studies 😊
A Formal Definition of BBC

Optimization problem: A set $F$ of functions $f : \{0,1\}^n \to \mathbb{R}$.

Aim is to find the maximum of a given $f \in F$.

Language:
- An $f \in F$ is called an "instance of $F$"
- $\{0,1\}^n$ "search space"
- $x \in \{0,1\}^n$ "search point"

Example "Maximum Clique": For each graph $G$ on the vertex set $\{1,\ldots,n\}$, $f_G(x)$ is the size of the vertex set represented by $x$, if this is a clique in $G$, and 0 otherwise. $F := \{f_G \mid G \text{ a graph with vertices } 1,\ldots,n\}$.

Black-box algorithm for $F$: A randomized algorithm $A$ that finds the maximum of any $f \in F$ by asking $f$-values of search points only (no explicit access to the instance, e.g., the graph $G$ in the clique example).

Performance $T(A,f)$ of $A$ for $f \in F$: Expected time until an $x$ with $f(x) = \text{OPT}(f)$ is queried.

Performance $T(A,F)$ of $A$ on $F$: max$_{f \in F} T(A,f)$.

BBC of $F$: min$_A T(A,F)$, where $A$ runs over all black-box algorithms for $F$.

From BBC to Guessing Games

Guessing game:
- BlackBox chooses a hidden $f \in F$.
- Algo tries to guess an $x$ with $f(x)$ maximal
- For each incorrect guess, BlackBox tells $f(x)$ to Algo

Optimal strategy for Algo = optimal black-box algorithm
Optimal strategy for black-box = "most difficult" $f \in F$
Optimal number of rounds in the game = BBC($F$)

Classic Guessing Game: Mastermind

2-player game
- CodeMaker hides a $\mathbf{4}$-digit $\mathbf{6}$-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")
2-Color Mastermind = BBC(OneMax)

- OneMax test function: \( f: \{0,1\}^n \rightarrow \{0,\ldots,n\}; x \mapsto \text{"number of ones in } x\text{"} 
- easy to find the unique global optimum \((1,\ldots,1)\).
- RLS, \((1+1)\) EA, ... do this in \(\Theta(n \log n)\) time.

- (Generalized) OneMax function, OneMax problem:
  - For each \( z \in \{0,1\}^n \), let \( f_z: \{0,1\}^n \rightarrow \{0,\ldots,n\}; x \mapsto \text{"number of bits in which } x\text{ and } z\text{ agree"} \)
  - All \( f_z \) have isomorphic fitness landscapes
  - OneMax problem: \( F := \{f_z | z \in \{0,1\}^n\} \), the set of all OneMax functions

Observation: Mastermind with the two "colors" 0 and 1 corresponds to the black-box complexity \(\text{BBC}(F)\)

Mastermind: 3 (?) Results

- \(\Theta(n / \log n)\) guesses sufficient&necessary for \( k = 2 \) (BBC of OneMax)

- \(\Theta(n \log k / \log n)\) for \( k \leq n^{1-\epsilon} \)

- \(\Theta(n / \log n)\) for \( k = 2 \)

Proof: Random Guessing

- CodeBreaker’s strategy:
  - Guess \(\Theta(n / \log n)\) random codes.
  - Look at all answers.
  - With high probability, no secret code other than the true one leads to these answers [elementary, straight-forward computation]

Comments:

- \textit{Erdős probabilistic method} at its best.
- Best possible (apart from constant factors hidden in \(\Theta(\ldots)\))
- Note: Non-adaptive strategy – questions do not depend on previous questions and answers.

A General Lower Bound

- [DJW'06, in the language of games] Consider a guessing game such that
  - there are \( s \) different secrets
  - each query has at most \( k \geq 2 \) different answers.
  
  Then the expected number \( Q \) of queries necessary to find the secret is at least \( (\log_2(s) / \log_2(k)) - 1 = \log_2(s) - 1 \).

- Information theoretic view: To encode the secret in binary, you need \(\log_2(s)\) bits. Each answer can be encoded in \(\log_2(k)\) bits. If \( Q \) rounds suffice, \( Q \log_2(k) \) bits could encode the secret. \(\dagger\)

- Game-theoretic view: In the game tree, each node has at most \( k \) children.
  
