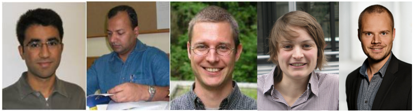
# The Query Complexity of a Mastermind Variant (DAM 2019)

P. Afshani, M. Agrawal,
B. Doerr, C. Doerr,
K. G. Larsen,
K. Mehlhorn



max planck institut
informatik

SIC Saarland Informatics Campus

April 25, 2019

# Mastermind


Secret: $z \in [k]^n$

Query: $x \in [k]^n$ Score: $|\{ i; z_i = x_i \}|$

k = 6, n = 4: Knuth (77): 5 queries

Erdós/Renyi (63), Chvatal (83), Doerr/Winzen (13)

$k \leq n^{1-\epsilon}$: $\Theta(n)$ queries

$k \geq n$: $\Omega(n)$, $O(n \log\log n)$ queries

Deciding whether a history is consistent is NP-complete (Stuckman/Zhang, Goodrich, Viglietta)

Commercial board game since 70's

Open Problem: Close the gap for $k \geq n$.

# Mastermind


Secret: $z \in [k]^n$

Query: $x \in [k]^n$ Score: $|\{ i; z_i = x_i \}|$

k = 6, n = 4: Knuth (77): 5 queries

Erdós/Renyi (63), Chvatal (83), Doerr/Winzen (13)

$k \leq n^{1-\epsilon}$: $\Theta(n)$ queries

$k \geq n$: $\Omega(n)$, $O(n \log\log n)$ queries

Deciding whether a history is consistent is NP-complete (Stuckman/Zhang, Goodrich, Viglietta)

Commercial board game since 70's

Open Problem: Close the gap for $k \geq n$.

# Mastermind


Secret: $z \in [k]^n$

Query: $x \in [k]^n$ Score: $|\{\, i;\, z_i = x_i \,\}|$

k = 6, n = 4: Knuth (77): 5 queries

Erdós/Renyi (63), Chvatal (83), Doerr/Winzen (13)

$k \leq n^{1-\epsilon}$: $\Theta(n)$ queries

$k \geq n$: $\Omega(n)$, $O(n \log\log n)$ queries

Deciding whether a history is consistent is NP-complete (Stuckman/Zhang, Goodrich, Viglietta)

Commercial board game since 70's

Open Problem: Close the gap for $k \geq n$.

# Mastermind


Secret: $z \in [k]^n$

Query: $x \in [k]^n$ Score: $|\{ i; z_i = x_i \}|$

k = 6, n = 4: Knuth (77): 5 queries

Erdós/Renyi (63), Chvatal (83), Doerr/Winzen (13)

$k \leq n^{1-\epsilon}$: $\Theta(n)$ queries

$k \geq n$: $\Omega(n)$, $O(n \log\log n)$ queries

Deciding whether a history is consistent is NP-complete (Stuckman/Zhang, Goodrich, Viglietta)

Commercial board game since 70's

Open Problem: Close the gap for $k \geq n$.

# Mastermind


Secret: $z \in [k]^n$

Query: $x \in [k]^n$ Score: $|\{ i; z_i = x_i \}|$

k = 6, n = 4: Knuth (77): 5 queries

Erdós/Renyi (63), Chvatal (83), Doerr/Winzen (13)

$k \leq n^{1-\epsilon}$: $\Theta(n)$ queries

$k \geq n$: $\Omega(n)$, $O(n \log\log n)$ queries

Deciding whether a history is consistent is NP-complete (Stuckman/Zhang, Goodrich, Viglietta)

Commercial board game since 70's

Open Problem: Close the gap for $k \geq n$.

# Mastermind


Secret: $z \in [k]^n$

Query: $x \in [k]^n$  Score: $|\{ i; z_i = x_i \}|$

k = 6, n = 4: Knuth (77): 5 queries

Erdós/Renyi (63), Chvatal (83), Doerr/Winzen (13)

$k \leq n^{1-\epsilon}$: $\Theta(n)$ queries
$k \geq n$: $\Omega(n)$, $O(n \log\log n)$ queries

Deciding whether a history is consistent is NP-complete (Stuckman/Zhang, Goodrich, Viglietta)

Commercial board game since 70's

Open Problem: Close the gap for $k \geq n$.

# The Game Board for the New Game

Codemaker on the right, Codebreaker on the left. Codemaker chooses a binary string of length $n$ ($= 4$) and enters it into the square on the right (one bit per row and column).

| | | | | | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | | | 0 | | | |
| 1 | 1 | 0 | | | | | 0 | |
| 0 | 1 | 1 | | | | 1 | | |
| 0 | 0 | 1 | | | | | | 0 |
| 1 | 0 | 3 | | | | | | |

Positioning of the bitstring encodes a permutation.

