The query complexity of a permutation-based variant of Mastermind

Peyman Afshani, Manindra Agrawal, Benjamin Doerr, Carola Doerr, Kasper Green Larsen, Kurt Mehlhorn

The Query Complexity of a Permutation-Based Variant of Mastermind

Peyman Afshani[1], Manindra Agrawal[2], Benjamin Doerr[3],
Carola Doerr[4]*, Kasper Green Larsen[1], Kurt Mehlhorn[5]

[1]MADALGO†, Department of Computer Science, Aarhus University, Denmark
[2]Indian Institute of Technology Kanpur, India
[3]École Polytechnique, CNRS, Laboratoire d'Informatique (LIX), Palaiseau, France
[4]Sorbonne Université, CNRS, LIP6, Paris, France
[5]Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract

We study the query complexity of a permutation-based variant of the guessing game Mastermind. In this variant, the secret is a pair (z, π) which consists of a binary string $z \in \{0,1\}^n$ and a permutation π of $[n]$. The secret must be unveiled by asking queries of the form $x \in \{0,1\}^n$. For each such query, we are returned the score

$$f_{z,\pi}(x) := \max\{i \in [0..n] \mid \forall j \leq i : z_{\pi(j)} = x_{\pi(j)}\}\,;$$

i.e., the score of x is the length of the longest common prefix of x and z with respect to the order imposed by π. The goal is to minimize the number of queries needed to identify (z, π). This problem originates from the study of black-box optimization heuristics, where it is known as the LeadingOnes problem.

In this work, we prove matching upper and lower bounds for the deterministic and randomized query complexity of this game, which are $\Theta(n \log n)$ and $\Theta(n \log \log n)$, respectively.

1 Introduction

Query complexity, also referred to as *decision tree complexity*, is one of the most basic models of computation. We aim at learning an unknown object (a secret) by asking queries of a certain type. The cost of the computation is the number of queries made until the secret is unveiled. All other computation is free. Query complexity is one of the standard measures in computational complexity theory.

A related performance measure can be found in the *theory of black-box optimization*, where we are asked to optimize a function $f : S \to \mathbb{R}$ without having access to it other than by evaluating ("querying") the function values $f(x)$ for solution candidates $x \in S$. In black-box optimization, the performance of an algorithm on a class $\mathcal{F}$ of functions is measured by the number of function evaluations that are needed until, for any unknown function $f \in \mathcal{F}$, an optimal search point is evaluated for the first time.

*Corresponding author. Carola.Doerr@mpi-inf.mpg.de
†Center for Massive Data Algorithms, a Center of the Danish National Research Foundation, Denmark

1.1 Problem Description and Summary of Main Results

In this work, we consider the query complexity of the following problem.

Let S_n denote the set of permutations of $[n] := \{1, \dots, n\}$; let $[0..n] := \{0, 1, \dots, n\}$. Our problem is that of learning a hidden permutation $\pi \in S_n$ together with a hidden bit-string $z \in \{0,1\}^n$ through queries of the following type. A query is again a bit-string $x \in \{0,1\}^n$. As answer we receive the length of the longest common prefix of x and z in the order of π, which we denote by

$$f_{z,\pi}(x) := \max\{i \in [0..n] \mid \forall j \leq i : z_{\pi(j)} = x_{\pi(j)}\}\,.$$

We call this problem the HiddenPermutation problem. It can be viewed as a guessing game like the well-known Mastermind problem (cf. Section 1.2); however, the secret is now a pair (z, π) and not just a string. The problem is a standard benchmark problem in black-box optimization.

It is easy to see (see Section 2) that learning one part of the secret (either z or π) is no easier (up to $O(n)$ questions) than learning the full secret. It is also not difficult to see that $O(n \log n)$ queries suffice deterministically to unveil the secret (see Section 3). Doerr and Winzen [DW12] showed that randomization allows to beat this bound. They gave a randomized algorithm with $O(n \log n / \log\log n)$ expected complexity. The information-theoretic lower bound is only $\Theta(n)$ as the answer to each query is a number between zero and n and hence may reveal as many as $\log n$ bits. We show that

(1) the deterministic query complexity is $\Theta(n \log n)$, cf. Section 3, and that

(2) the randomized query complexity is $\Theta(n \log\log n)$, cf. Sections 4 and 5.