  Hence at height \( Q \), there are at most \( k^Q \) nodes. If \( s \) is bigger, then at some nodes, more secrets are possible. \(\dagger\)

\(\dagger\) Argument correct for deterministic strategies. For randomized ones, in addition, Yao’s minimax principle is needed.
Back to 2-Color Mastermind…

- Lower bound: \((1 + o(1)) \frac{n}{\log_2(n)}\)
  - Argument: \(2^n\) possible secrets, \(n+1\) possible answers
  - General lower bound: \(\log_2(2^n) / \log_2(n+1) = (1+o(1))n / \log_2(n)\)
  - Information theoretic view: “learn at most \(\log_2(n)\) bits per question”

- Upper bound computed precisely: \((2 + o(1)) \frac{n}{\log_2(n)}\)
  - Weaker by a factor of 2
  - Reason (informal): Typically, a random question yields an answer between \(n/2 - \Theta(\sqrt{n})\) and \(n/2 + \Theta(\sqrt{n})\).
    - “Learn \(\log_2(\Theta(\sqrt{n})) = (1/2) \log_2(n)\) bits per question”

- Big open problem (already mentioned in the Erdős-Rényi paper): What is the correct bound? Can you ask better questions?

Clever Guessing: Mastermind for \(k = n\)

- Random guessing (Chvátal): \(\Theta(n \log(n))\) needed and sufficient.
  - Informal justification:
    - The expected answer to a random question is 1.
    - “Learn only a constant number of bits per question”
    - Information theory: \(\log(n^n) / \log(constant) = n \log(n)\) questions

- Can we ask better questions?
  - Info-theory argument: We need to “learn more bits per question”
  - Problem: For the first question, the expected answer is 1, no matter what we ask (⇒ learn constant number of bits)
  - If something works, it must be adaptive: Current question uses previous answers!

Clever Guessing: First Step

- Story-line so far: Adaptively ask clever questions!
  - Difficulty: How to use previous answers?
  - One idea (inspired by Goodrich (IPL 2009)):
    - If you get the answer “0”, then for each position you know one color that does not appear there
    - Basically reduces the number of colors by one
    - Future questions: only use possible colors
    - Good news: the answer “0” is not too rare
      - for \(k = n\) colors, the probability that a random guess gets a “0”-answer, is \((1 - (1/n))^n = 1/e = 0.37\)

Clever Guessing: Reduce the Colors

- Story-line: Adaptively ask clever questions!
  - Plan: Get a “0”-answer and get rid of one color per position.
  - Lemma: For \(k\) colors and \(n\) positions, the probability that a random guess is answered “0”, is \((1 - (1/k))^n \geq 4^{-k}\).
  - Rough estimate: Reducing the number of colors from \(n\) to \(8n / \loglog(n)\) takes time \(n \frac{4^{(8n / \loglog(n))}}{4^{(n \log(n))}} = n \log(n)^2\).
  - With only \(8n / \loglog(n)\) colors possible at each position, a random guess has an expected answer of \(\log(\loglog(n))/8\)
    - “Learn \(\Theta(\logloglog(n))\) bits” [can be made precise]
  - “Theorem”: \(O(n \log(n) / \logloglog(n))\) questions suffice!
Clever Guessing: Reduce the Colors (2)

- Story-line: Adaptively ask clever questions by reducing the number of colors (by getting a "0"-answer)
  - gains so far: a \( \log \log(n) \) factor 😊

- Reducing the number of colors from \( k \) to \( k-1 \) per position:
  - so far: get a "0"-answer after at most \( 4^{1/k} \) random guesses
  - Example: \( k = n/100 \)
    - Random guess has an expected answer of 100.
    - Time to wait for a "0" is \( (1+o(1)) \cdot e^{100} \)
    - Waiting for something quite rare 😞
  - Better: Partition the \( n \) positions into blocks of size \( n/100 \) and ask randomly in each block (fill up the rest with dummy colors)
    - expected contribution per block: 1
    - waiting time for a "0" in a block: constant