Score = # of leading bits in which query and secret agree.

# A Mastermind Variant

- Find an unknown bitstring $z \in \{0,1\}^n$ and an unknown permutation $\pi$ by asking queries $x \in \{0,1\}^n$.
- Answer to query $x \in \{0,1\}^n$ is

$$f(x) := \max\{\, k;\ x_{\pi(i)} = z_{\pi(i)} \text{ for } i \le k \,\}.$$

For example ($\pi(1) = 2$, $\pi(2) = 3$, $\ldots$)

$$\begin{array}{rcccc} (z,\pi) = & 1_4 & 0_1 & 1_2 & 0_3 \\ x = & 1 & 0 & 1 & 1 \end{array}$$ has answer 2.

- How many questions are needed to unveil the secret?
- How did we get interested? Model problem in evolutionary computation. Carola and Benjamin found a randomized $O(n \log n / \log\log n)$ algorithm.

# A Mastermind Variant

- Find an unknown bitstring $z \in \{0,1\}^n$ and an unknown permutation $\pi$ by asking queries $x \in \{0,1\}^n$.
- Answer to query $x \in \{0,1\}^n$ is

$$f(x) := \max \{ k; x_{\pi(i)} = z_{\pi(i)} \text{ for } i \le k \}.$$

For example ($\pi(1) = 2$, $\pi(2) = 3, \ldots$)

$$\begin{array}{rcccc} (z,\pi) = & 1_4 & 0_1 & 1_2 & 0_3 \\ x = & 1 & 0 & 1 & 1 \end{array} \quad \text{has answer 2.}$$

- How many questions are needed to unveil the secret?
- How did we get interested? Model problem in evolutionary computation. Carola and Benjamin found a randomized $O(n \log n / \log \log n)$ algorithm.

# Results

- Information theoretic lower bound: $\Omega(n)$ queries.
- Deterministic algs: $\Theta(n \log n)$ queries suffice and are needed in the worst case, $O(n \log\log n)$ queries suffice on average.
- Randomized algs: $\Theta(n \log\log n)$ queries suffice in expectation and are needed; also high probability.
- The randomized lower bound uses a sophisticated potential function argument, the randomized upper bound is by a non-trivial algorithm.
- A characterization of what is known after a sequence of queries and answers: counting the number of secrets consistent with a history is in P.

# Results

- Information theoretic lower bound: $\Omega(n)$ queries.
- Deterministic algs: $\Theta(n \log n)$ queries suffice and are needed in the worst case, $O(n \log\log n)$ queries suffice on average.
- Randomized algs: $\Theta(n \log\log n)$ queries suffice in expectation and are needed; also high probability.
- The randomized lower bound uses a sophisticated potential function argument, the randomized upper bound is by a non-trivial algorithm.
- A characterization of what is known after a sequence of queries and answers: counting the number of secrets consistent with a history is in P.

# Results

- Information theoretic lower bound: $\Omega(n)$ queries.
- Deterministic algs: $\Theta(n \log n)$ queries suffice and are needed in the worst case, $O(n \log\log n)$ queries suffice on average.
- Randomized algs: $\Theta(n \log\log n)$ queries suffice in expectation and are needed; also high probability.
- The randomized lower bound uses a sophisticated potential function argument, the randomized upper bound is by a non-trivial algorithm.
- A characterization of what is known after a sequence of queries and answers: counting the number of secrets consistent with a history is in P.

# Results

- Information theoretic lower bound: $\Omega(n)$ queries.
- Deterministic algs: $\Theta(n \log n)$ queries suffice and are needed in the worst case, $O(n \log\log n)$ queries suffice on average.
- Randomized algs: $\Theta(n \log\log n)$ queries suffice in expectation and are needed; also high probability.
- The randomized lower bound uses a sophisticated potential function argument, the randomized upper bound is by a non-trivial algorithm.
- A characterization of what is known after a sequence of queries and answers: counting the number of secrets consistent with a history is in P.

# Results

- Information theoretic lower bound: $\Omega(n)$ queries.
- Deterministic algs: $\Theta(n \log n)$ queries suffice and are needed in the worst case, $O(n \log\log n)$ queries suffice on average.
- Randomized algs: $\Theta(n \log\log n)$ queries suffice in expectation and are needed; also high probability.
- The randomized lower bound uses a sophisticated potential function argument, the randomized upper bound is by a non-trivial algorithm.
- A characterization of what is known after a sequence of queries and answers: counting the number of secrets consistent with a history is in P.