Both upper bound strategies are efficient, i.e., they can be implemented in polynomial time. The lower bound is established by a (standard) adversary argument in the deterministic case and by a potential function argument in the randomized case. We remark that for many related problems (e.g., sorting, Mastermind, and many coin weighing problems) the asymptotic query complexity, or the best known lower bound for it, equals the information-theoretic lower bound. For our problem, deterministic and randomized query complexity differ, and the randomized query complexity exceeds the information theoretic lower bound. The randomized upper and lower bound require non-trivial arguments. In Section 2 we derive auxiliary results, of which some are interesting in their own right. For example, it can be decided efficiently whether a sequence of queries and answers is consistent.

A summary of the results presented in this work previously appeared in [AAD+13].

1.2 Origin of the Problem and Related Work

The HiddenPermutation problem can be seen as a *guessing game.* An archetypal and well-studied representative of guessing games is *Mastermind*, a classic two-player board game from the Seventies. In the Mastermind game, one player chooses a secret code $z \in [k]^n$. The role of the second player is to identify this code using as few guesses as possible. For each guess $x \in [k]^n$ she receives the number of positions in which z and x agree (and in some variants also the number of additional "colors" which appear in both x and z, but in different positions).

The Mastermind game has been studied in different context. Among the most prominent works is a study of Erdős and Rényi [ER63], where the 2-color variant of Mastermind is studied in the context of a coin-weighing problem. Their main result showed that the query complexity of this game is $\Theta(n/\log n)$. This bound has subsequently been generalized in various ways, most notably to the Mastermind game with $k = n$ colors. Using similar techniques as Erdős and Rényi, Chvátal [Chv83] showed an upper bound of $O(n \log n)$ for the query complexity of

this game. This bound has been improved $O(n \log\log n)$ [DDST16]. The best known lower bound is the information-theoretic linear one. This problem is open for more than 30 years.

Our permutation-based variant of Mastermind has its origins in the field of evolutionary computation. There, the LeadingOnes function $\{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\}$ counting the number of initial ones in a binary string of length n, is a commonly used benchmark function both for experimental and theoretical analyses (e.g., [Rud97]). It is studied as one of the simplest examples of a unimodal non-separable function.

An often desired feature of general-purpose optimization algorithms like genetic and evolutionary algorithms is that they should be oblivious of problem representation. In the context of optimizing functions of the form $f : \{0,1\}^n \to \mathbb{R}$, this is often formalized as an *unbiasedness* restriction, which requires that the performance of an unbiased algorithm is identical for all composed functions $f \circ \sigma$ of f with an automorphism σ of the n-dimensional hypercube. Most standard evolutionary algorithms as well many other commonly used black-box heuristics like local search variants, simulated annealing, etc. are unbiased. Most of them query an average number of $\Theta(n^2)$ solution candidates until they find the maximum of LeadingOnes, see, e.g., [DJW02].

It is not difficult to see that

$$\{\textsc{LeadingOnes} \circ \sigma \mid \sigma \text{ is an automorphism of } \{0,1\}^n\} = \{f_{z,\pi} \mid z \in \{0,1\}^n, \pi \in S_n\},$$

showing that HiddenPermutation generalizes the LeadingOnes function by indexing the bits not from left to right, but by an arbitrary permutation, and by swapping the interpretation of 0 and 1 for indices i with $z_i = 0$.

The first to study the query complexity of HiddenPermutation were Droste, Jansen, and Wegener [DJW06], who introduced the notion of *black-box complexity* as a measure for the difficulty of black-box optimization problems. As mentioned, the main objective in black-box optimization is the identification of an optimal solution using as few function evaluations as possible. Denoting by $T_A(f)$ the number of queries needed by algorithm A until it queries for the first time a search point $x \in \arg\max f$ (this is the so-called *runtime* or *optimization time* of A), the *black-box complexity* of a collection $\mathcal{F}$ of functions is

$$\inf_A \max_{f \in \mathcal{F}} \mathbb{E}[T_A(f)],$$

the best (among all algorithms) worst-case (with respect to all $f \in \mathcal{F}$) expected runtime. Black-box complexity is essentially query-complexity, with a focus on optimization rather than on learning. A survey on this topic can be found in [Doe18].

Droste et al. [DJW06] considered only the 0/1-invariant class $\{f_{z,\mathrm{id}} \mid z \in \{0,1\}^n\}$ (where id denotes the identity permutation). They showed that the black-box complexity of this function class is $n/2 \pm o(n)$.