Clever Guessing: Reduce the Colors (3)

- Story-line: Adaptively ask clever questions by reducing the number of colors.
  - So far: Ask randomly and hope for a "0"

- Improved reducing the number of colors from \( k \) to \( k-1 \):
  - Partition the \( n \) positions into \( n/k \) blocks of roughly equal size.
  - For each block:
    - Ask random colors in the block, put a dummy color in the rest
    - expected waiting time for a "0": at most 4
  - Total expected waiting time: \( 4 \cdot n/k \) [previously: \( 4^{1/k} \) 😊😊]

- Total time to reduce the number of colors from \( k \) to \( k/2 \):
  - at most \( (k/2) \cdot 4 \cdot n/(k/2) = 4n \)

Clever Guessing: Reduce the Colors (4)

- Story-line: Adaptively ask clever questions.
  - Clever color reducing: From \( k \) to \( k/2 \) colors in \( 4n \) queries

- Goodrich 2009: \( \log(n) \) times halving the colors finds the secret code in \( O(n \log n) \) questions [apart from constants, the same bound as Chvátal]

- We [D., Spöhel, Thomas, Winzen]:
  - Do the halving trick \( \sqrt{\log n} \) times
    - \( n / 2^{\log_2 n} \) colors possible at each position
  - Then do random guesses (using only possible colors)
    - expected answer: \( 2 \cdot \log_2 n \)
      - "learn \( \log(2^{\log_2 n}) = \sqrt{\log n} \) bits per question"
  - Theorem: Solve Mastermind with \( k=2n \) colors in \( O(n \sqrt{\log n}) \) questions 😊

Intermediate Summary: Methods

- Information theoretic argument:
  - If for each query only \( k \) different answers exist and if \( F \) contains \( s \) functions with distinct unique optima, then the black-box complexity of \( F \) is at least \( (\log_2(s) / \log_2(k)) - 1 \).

- Random guessing:
  - Often, a small number of random guesses together with the answers received uniquely determine the solution.
  - "Information theoretic hand-waiving": If a random query typically sees \( k \) answers each with probability at least \( O(1/k) \), then around \( \log_2(s)/ \log_2(k) \) question might suffice.

- Clever guessing: To get a better bound, you have to ask questions that reveal more information (example: reducing the colors in MasterMind).
A Second Example of “Clever Guessing”

- Original problem: Memory-restricted BBC of OneMax
- Memory-restriction: From one iteration to the next, the BB-algorithm may only store $k$ search points together with their fitness.
- Conjecture [DJW’06]: For $k = 1$, the BBC becomes the $\Theta(n \log n)$ we know from the (1+1) EA.
- Transfer to guessing games [easy to see]:
  - This BBC problem is equivalent to Mastermind with two rows only.
- Theorem [DW’12]: You can win 2-row Mastermind with $O(n / \log n)$ queries.
  - Details: next few slides.
- Corollary: The memory-1 restricted BBC of OneMax is $\Theta(n / \log n)$.

Fewer Rows: Proof Ideas

- Original Mastermind: Guess $\Theta(n / \log n)$ random codes. Store all guesses and answers on the board. Think.
  - Needs $\Theta(n / \log n)$ rows.
- 3 ingredients of our proof:
  - Find parts of the code: Determine $\Theta(n^t)$ code letters with $\Theta(n^t / \log n)$ relatively random guesses ($\epsilon$ constant)  
    - Do this $n^{1-\epsilon}$ times: find the code with $\Theta(n^t / \log n)$ rows.
    - Determine such a part with constant number of rows  
    - Do this $n^{1-\epsilon}$ times: find the code with $\Theta(1)$ rows.
    - Do everything in two rows