# Deterministic Upper Bound

- Query $0^n$
- Answer tells whether $z_{\pi(1)}$ is 0 or 1, say 0.
- Then use binary search to determine $\pi(1)$:
  - Ask $0^{n/2}1^{n/2}$, answer reduces candidate set to first or second half.
- $\#$ of queries: $1 + \log n$ per position, $n + n \log n$ overall.
- Randomized binary search: ask a random string $x$ with exactly $n/2$ 0's.
- Then candidate set (= possible values for $\pi(1)$) contains correct position and is random otherwise (useful observation for later).

# Deterministic Upper Bound

- Query $0^n$
- Answer tells whether $z_{\pi(1)}$ is 0 or 1, say 0.
- Then use binary search to determine $\pi(1)$:
  - Ask $0^{n/2}1^{n/2}$, answer reduces candidate set to first or second half.
- $\#$ of queries: $1 + \log n$ per position, $n + n \log n$ overall.
- Randomized binary search: ask a random string $x$ with exactly $n/2$ 0's.
- Then candidate set (= possible values for $\pi(1)$) contains correct position and is random otherwise (useful observation for later).

## Deterministic Upper Bound

- Query $0^n$
- Answer tells whether $z_{\pi(1)}$ is 0 or 1, say 0.
- Then use binary search to determine $\pi(1)$:
  - Ask $0^{n/2}1^{n/2}$, answer reduces candidate set to first or second half.

- $\#$ of queries: $1 + \log n$ per position, $n + n \log n$ overall.

- Randomized binary search: ask a random string $x$ with exactly $n/2$ 0's.
- Then candidate set (= possible values for $\pi(1)$) contains correct position and is random otherwise (useful observation for later).

# Deterministic Lower Bound (Adversary Argument)

**Theorem**

*For every deterministic algorithm there is an input on which the alg needs $\frac{1}{2} n \log n$ queries.*

**Adversary Argument**

Adversary keeps track of which secrets are still consistent with his answers.
Answers queries so that options are reduced as little as possible.

# Deterministic Lower Bound (Adversary Argument)

- Adversary works in phases of $\log n$ queries. In each phase, the adversary commits to the next two bits and positions.
- Let $x^*$ be the first query. *Adversary* gives it a score of 1 and hence commits to "$z_{\pi(1)} = x^*_{\pi(1)}$" and "$z_{\pi(2)} = 1 - x^*_{\pi(2)}$".
- $R_1 =$ possible values for $\pi(1)$,
  $R_2 =$ possible values for $\pi(2)$; $R_1 = R_2 = [n]$ initially
- Let $x$ be the next query; let $I = \{ i;\ x_i = x^*_i \}$
  - If adversary gives a score of 0: $R_1 = R_1 \setminus I$ and $R_2 = R_2$
  - If adversary gives a score of 1: $R_1 = R_1 \cap I$ and $R_2 = R_2 \cap I$.
- Adversary chooses the answer that at most halves $R_1$ and hence can stick to scores 0 and 1 for the next $\log n$ queries.
- After $\log n$ queries, adv. commits to $\pi(1)$ and $\pi(2)$ and the bits in these locations.
  Codebreaker has not learned anything about $R_3, R_4, \ldots$

# Deterministic Lower Bound (Adversary Argument)

- Adversary works in phases of $\log n$ queries. In each phase, the adversary commits to the next two bits and positions.
- Let $x^*$ be the first query. *Adversary* gives it a score of 1 and hence commits to "$z_{\pi(1)} = x^*_{\pi(1)}$" and "$z_{\pi(2)} = 1 - x^*_{\pi(2)}$".
- $R_1 =$ possible values for $\pi(1)$,
  $R_2 =$ possible values for $\pi(2)$; $R_1 = R_2 = [n]$ initially
- Let $x$ be the next query; let $I = \{\ i;\ x_i = x^*_i\ \}$
  - If adversary gives a score of 0: $R_1 = R_1 \setminus I$ and $R_2 = R_2$
  - If adversary gives a score of 1: $R_1 = R_1 \cap I$ and $R_2 = R_2 \cap I$.
- Adversary chooses the answer that at most halves $R_1$ and hence can stick to scores 0 and 1 for the next $\log n$ queries.
- After $\log n$ queries, adv. commits to $\pi(1)$ and $\pi(2)$ and the bits in these locations.
  Codebreaker has not learned anything about $R_3, R_4, \ldots$