The first bound for the black-box complexity of the full class HiddenPermutation was presented in [LW12] by Lehre and Witt, who showed that any so-called *unary unbiased black-box algorithm* needs $\Omega(n^2)$ steps, on average, to solve the HiddenPermutation problem. In [DJK+11] it was then shown that already with binary distributions one can imitate a binary search algorithm (similar to the one presented in the proof of Theorem 3.1), thus yielding an $O(n \log n)$ black-box algorithm for the HiddenPermutation problem. The algorithm achieving the $O(n \log n / \log\log n)$ bound mentioned in Section 1.1 can be implemented as a ternary unbiased one [DW12]. This bound, as mentioned above, was the previously best known upper bound for the black-box complexity of HiddenPermutation.

In terms of lower bounds, the best known bound was the linear information-theoretic one. However, more recently a stronger lower bound has been proven to hold for a restricted class

of algorithms. More precisely, Doerr and Lengler [DL18] recently proved that the so-called $(1+1)$ *elitist black-box complexity* is $\Omega(n^2)$, showing that any algorithm beating this bound has to keep in its memory more than one previously evaluated search point or has to make use of information hidden in non-optimal search points. It is conjectured in [DL18] that the (1+1) memory-restriction alone (which allows an algorithm to store only one previously queried search point in its memory and to evaluate in each iteration only one new solution candidate) already causes the $\Omega(n^2)$ bound, but this conjecture stands open. We note that many local search variants, including simulated annealing, are $(1+1)$ memory-restricted.

2 Preliminaries

For all positive integers $k \in \mathbb{N}$ we define $[k] := \{1, \dots, k\}$ and $[0..k] := [k] \cup \{0\}$. By e_k^n we denote the kth unit vector $(0, \dots, 0, 1, 0, \dots, 0)$ of length n. For a set $I \subseteq [n]$ we define $e_I^n := \sum_{i \in I} e_i^n = \oplus_{i \in I} e_i^n$, where $\oplus$ denotes the bitwise exclusive-or. We say for two bitstrings $x, y \in \{0,1\}^n$ that we *created y from x by flipping I* or that we *created y from x by flipping the entries in position(s) I* if $y = x \oplus e_I^n$. By S_n we denote the set of all permutations of $[n]$. For $r \in \mathbb{R}_{\geq 0}$, let $\lceil r \rceil := \min\{n \in \mathbb{N}_0 \mid n \geq r\}$. and $\lfloor r \rfloor := \max\{n \in \mathbb{N}_0 \mid n \leq r\}$. To increase readability, we sometimes omit the $\lceil \cdot \rceil$ signs; that is, whenever we write r where an integer is required, we implicitly mean $\lceil r \rceil$.

Let $n \in \mathbb{N}$. For $z \in \{0,1\}^n$ and $\pi \in S_n$ we define

$$f_{z,\pi} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : z_{\pi(j)} = x_{\pi(j)}\}.$$

z and π are called the *target string* and *target permutation* of $f_{z,\pi}$, respectively. We want to identify target string and permutation by asking queries x^i, $i = 1, 2, \dots$, and evaluating the answers ("scores") $s^i = f_{z,\pi}(x^i)$. We may stop after t queries if there is only a *single* pair $(z, \pi) \in \{0,1\}^n \times S_n$ with $s^i = f_{z,\pi}(x^i)$ for $1 \leq i \leq t$.

A *deterministic strategy* for the HiddenPermutation problem is a tree of outdegree $n+1$ in which a query in $\{0,1\}^n$ is associated with every node of the tree. The search starts as the root. If the search reaches a node, the query associated with the node is asked, and the search proceeds to the child selected by the score. The complexity of a strategy on input (z, π) is the number of queries required to identify the secret, and the complexity of a deterministic strategy is the worst-case complexity of any input. That is, the complexity is the height of the search tree.

A *randomized strategy* is a probability distribution over deterministic strategies. The complexity of a randomized strategy on input (z, π) is the expected number of queries required to identify the secret, and the complexity of a randomized strategy is the worst-case complexity of any input. The probability distribution used for our randomized upper bound is a product distribution in the following sense: a probability distribution over $\{0,1\}^n$ is associated with every node of the tree. The search starts as the root. In any node, the query is selected according to the probability distribution associated with the node, and the search proceeds to the child selected by the score.

We remark that knowing z allows us to determine π with $n-1$ queries $z \oplus e_i^n$, $1 \leq i < n$. Observe that $\pi^{-1}(i)$ equals $f_{z,\pi}(z \oplus e_i^n) + 1$. Conversely, knowing the target permutation π we can identify z in a linear number of guesses. If our query x has a score of k, all we need to do next is to query the string x' that is created from x by flipping the entry in position $\pi(k+1)$. Thus, learning one part of the secret is no easier (up to $O(n)$ questions) than learning the full.