Proof Idea (1): Find Parts of the Code

- Lemma:
  - Let $B \subseteq [n]$, $|B| = n^\epsilon$, “part” 
  - Let $G_1, G_2, \ldots$ be $\Theta(n^t / \log n)$ guesses such that
    - $G_i$ is random in positions in $B$ 
    - All $G_i$ are equal in positions in $[n] \setminus B$ 
  - Then with high probability these guesses and answers determine the secret code in $B$.
- Argument:
  - Basically, we play the game in $B$ (and use the previous proof) 
  - Only difficulty: The answers we get “are not for $B$ only”, but for the whole guess 
    - Same deviation for all guesses 
    - Some maths: Not a problem, guesses also determine deviation
Proof Idea (2): Same with $O(1)$ Rows

- Plan: Simulate the previous slide in $O(1)$ rows
- Example: Find the first $L = \Theta(n^\varepsilon / \log n)$ code letters
  - $B_1 := \ell$ random letters.
  - Guess $B_1 1...1$ in row 1 and learn answer $A_1$.
  - Guess $B_1 A_1 1...1$ in row 2 and ignore answer.
  - $B_2 := \ell$ random letters.
  - Guess $B_2 1...1$ in row 1 and learn answer $A_2$.
  - Guess $B_1 A_1 B_2 A_2 1...1$ in row 3 and ignore answer.
- General:
  - Do an honest guess as on the previous slide.
  - Use the next guess to store guess+answer+what you learned before.
- Needs 3+ rows: current guess + old storage \(\rightarrow\) new storage

$A_1$: Suitably encoded with $O(\log n)$ of letters

Proof Idea (3): Two Rows Only

- Difficulty:
  - To enter a new guess, one of the two rows must be emptied.
  - You must store and guess in the same row.
  - Problem: Storage influences CodeMaker’s answers!
  - All control information must also be stored in this one row.
    - what is the block I’m just optimizing?
    - what am I currently doing (guessing, storing, finding the unique solution, finding the last few letters in a different way...)
- Solution:
  - technical.
  - read the paper at STACS'12 or arxiv.org/abs/1110.3619.

Summary: Memory-BBC of OneMax

- Result: The complexity of Mastermind remains at $\Theta(n^\varepsilon / \log n)$ guesses even if we allow only two rows.
- Key proof argument: Clever guesses inspired by random guesses
- Open problems / future work:
  - Our proof works for any constant number of colors – what happens for larger numbers of colors?
  - constant factors: “what’s hidden in the $\Theta(\ldots)$”
    - does a memory restriction lose us a constant factor?

Finally: A New Guessing Game

- So far: BBC is strongly related to guessing games
  - In particular: $\text{BBC(OneMax)} = \text{Mastermind}$
  - Therefore: Use game theoretic arguments to solve BBC problems
- Now [next few slides]: Use BBC problems to derive a fun game 😊
  - LeadingOnes Game
LeadingOnes Test Functions

- Classic test function:
  - LeadingOnes: \( \{0,1\}^n \rightarrow \{0,\ldots,n\} \); \( x \mapsto \max\{i \in \{0,\ldots,n\} \mid x_1 = \ldots = x_i = 1\} \)
    - "how many bits counted from the left are one"
  - Unique optimum \((1,\ldots,1)\)
  - "Harder than OneMax": Each non-optimal solution has only one superior Hamming neighbor

- LeadingOnes function class \( LO_n \):
  - Let \( \sigma \) be a permutation of \( \{1,\ldots,n\} \)
  - Let \( z \in \{0,1\}^n \) ("target string")
  - \( f_{\sigma} : \{0,1\}^n \rightarrow \{0,\ldots,n\} ; \; x \mapsto \max\{i \in \{0,\ldots,n\} \mid x_{\sigma(1)} = z_{\sigma(1)}, \ldots, x_{\sigma(i)} = z_{\sigma(i)}\} \)
    - "how many bits, counted in the order of \( \sigma \), are as in \( z \)"
  - Same fitness landscape as LeadingOnes

The LeadingOnes Game

- Transfer the BBC(LO) problem into a game:
  - CodeMaker: Picks a secret code \( z \) and a secret permutation \( \sigma \)
  - Round:
    - CodeBreaker guesses a bit-string \( x \in \{0,1\}^n \)
    - CodeMaker's answer: \( f_{\sigma}(x) = "\text{how many code letters in the order of } \sigma \text{ are correct?}" \)
  - How many rounds until CodeBreaker guesses the secret code \( z \)?
  - Practice: Fun to play with \( n=5 \) or \( n=6 \) [and that's the message of this slide]
  - Theory: next few slides, fun as well, but doesn't help you play the actual game