## Deterministic Lower Bound (Adversary Argument)

- Adversary works in phases of $\log n$ queries. In each phase, the adversary commits to the next two bits and positions.
- Let $x^*$ be the first query. *Adversary* gives it a score of 1 and hence commits to "$z_{\pi(1)} = x^*_{\pi(1)}$" and "$z_{\pi(2)} = 1 - x^*_{\pi(2)}$".
- $R_1 =$ possible values for $\pi(1)$,
  $R_2 =$ possible values for $\pi(2)$; $R_1 = R_2 = [n]$ initially
- Let $x$ be the next query; let $I = \{ i; x_i = x^*_i \}$
  - If adversary gives a score of 0: $R_1 = R_1 \setminus I$ and $R_2 = R_2$
  - If adversary gives a score of 1: $R_1 = R_1 \cap I$ and $R_2 = R_2 \cap I$.
- Adversary chooses the answer that at most halves $R_1$ and hence can stick to scores 0 and 1 for the next $\log n$ queries.
- After $\log n$ queries, adv. commits to $\pi(1)$ and $\pi(2)$ and the bits in these locations.
  Codebreaker has not learned anything about $R_3, R_4, \ldots$

# Doerr/Doerr (2011): An $O(n \log n / \log\log n)$ Rand. Alg.

The algorithm alternates between exploration phases and consolidation phases.

In exploration phases, it increases the score by $k_0 = \sqrt{\log n}$.

In consolidation phases, it determines where these $k_0$ bits are positioned.

A phase requires $k_0 \cdot \log n / \log\log n$ queries.

$$\text{total \# of queries} = \frac{n}{k_0} \cdot \frac{k_0 \log n}{\log\log n} = \frac{n \log n}{\log\log n}.$$

# Doerr/Doerr (2011): An $O(n \log n / \log\log n)$ Rand. Alg.

Let $(z, \pi)$ be the secret.

**Alg: Part I (Explore and Increase Score)**

- Find an $x^*$ with $f(x^*) = 0$, $0^n$ or $1^n$ will do
- Repeat until $f(x^*) \geq k_0 = \sqrt{\log n}$
  - Obtain $y$ from $x^*$ by flipping bits with probability $p = 1/k_0$
  - If $f(y) > f(x^*)$ then $x^* \leftarrow y$
- Now $x^*_{\pi(i)} = z_{\pi(i)}$ for $i \leq k_0$

## Doerr/Doerr (2011): An $O(n \log n / \log\log n)$ Rand. Alg.

### Alg: Part I (Explore and Increase Score)

- Find an $x^*$ with $f(x^*) = 0$, $0^n$ or $1^n$ will do
- Repeat until $f(x^*) \geq k_0 = \sqrt{\log n}$
  - Obtain $y$ from $x^*$ by flipping bits with probability $p = 1/k_0$
  - If $f(y) > f(x^*)$ then $x^* \leftarrow y$

### Analysis

Assume $f(x^*) = k \in [0, k_0]$. Then $\text{prob}(f(y) > f(x^*)) =$

$$\text{prob}(x^*_{\pi(1)}, \ldots, x^*_{\pi(k)} \text{ are not flipped, but } x^*_{\pi(k+1)} \text{ is}) = (1-p)^k \cdot p$$

$$= (1 - \frac{1}{k_0})^k \cdot p \geq p/e$$

We reach a score of $k_0 = \sqrt{\log n}$ in $k_0 \cdot \frac{1}{p} = k_0^2 = \log n$ steps.

# Doerr/Doerr (2011): An $O(n \log n / \log\log n)$ Rand. Alg.

## Alg: Part I (Explore and Increase Score)

- Find an $x^*$ with $f(x^*) = 0$, $0^n$ or $1^n$ will do
- Repeat until $f(x^*) \geq k_0 = \sqrt{\log n}$
  - Obtain $y$ from $x^*$ by flipping bits with probability $p = 1/k_0$
  - If $f(y) > f(x^*)$ then $x^* \leftarrow y$

## Alg: Part II (Consolidate)

- $R_1 = R_2 = \ldots = R_{k_0} = [1, n]$ recall $x^*_{\pi(i)} = z_{\pi(i)}$ for $i \leq k_0$
- Repeat $c \cdot k_0 \cdot \log n / \log\log n$ times
  - Obtain $y$ from $x^*$ by flipping bits with probability $p = 1/k_0$. Let $F$ be the bits flipped.
  - If $f(y) = \ell < f(x^*)$ then $R_{\ell+1} \leftarrow R_{\ell+1} \cap F$ also, $R_j \leftarrow R_j \setminus F$ for $j \leq \ell$
- Claim: $R_1$ to $R_{k_0}$ are now singletons with high probability, i.e., $R_i = \{ \pi(i) \}$. If some $R_i$ is not a singleton, switch to deterministic alg.