A simple information-theoretic argument gives an $\Omega(n)$ lower bound for the deterministic query complexity and, together with Yao's minimax principle [Yao77], also for the randomized

complexity. The *search space* has size $2^n n!$, since the unknown secret is an element of $\{0,1\}^n \times S_n$. A deterministic strategy is a tree with outdegree $n+1$ and $2^n n!$ leaves. The maximal and average depth of any such tree is $\Omega(n)$.

Let $\mathcal{H} := (x^i, s^i)_{i=1}^t$ be a sequence of queries $x^i \in \{0,1\}^n$ and scores $s^i \in [0..n]$. We call $\mathcal{H}$ a *guessing history*. A secret (z,π) is *consistent with* $\mathcal{H}$ if $f_{z,\pi}(x^i) = s^i$ for all $i \in [t]$. $\mathcal{H}$ is *feasible* if there exists a secret consistent with it.

An observation crucial in our proofs is the fact that a vector $(V_1, \ldots, V_n)$ of subsets of $[n]$, together with a top score query (x^*, s^*), captures the total knowledge provided by a *guessing history* $\mathcal{H} = (x^i, s^i)_{i=1}^t$ about the set of secrets consistent with $\mathcal{H}$. We will call V_j the *candidate set* for position j; V_j will contain all indices $i \in [n]$ for which the following simple rules (1) to (3) do not rule out that $\pi(j)$ equals i. Put differently, V_j contains the set of values that are still possible images for $\pi(j)$, given the previous queries.

Theorem 2.1. *Let $t \in \mathbb{N}$, and let $\mathcal{H} = (x^i, s^i)_{i=1}^t$ be a guessing history. Construct the candidate sets $V_1, \ldots, V_n \subseteq [n]$ according to the following rules:*

(1) If there are h and ℓ with $j \le s^h \le s^\ell$ and $x_i^h \neq x_i^\ell$, then $i \notin V_j$.

(2) If there are h and ℓ with $s = s^h = s^\ell$ and $x_i^h \neq x_i^\ell$, then $i \notin V_{s+1}$.

(3) If there are h and ℓ with $s^h < s^\ell$ and $x_i^h = x_i^\ell$, then $i \notin V_{s^h+1}$.

(4) If i is not excluded by one of the rules above, then $i \in V_j$.

Furthermore, let $s^* := \max\{s^1, \ldots, s^t\}$ and let $x^* = x^j$ for some j with $s^j = s^*$.

Then a pair (z,π) is consistent with $\mathcal{H}$ if and only if

(a) $f_{z,\pi}(x^*) = s^*$ and

(b) $\pi(j) \in V_j$ for all $j \in [n]$.

Proof. Let (z,π) satisfy conditions (a) and (b). We show that (z,π) is consistent with $\mathcal{H}$. To this end, let $h \in [t]$, let $x = x^h$, $s = s^h$, and $f := f_{z,\pi}(x)$. We need to show $f = s$.

Assume $f < s$. Then $z_{\pi(f+1)} \neq x_{\pi(f+1)}$. Since $f+1 \le s^*$, this together with (a) implies $x_{\pi(f+1)} \neq x^*_{\pi(f+1)}$. Rule (1) yields $\pi(f+1) \notin V_{f+1}$; a contradiction to (b).

Similarly, if we assume $f > s$, then $x_{\pi(s+1)} = z_{\pi(s+1)}$. We distinguish two cases. If $s < s^*$, then by condition (a) we have $x_{\pi(s+1)} = x^*_{\pi(s+1)}$. By rule (3) this implies $\pi(s+1) \notin V_{s+1}$; a contradiction to (b).

On the other hand, if $s = s^*$, then $x_{\pi(s+1)} = z_{\pi(s+1)} \neq x^*_{\pi(s+1)}$ by (a). Rule (2) implies $\pi(s+1) \notin V_{\pi(s+1)}$, again contradicting (b).

Necessity is trivial. □

We may also construct the sets V_j incrementally. The following update rules are direct consequences of Theorem 2.1. In the beginning, let $V_j := [n]$, $1 \le j \le n$. After the first query, record the first query as x^* and its score as s^*. For all subsequent queries, do the following: Let I be the set of indices in which the current query x and the current best query x^* agree. Let s be the score of x and let s^* be the score of x^*.