Black-Box Complexity of LeadingOnes

- Reminder: \( LO_n \) consists of all functions
  - \( f_{\sigma} : \{0,1\}^n \rightarrow \{0,\ldots,n\} ; \; x \mapsto \max\{i \in \{0,\ldots,n\} \mid x_{\sigma(1)} = z_{\sigma(1)}, \ldots, x_{\sigma(i)} = z_{\sigma(i)}\} \)
  - Black-box complexity of \( LO_n \), lower bound
    - \( \Omega(n) \), because you need \( \Omega(n) \) fitness evaluations even if \( \sigma = \text{id} \)
  - Black-box complexity of \( LO_n \), upper bounds
    - \( O(n^2) \), run-time of RLS, (1+1) EA, ...
    - \( O(n \log(n)) \): determine "the next bit" with \( \log(n) \) queries by simulating binary search (more details next slide)
      - Would be a natural lower bound:
        - "next bit"-position is a number in \( \{1,\ldots,n\} \), coding length \( \log(n) \)
        - A typical query teaches you a constant amount of information
    - DW (EA'11): \( O(n \log(n) / \log\log(n)) \) is enough...

The BinarySearch Trick

- Assume that you have a solution \( x \) with \( f_{\sigma}(x) = k \) and you know which \( k \) bit-positions are responsible for this. Denote by \( I \) the remaining bit-positions.
  - While \( |I| > 1 \) do
    - Choose \( J \subseteq I \) with \( |J| = |I|/2 \)
    - Obtain \( y \) from \( x \) by flipping the bits in \( J \)
    - If \( f_{\sigma}(y) > f_{\sigma}(x) \) then \( I := J \)
    - Else \( I := I \setminus J \)
  - Determines "the next bit" with at most \( \log(n) \) fitness queries
  - \( n \log(n) \) queries suffice to optimize \( LO_n \)
  - How can we do better?
### Proving $O(n \log(n) / \log\log(n))$: Outline

- Assume that you have a solution $x$ with $f_\sigma(x) = k$ and you know which $k$ bit-positions are responsible for this. Denote by $I$ the remaining bit-positions. Let $L := \log(n)^{1/2}$.

- **Step 1**: Use $L^2 = \log(n)$ iterations to find a $y$ with $f_\sigma(y) = k + L$.
  - Flip the bits in $I$ with probability $1/L$, accept if improvement.
  - Note: We don’t learn which $L$ bit-positions lead to the improvement!!!

- **Step 2**: Use $\log(n)^{3/2} / \log\log(n)$ queries to determine the $L$ bit-positions.
  - In $y$, flip the $I$-bits with probability $1/L$. Do so $\log(n)^{3/2} / \log\log(n)$ times.
  - Look at all outcomes with fitness $k+j$ and find out bit number $k+j+1$.
  - With high probability, the $\log(n)^{3/2} / \log\log(n)$ samples suffice to learn all $L$ bit-positions.

- Step 1+2: $\log(n)^{3/2} / \log\log(n)$ fitness evaluations to gain $\log(n)^{1/2}$ bits...

### Final Summary 😊

- Black-box complexity: Expected number of fitness evaluations the best black-box algorithm needs to query the optimum of the hardest instance.
  - $\min_{A} \max_{I} I(A,I)$
  - Note: lower bound on the performance of any EA, ACO, …

- Strongly related to guessing games
  - BBC(OneMax) = Mastermind
  - BBC(LeadingOnes) = what you should play in the next tutorial 😊

- Techniques:
  - Information theory: BBC $\geq \log(|\text{SearchSpace}|) / \log(|\text{fitness_values}|)$
  - Random guesses: Often $\leq \log(|\text{SearchSpace}|) / \log(|\text{typical_answers}|)$
  - Clever guesses: Be creative!