# Doerr/Doerr (2011): An $O(n \log n / \log\log n)$ Rand. Alg.

## Alg: Part II (Consolidate)

- $R_1 = R_2 = \ldots = R_{k_0} = [1, n]$ recall $x^*_{\pi(i)} = z_{\pi(i)}$ for $i \leq k_0$
- Repeat $c \cdot k_0 \cdot \log n / \log\log n$ times
  - Obtain $y$ from $x^*$ by flipping bits with probability $p = 1/k_0$. Let $F$ be the bits flipped.
  - If $f(y) = \ell < f(x^*)$ then $R_{\ell+1} \leftarrow R_{\ell+1} \cap F$ also, $R_j \leftarrow R_j \setminus F$ for $j \leq \ell$

## Analysis

For each $\ell$: $f(y) = \ell$ occurs $\Omega(\log n / \log\log n)$ times, since

prob$(f(y) = \ell) \geq p/e$.

Whenever $f(y) = \ell$: only $j$ for which $x^*_j$ is flipped stay in $R_{\ell+1}$ and hence $|R_{\ell+1}| \to |R_{\ell+1}|/k_0$

After $\Omega(\log n / \log\log n)$ iterations, $|R_{\ell+1}| = 1$.

## The Complete Algorithm

Algorithm alternates between explore and consolidate.

Explore: Increases $f(x^*)$ by $k_0 = \sqrt{\log n}$ at cost $k_0\sqrt{\log n}$.

Consolidate: Determines correspondings $\pi(i)$'s at cost $k_0 \log n / \log\log n$.

Thus: $\log n / \log\log n$ queries per position.

## Main Idea for Improvement to $O(n \log\log n)$

- Intertwine the various phases.
- On average: One candidate block is reduced to the root of its size per query.
- $s \to s^{1/2}$ costs one query, $n \to n^{1/2^d}$ costs $d$ queries.
- $n^{1/2^d} = 1$ iff $\frac{1}{2^d} \log n = 1$ iff $2^d = \log n$ iff $d = \log\log n$.

# Doerr and Doerr (2011): $O(n \log n / \log\log n)$ RA

## The Complete Algorithm

Algorithm alternates between explore and consolidate.

Explore: Increases $f(x^*)$ by $k_0 = \sqrt{\log n}$ at cost $k_0\sqrt{\log n}$.

Consolidate: Determines correspondings $\pi(i)$'s at cost $k_0 \log n / \log\log n$.

Thus: $\log n / \log\log n$ queries per position.

## Main Idea for Improvement to $O(n \log\log n)$

- Intertwine the various phases.
- On average: One candidate block is reduced to the root of its size per query.
- $s \to s^{1/2}$ costs one query, $n \to n^{1/2^d}$ costs $d$ queries.
- $n^{1/2^d} = 1$ iff $\frac{1}{2^d} \log n = 1$ iff $2^d = \log n$ iff $d = \log\log n$.

# Randomized Algorithms, a Closer Look

Deterministic algorithms can be viewed as trees with branching factor $n+1$. Each node has as associated query and alg. branches on the score.

Randomized algorithms can also be considered as trees. Each node has as an associated probability distribution over the possible $2^n$ queries. Query is chosen according to this distribution, e.g., by choosing a random number in $[0, 1]$.

We may choose all random numbers upfront:

RA + particular choice of random numbers = deterministic alg.

So RA is a probability distribution over deterministic algorithms.

# Randomized Algorithms, a Closer Look

Deterministic algorithms can be viewed as trees with branching factor $n+1$. Each node has as associated query and alg. branches on the score.

Randomized algorithms can also be considered as trees. Each node has as an associated probability distribution over the possible $2^n$ queries. Query is chosen according to this distribution, e.g., by choosing a random number in $[0,1]$.

We may choose all random numbers upfront:

RA + particular choice of random numbers = deterministic alg.

So RA is a probability distribution over deterministic algorithms.

# Randomized Algorithms, a Closer Look

Deterministic algorithms can be viewed as trees with branching factor $n + 1$. Each node has as associated query and alg. branches on the score.

Randomized algorithms can also be considered as trees. Each node has as an associated probability distribution over the possible $2^n$ queries. Query is chosen according to this distribution, e.g., by choosing a random number in $[0, 1]$.