Rule 1: If $s < s^*$, then $V_i \leftarrow V_i \cap I$ for $1 \le i \le s$ and $V_{s+1} \leftarrow V_{s+1} \setminus I$.

Rule 2: If $s = s^*$, then $V_i \leftarrow V_i \cap I$ for $1 \le i \le s^* + 1$.

Rule 3: If $s > s^*$, then $V_i \leftarrow V_i \cap I$ for $1 \le i \le s^*$ and $V_{s^*+1} \leftarrow V_{s^*+1} \setminus I$. We further replace $s^* \leftarrow s$ and $x^* \leftarrow x$.

It is immediate from the update rules that the sets V_j form a *laminar family*; i.e., for $i < j$ either $V_i \cap V_j = \emptyset$ or $V_i \subseteq V_j$. As a consequence of Theorem 2.1 we obtain a polynomial time test for the feasibility of histories. It gives additional insight in the meaning of the candidate sets $V_1, \ldots, V_n$.

Theorem 2.2. *It is decidable in polynomial time whether a guessing history is feasible. Furthermore, we can efficiently compute the number of pairs $(z, \pi) \in \{0,1\}^n \times S_n$ consistent with it.*

Proof. We first show that feasibility can be checked in polynomial time. Let $\mathcal{H} = (x^i, s^i)_{i=1}^t$ be given. Construct the sets $V_1, \ldots, V_n$ as described in Theorem 2.1. Now construct a bipartite graph $G(V_1, \ldots, V_n)$ with node set $[n]$ on both sides. Connect j to all nodes in V_j on the other side. Permutations π with $\pi(j) \in V_j$ for all j are in one-to-one correspondence to perfect matchings in this graph. If there is no perfect matching, the history in infeasible. Otherwise, let π be any permutation with $\pi(j) \in V_j$ for all j. We next construct z. We use the obvious rules:

(a) If $i = \pi(j)$ and $j \leq s^h$ for some $h \in [t]$ then set $z_i := x_i^h$.
(b) If $i = \pi(j)$ and $j = s^h + 1$ for some $h \in [t]$ then set $z_i := 1 - x_i^h$.
(c) If z_i is not defined by one of the rules above, set it to an arbitrary value.

We need to show that these rules do not lead to a contradiction. Assume otherwise. There are three ways, in which we could get into a contradiction. There is some $i \in [n]$ and some $x^h, x^\ell \in \{0,1\}^n$

(1) setting z_i to opposite values by rule (a)
(2) setting z_i to opposite values by rule (b)
(3) setting z_i to opposite values by rules (b) applied to x^h and rule (a) applied to x^ℓ.

In each case, we readily derive a contradiction. In the first case, we have $j \leq s^h$, $j \leq s^\ell$ and $x_i^h \neq x_i^\ell$. Thus $\pi(j) = i \notin V_j$ by rule (1). In the second case, we have $j = s^h + 1 = s^\ell + 1$ and $x_i^h \neq x_i^\ell$. Thus $i \notin V_j$ by (2). In the third case, we have $j = s^h + 1$, $j \leq s^\ell$, and $x_i^h = x_i^\ell$. Thus $i \notin V_j$ by (3).

Finally, the pair (z, π) defined in this way is clearly consistent with the history.

Next we show how to efficiently compute the number of consistent pairs. We recall Hall's condition for the existence of a perfect matching in a bipartite graph. A perfect matching exists if and only if $|\bigcup_{j \in J} V_j| \geq |J|$ for every $J \subseteq [n]$. According to Theorem 2.1, the guessing history $\mathcal{H}$ can be equivalently described by a state $(V_1, \ldots, V_{s^*+1}, x^*, s^*)$. How many pairs (z, π) are compatible with this state?

Once we have chosen π, there are exactly $2^{n-(s^*+1)}$ different choices for z if $s^* < n$ and exactly one choice if $s^* = n$. The permutations can be chosen in a greedy fashion. We fix $\pi(1), \ldots, \pi(n)$ in this order. The number of choices for $\pi(i)$ equals $|V_i|$ minus the number of $\pi(j)$, $j < i$, lying in V_i. If V_j is disjoint from V_i, $\pi(j)$ never lies in V_i and if V_j is contained in V_i, $\pi(j)$ is always contained in V_i. Thus the number of permutations is equal to

$$\prod_{1 \leq i \leq n} \left(|V_i| - |\{j < i \mid V_j \subseteq V_i\}| \right) .$$

It is easy to see that the greedy strategy does not violate Hall's condition. □

The proof of Theorem 2.2 explains which values in V_i are actually possible as value for $\pi(i)$. A value $\ell \in V_i$ is feasible if there is a perfect matching in the graph $G(V_1, \ldots, V_n)$ containing the edge (i, ℓ). The existence of such a matching can be decided in polynomial time; we only need to test for a perfect matching in the graph $G \setminus \{i, \ell\}$. Hall's condition says that there is no such perfect matching if there is a set $J \subseteq [n] \setminus \{i\}$ such that $|\bigcup_{j \in J} V_j \setminus \{\ell\}| < |J|$. Since G contains a perfect matching (assuming a consistent history), this implies $|\bigcup_{j \in J} V_j| = |J|$; i.e., J is tight for Hall's condition. We have now shown: Let $\ell \in V_i$. Then ℓ is infeasible for $\pi(i)$ if and only if there is a tight set J with $i \notin J$ and $\ell \in \bigcup_{j \in J} V_j$. Since the V_i form a laminar family,

Algorithm 1: A deterministic $O(n \log n)$ strategy for the HiddenPermutation problem. We write f instead of $f_{z,\pi}$

Initialization: $x \leftarrow (0, \dots, 0)$;
for $i = 1, ..., n$ **do**
  // $f(x) \geq i-1$ and $\pi(1), \dots, \pi(i-1)$ are already determined
  $\pi(i) \leftarrow$ `BinSearch`$(x, i, [n] \setminus \{\pi(1), \dots, \pi(i-1)\})$;
  Update x by flipping $\pi(i)$;

// where `BinSearch` is the following function.
`BinSearch`(x, i, V)
if $f(x) > i-1$ **then** update x by flipping all bits in V;
while $|V| > 1$ **do**
  // $\pi(i) \in V$, $\pi(1), \dots, \pi(i-1) \notin V$, and $f(x) = i-1$
  Select a subset $F \subseteq V$ of size $|V|/2$;
  Create y from x by flipping all bits in F and query $f(y)$;
  if $f(y) = i-1$ **then** $V \leftarrow V \setminus F$;
  else $V \leftarrow F$;
return the element in V;

minimal tight sets have a special form; they consist of an i and all j such that V_j is contained in V_i. In the counting formula for the number of permutations such i are characterized by $|V_i| - |\{j < i \mid V_j \subseteq V_i\}| = 1$. In this situation, the elements of V_i are infeasible for all $\pi(j)$ with $j > i$. We may subtract V_i from each V_j with $j > i$.

If Hall's condition is tight for some J, i.e., $|\bigcup_{j \in J} V_j| = |J|$, we can easily learn how π operates on J. We have $V_j = [n]$ for $j > s^* + 1$ and hence the largest index in J is at most $s^* + 1$. Perturb x^* by flipping each bit in $\bigcup_{j \in J} V_j$ exactly once. The scores determine the permutation.

The fact that the sets V_j are a laminar family is crucial for the counting result. Counting the number of perfect matchings in a general bipartite graph is #P-complete [Val79].

3 Deterministic Complexity

We settle the deterministic query complexity of the HiddenPermutation problem. The upper and lower bound match up to a small constant factor. Specifically, we prove

Theorem 3.1. *The deterministic query complexity of the* HiddenPermutation *problem with n positions is $\Theta(n \log n)$.*

Proof. The *upper bound* is achieved by an algorithm that resembles binary search and iteratively identifies $\pi(1), \dots, \pi(n)$ and the corresponding bit values $z_{\pi(1)}, \dots, z_{\pi(n)}$: We start by setting the set V of candidates for $\pi(1)$ to $[n]$ and by determinining a string x with score 0; either the all-zero-string or the all-one-string will work. We iteratively reduce the size of V keeping the invariant that $\pi(1) \in V$. We select an arbitrary subset F of V of size $|V|/2$ and create y from x by flipping the bits in F. If $f_{z,\pi}(y) = 0$, $\pi(1) \notin F$, if If $f_{z,\pi}(y) > 0$, $\pi(1) \in F$. In either case, we essentially halve the size of the candidate set. Continuing in this way, we determine $\pi(1)$ in $O(\log n)$ queries. Once $\pi(1)$ and $z_{\pi(1)}$ are known, we iterate this strategy on the remaining bit positions to determine $\pi(2)$ and $z_{\pi(2)}$, and so on, yielding an $O(n \log n)$ query strategy for identifying the secret. The details are given in Algorithm 1.