We may choose all random numbers upfront:

RA + particular choice of random numbers $=$ deterministic alg.

So RA is a probability distribution over deterministic algorithms.

# Randomized Lower Bound: Yao's Principle

Define a probability distribution $D$ on inputs and show: Every deterministic alg has average query complexity $\Omega(n \log\log n)$ for this distribution.

A randomized algorithm is a probability distribution on deterministic algs, say RA is $A_i$ with probability $p_i$. Then $\mathrm{Cost}_{RA}(x) = \sum_i p_i \cdot \mathrm{Cost}_i(x)$ and hence

$$
\begin{aligned}
\max_x \mathrm{Cost}_{RA}(x) &\geq \mathrm{E}_{x \sim D}[\mathrm{Cost}_{RA}(x)] \\
&= \mathrm{E}_{x \sim D}[\sum_i p_i \cdot Cost_i(x)] \\
&= \sum_i p_i \cdot \mathrm{E}_{x \sim D}[Cost_i(x)] \\
&\geq \sum_i p_i \cdot \Omega(n \log\log n) \\
&= \Omega(n \log\log n).
\end{aligned}
$$

# Randomized Lower Bound: Warm-Up

- Given $A[1..n]$ and a value $x$, determine $i$ such that $A[i] = x$.
- Randomization, powerful queries: Is $i \in Q$, where $Q \subseteq [n]$.
- Input distribution: $\text{prob}(x = A[i]) = 1/n$ for all $i \in [n]$.
- For a node $v$: Let $C_v$ be the candidates at $v$ and $\Phi(v) = \log n/|C_v|$.
- $\Phi(\text{root}) = 0$, $\Phi(\text{any leaf}) = \log n$, expected gain in potential per query at most one $\Rightarrow$ expected depth of tree is $\log n$.
- Expected gain: Let $w_0$ and $w_1$ be the children of $v$ and let $Q_v$ be the query at $v$. Then $C_{w_0} = C_v \cap Q_v$ and $C_{w_1} = C_v \setminus Q_v$ and $\varepsilon_0 = \text{prob}(v \to w_0) = \frac{|C_{w_0}|}{|C_v|}$. Therefore

$$\begin{aligned}
&\varepsilon_0 \Phi(w_0) + (1-\varepsilon_0)\Phi(w_1) - \Phi(v) \\
&\qquad = \varepsilon_0 \log \frac{n}{\varepsilon_0 |C_v|} + (1-\varepsilon_0) \log \frac{n}{(1-\varepsilon_0)|C_v|} - \log \frac{n}{|C_v|} \\
&\qquad = \varepsilon_0 \log \frac{1}{\varepsilon_0} + (1-\varepsilon_0) \log \frac{1}{1-\varepsilon_1} \leq 1.
\end{aligned}$$

# Randomized Lower Bound: A Potential Function Argument

Distribution on secrets $(\pi, z)$: $\pi$ is a random permutation and $z$ is 0 in even positions according to $\pi$, 1 in odd.

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| | 0 | | |
| | | 1 | |
| 1 | | | |
| | | | 0 |

- For any node $v$ of the decision tree:
  $$R_i^v = \{\, \text{values still possible for } \pi(i) \,\}.$$
- $R_i^{\text{root}} = [n]$ and $R_i^{\text{leaf}} = 1$ for all $i$.
- Potential function: $\Phi(v) = \ldots + \sum_j \log\log \frac{2n}{|R_j^v|}$; inspired by the upper bound.
- Potential is zero for the root, $n \log\log n$ for leaves, and expected increase of potential per query is constant. Therefore, average depth of decision tree is $\Omega(n \log\log n)$.

$$\Omega(n \log\log n) = \sum_{\text{leaf } \ell} p(\ell)\Phi(\ell) - \Phi(\text{root})$$

$$= \sum_{\text{leaf } \ell} p(\ell)\Phi(\ell) - \Phi(\text{root}) \pm \sum_{\text{non-leaf, non-root v}} p(v)\Phi(v)$$

$$= \sum_{\text{non-leaf } v} p(v) \left( \sum_{\text{child } w \text{ of } v} \text{prob}(v \to w)\Phi(w) - \Phi(v) \right)$$

# Randomized Lower Bound: A Potential Function Argument

Distribution on secrets $(\pi, z)$: $\pi$ is a random permutation and $z$ is 0 in even positions according to $\pi$, 1 in odd.