The *lower bound* is proved by examining the decision tree of the deterministic query scheme and exhibiting an input for which the number of queries asked is high. More precisely, we show that for every deterministic strategy, there exists an input (z, π) such that after $\Omega(n \log n)$ queries the maximal score ever returned is at most $n/2$. This is done by a simple adversarial argument: First consider the root node r of the decision tree. Let x^1 be the first query. We proceed to the child corresponding to score 1. According to the rules from the preceding section V_1 to V_n are initialized to $[n]$. Let x be the next query asked by the algorithm and let I be the set of indices in which x and x^1 agree.

(1) If we would proceed to the child corresponding to score 0, then V_1 would become $V_1 \setminus I$ and V_2 would not not change according to Rule 1.

(2) If we would proceed to the child corresponding to score 1, then V_1 would become $V_1 \cap I$ and V_2 would become $V_2 \cap I$ according to Rule 2.

We proceed to the child where the size of V_1 at most halves. Observe that $V_1 \subseteq V_2$ always, and the maximum score is 1. Moreover, V_3 to V_n are not affected.

We continue in this way until $|V_1| = 2$. Let v be the vertex of the decision tree reached. Query $x^* = x^1$ is still the query with maximum score. We choose $i_1 \in V_1$ and $i_2 \in V_2$ arbitrarily and consider the subset of all inputs for which $i_1 = \pi(1)$, $i_2 = \pi(2)$, $z_{i_1} = x^*_{i_1}$, and $z_{i_2} = 1 - x^*_{i_2}$. For all such inputs, the query path followed in the decision tree descends from the root to the node v. For this collection of inputs, observe that there is one input for every assignment of values to $\pi(3), \ldots, \pi(n)$ different from i_1 and i_2, and for every assignment of 0/1 values to $z_{\pi(3)}, \ldots, z_{\pi(n)}$. Hence we can recurse on this subset of inputs starting at v ignoring $V_1, V_2, \pi(1), \pi(2), z_{\pi(1)}$, and $z_{\pi(2)}$. The setup is identical to what we started out with at the root, with the problem size decreased by 2. We proceed this way, forcing $\Omega(\log n)$ queries for every two positions revealed, until we have returned a score of $n/2$ for the first time. At this point, we have forced at least $n/4 \cdot \Omega(\log n) = \Omega(n \log n)$ queries. □

4 The Randomized Strategy

We now show that the randomized query complexity is only $O(n \log \log n)$. The randomized strategy overcomes the sequential learning process of the binary search strategy (typically revealing a constant amount of information per query) and instead has a typical information gain of $\Theta(\log n / \log \log n)$ bits per query. In the language of the candidate sets V_i, we manage to reduce the sizes of many of these sets in parallel, that is, we gain information on several values $\pi(i)$ despite the seemingly sequential way the function $f_{z,\pi}$ offers information. The key to this is using partial information given by the V_i (that is, information that does not determine π_i, but only restricts it) to guess with good probability an x with $f_{z,\pi}(x) > s^*$.

Theorem 4.1. *The randomized query complexity of the* HiddenPermutation *problem with n positions is $O(n \log \log n)$.*

The strategy has two parts. In the first part, we identify the positions $\pi(1), \ldots, \pi(q)$ and the corresponding bit values $z_{\pi(1)}, \ldots, z_{\pi(q)}$ for some $q \in n - \Theta(n/\log n)$ with $O(n \log \log n)$ queries. In the second part, we find the remaining $n - q \in \Theta(n/\log n)$ positions and entries using the binary search algorithm with $O(\log n)$ queries per position.

4.1 A High Level View of the First Part

We give a high level view of the first part of our randomized strategy. Here and in the following we denote by s^* the current best score, and by x^* we denote a corresponding query; i.e., $f_{z,\pi}(x^*) = s^*$. For brevity, we write f for $f_{z,\pi}$.

The goal of any strategy must be to increase s^* and to gain more information about π by reducing the sets $V_1, \ldots, V_{s^*+1}$. Our strategy carefully balances the two subgoals. If $|\bigcup_{i \le s^*+1} V_i|$ is "large", it concentrates on reducing sets, if $|\bigcup_{i \le s^*+1} V_i|$ is "small", it concentrates on increasing s^*. The latter will simulatenously reduce V_{s^*+1}.