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| | 0 | | |
| | | 1 | |
| 1 | | | |
| | | | 0 |

- For any node $v$ of the decision tree:
  $$R_i^v = \{\text{ values still possible for } \pi(i)\ \}.$$
- $R_i^{\text{root}} = [n]$ and $R_i^{\text{leaf}} = 1$ for all $i$.
- Potential function: $\Phi(v) = \ldots + \sum_j \log\log \frac{2n}{|R_j^v|}$;
  inspired by the upper bound.
- Potential is zero for the root, $n \log\log n$ for leaves, and expected increase of potential per query is constant. Therefore, average depth of decision tree is $\Omega(n \log\log n)$.

$$\Omega(n \log\log n) = \sum_{\text{leaf } \ell} p(\ell)\Phi(\ell) - \Phi(\text{root})$$

$$= \sum_{\text{leaf } \ell} p(\ell)\Phi(\ell) - \Phi(\text{root}) \pm \sum_{\text{non-leaf, non-root v}} p(v)\Phi(v)$$

$$= \sum_{\text{non-leaf } v} p(v) \left( \sum_{\text{child } w \text{ of } v} \text{prob}(v \to w)\Phi(w) - \Phi(v) \right)$$

# Randomized Lower Bound: A Potential Function Argument

Distribution on secrets $(\pi, z)$: $\pi$ is a random permutation and $z$ is 0 in even positions according to $\pi$, 1 in odd.

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| | 0 | | |
| | | 1 | |
| 1 | | | |
| | | | 0 |

- For any node $v$ of the decision tree:
  $$R_i^v = \{\text{ values still possible for } \pi(i)\,\}.$$
- $R_i^{\text{root}} = [n]$ and $R_i^{\text{leaf}} = 1$ for all $i$.
- Potential function: $\Phi(v) = \ldots + \sum_j \log\log \frac{2n}{|R_j^v|}$;
  inspired by the upper bound.
- Potential is zero for the root, $n \log\log n$ for leaves, and expected increase of potential per query is constant. Therefore, average depth of decision tree is $\Omega(n \log\log n)$.

$$\Omega(n\log\log n) = \sum_{\text{leaf } \ell} p(\ell)\Phi(\ell) - \Phi(\text{root})$$

$$= \sum_{\text{leaf } \ell} p(\ell)\Phi(\ell) - \Phi(\text{root}) \pm \sum_{\text{non-leaf, non-root } v} p(v)\Phi(v)$$

$$= \sum_{\text{non-leaf } v} p(v) \left( \sum_{\text{child } w \text{ of } v} \text{prob}(v \to w)\Phi(w) - \Phi(v) \right)$$

# Randomized Lower Bound: A Potential Function Argument

- Potential is zero for the root, $n \log\log n$ for leaves, and expected increase of potential per query is constant. Therefore, average depth of decision tree is $\Omega(n \log\log n)$.

$$
\begin{aligned}
\Omega(n \log\log n) &= \sum_{\text{leaf } \ell} p(\ell)\Phi(\ell) - \Phi(\text{root}) \\
&= \sum_{\text{leaf } \ell} p(\ell)\Phi(\ell) - \Phi(\text{root}) \pm \sum_{\text{non-leaf, non-root v}} p(v)\Phi(v) \\
&= \sum_{\text{non-leaf } v} p(v) \cdot \left( \sum_{\text{child } w \text{ of } v} \text{prob}(v \to w)\Phi(w) - \Phi(v) \right) \\
&\leq \sum_{\text{non-leaf } v} p(v) \cdot O(1) \\
&= O(1) \sum_{\text{leaf } \ell} p(\ell) \cdot \text{depth}(\ell).
\end{aligned}
$$

# Information Gain by a Query

- For any node $v$: $R_i^v$ = values still possible for $\pi(i)$ in $v$.
- $R_i^{\text{root}} = [n]$ for all $i$.
- It is easy to keep track of the sets $R_i^v$ and the probability of having a particular score, for example for $v =$ root we have:
  - Assume we query 0110 in the root.
  - score $= 0$, $R_1 \leftarrow \{1, 4\}$, prob() $= 1/2$.
  - score $= 1$, $R_1 \leftarrow \{2, 3\}$, $R_2 \leftarrow \{2, 3\}$, prob() $= 4/24$.
  - score $= 2$, $R_1 \leftarrow \{2, 3\}$, $R_2 \leftarrow R_3 \leftarrow \{1, 4\}$, prob() $= 4/24$.
  - score $= 3$, impossible.
  - score $= 4$, $R_1 \leftarrow R_3 \leftarrow \{2, 3\}$, $R_2 \leftarrow R_4 \leftarrow \{1, 4\}$, prob() $= 4/24$.
- Generally, we have essentially. Let $w_j$ be the child for score $j$ and let $\varepsilon_j = \text{prob}(v \to w_j)$. Then
  - $|R_{j+1}^{w_j}| \approx \epsilon_j |R_{j+1}^v|$.
  - if $\epsilon_j > 0$, then $\epsilon_j \geq 1/n$.

# Information Gain by a Query

- For any node $v$: $R_i^v =$ values still possible for $\pi(i)$ in $v$.
- $R_i^{\text{root}} = [n]$ for all $i$.
- It is easy to keep track of the sets $R_i^v$ and the probability of having a particular score, for example for $v =$ root we have:
  - Assume we query 0110 in the root.
  - score $= 0$, $R_1 \leftarrow \{1, 4\}$, prob() $= 1/2$.
  - score $= 1$, $R_1 \leftarrow \{2, 3\}$, $R_2 \leftarrow \{2, 3\}$, prob() $= 4/24$.
  - score $= 2$, $R_1 \leftarrow \{2, 3\}$, $R_2 \leftarrow R_3 \leftarrow \{1, 4\}$, prob() $= 4/24$.
  - score $= 3$, impossible.
  - score $= 4$, $R_1 \leftarrow R_3 \leftarrow \{2, 3\}$, $R_2 \leftarrow R_4 \leftarrow \{1, 4\}$, prob() $= 4/24$.
- Generally, we have essentially. Let $w_j$ be the child for score $j$ and let $\varepsilon_j = \text{prob}(v \to w_j)$. Then
  - $|R_{j+1}^{w_j}| \approx \epsilon_j |R_{j+1}^v|$.
  - if $\epsilon_j > 0$, then $\epsilon_j \geq 1/n$.

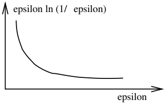
# Increase of Potential

Assume we are in node $v$ and see a score $j$ with probability $\varepsilon_j$.
Then expected increase in potential:

$$\Delta := \sum_j \varepsilon_j(\Phi(w_j) - \Phi(v)).$$

If we proceed to child $w_j$ the size of $R_{j+1}$ is multiplied by $\varepsilon_j$ (ignore effect on other sets). Thus

$$\begin{aligned}\Delta &= \sum_j \varepsilon_j \log\left(\frac{\log(2n/(\varepsilon_j|R^v_{j+1}|))}{\log(2n/|R^v_{j+1}|)}\right) = \sum_j \varepsilon_j \log\left(1 + \frac{\log 1/\varepsilon_j}{\log(2n/|R^v_{j+1}|)}\right)\\ &\le \sum_j \varepsilon_j \frac{\log(1/\varepsilon_j)}{\log(2n/|R^v_{j+1}|)} \le \sum_j \frac{1}{n} \frac{\log n}{\log(2n/|R^v_{j+1}|)}\\ &\le O\left(\sum_{i=0}^{\log n - 1} \frac{\log n}{2^i \log(2n/2^i)}\right) = O(1).\end{aligned}$$




at most $n/2^i$ indices $j$ with $|R_j| \approx 2^i$

# Summary and Open Problems

- Information theoretic lower bound: $\Omega(n)$ queries.
- Deterministic algs: $\Theta(n \log n)$ questions suffice and are needed. On average (random secret) $O(n \log\log n$ questions suffice.
- Randomized algs: $\Theta(n \log\log n)$ questions suffice in expectation and are needed.
- Open problems:
  - Average case complexity of deterministic alg: Explicite construction.
  - Lower bound for standard Mastermind with $n$ positions and $n$ colors (best upper bound is $O(n \log\log n)$, best lower bound is $\Omega(n)$).
  - This problem for larger alphabet sizes: $\Theta(n(k + \log n))$ for det. algs.
  - Applications of the problem: Some applications of Mastermind to computer privacy were found recently.

## Summary and Open Problems

- Information theoretic lower bound: $\Omega(n)$ queries.
- Deterministic algs: $\Theta(n \log n)$ questions suffice and are needed. On average (random secret) $O(n \log\log n$ questions suffice.
- Randomized algs: $\Theta(n \log\log n)$ questions suffice in expectation and are needed.
- Open problems:
  - Average case complexity of deterministic alg: Explicite construction.
  - Lower bound for standard Mastermind with $n$ positions and $n$ colors (best upper bound is $O(n \log\log n)$, best lower bound is $\Omega(n)$).
  - This problem for larger alphabet sizes: $\Theta(n(k + \log n))$ for det. algs.
  - Applications of the problem: Some applications of Mastermind to computer privacy were found recently.