We arrange the candidate sets V_1 to V_n into $t+2$ *levels* 0 to $t+1$, where $t = \Theta(\log \log n)$. Initially, all candidate sets are on level 0, and we have $V_i = [n]$ for all $i \in [n]$. The sets in level i have larger index than the sets in level $i+1$. Level $t+1$ contains an initial segment of candidate sets, and all candidate sets on level $t+1$ are singletons, i.e., we have identified the corresponding π-value. On level i, $1 \le i \le t$, we can have up to α_i sets. We also say that the *capacity* of level i is α_i. The size of any set on level i is at most n/α_i^d, where d is any constant greater than or equal to 4. We choose $\alpha_1 = \log n$, $\alpha_i = \alpha_{i-1}^2$ for $1 \le i \le t$ and t maximal such that $\alpha_t^d \le n$. Depending on the status (i.e., the fill rate) of these levels, either we try to increase s^*, or we aim at reducing the sizes of the candidate sets.

The algorithm maintains a counter $s \le s^*$ and strings $x, y \in \{0,1\}^n$ with $f(x) = s < f(y)$. The following invariants hold for the candidate sets V_1 to V_n:

(1) $\pi(j) \in V_j$ for all j.
(2) The sets V_j, $j \le s$, are pairwise disjoint.
(3) $V_j = [n]$ for $j > s$.
(4) $V_j \setminus \{\pi(j)\}$ is random. More precisely, there is a set $V_j^* \subseteq [n]$ such that $\pi(j) \in V_j$ and $V_j \setminus \{\pi(j)\}$ is a random subset of $V_j^* \setminus \{\pi(j)\}$ of size $|V_j| - 1$.

Our first goal is to increase s^* to $\log n$ and to move the sets $V_1, \ldots, V_{\log n}$ to the first level, i.e., to decrease their size to $n/\alpha_1^d = n/\log^d n$. This is done sequentially. We start by querying $f(x)$ and $f(y)$, where x is arbitrary and $y = x \oplus 1^n$ is the bitwise complement of x. By swapping x and y if needed, we may assume $f(x) = 0 < f(y)$. We now run a randomized binary search for finding $\pi(1)$. We choose uniformly at random a subset $F_1 \subseteq V_1$ ($V_1 = [n]$ in the beginning) of size $|F_1| = |V_1|/2$. We query $f(y')$ where y' is obtained from x by flipping the bits in F_1. If $f(y') = 0$, we set $V_1 \leftarrow V_1 \setminus F_1$; we set $V_1 \leftarrow F_1$ otherwise. This ensures $\pi(1) \in V_1$ and invariant (4) We stop this binary search once $\pi(2) \notin V_1$ is sufficiently likely; the analysis will show that $\Pr[\pi(2) \in V_1] \le 1/\log^d n$ (and hence $|V_1| \le n/\log^d n$) for some large enough constant d is a good choice.

We next try increase s to a value larger than one and to simultaneously decrease the size of V_2. Let $\{x, y\} = \{y, y \oplus 1_{[n]\setminus V_1}\}$. If $\pi(2) \notin V_1$, one of $f(x)$ and $f(y)$ is one and the other is larger than one. Swapping x and y if necessary, we may assume $f(x) = 1 < f(y)$. We use randomized binary search to reduce the size of V_2 to $n/\log^d n$. The randomized binary search is similar to before. Initially, V_2 is equal to $V_2^* = [n] - V_1$. At each step we chose a subset $F_2 \subseteq V_2$ of size $|V_2|/2$ and we create y' from x by flipping the bits in positions F_2. If $f(y') > 1$ we update V_2 to F_2 and we update V_2 to $V_2 \setminus F_2$ otherwise. We stop once $|V_2| \le n/\log^d n$.

At this point we have $|V_1|, |V_2| \le n/\log^d n$ and $V_1 \cap V_2 = \emptyset$. We hope that $\pi(3) \notin V_1 \cup V_2$, in which case we can increase s to three and move set V_3 from level 0 to level 1 by random binary search (the case $\pi(3) \in V_1 \cup V_2$ is called a *failure* and will be treated separately at the end of this overview).

At some point the probability that $\pi(i) \notin V_1 \cup \ldots \cup V_{i-1}$ drops below a certain threshold and we cannot ensure to make progress anymore by simply querying $y \oplus ([n] \setminus (V_1 \cup \ldots \cup V_{i-1}))$. This situation is reached when $i = \log n$ and hence we abandon the previously described strategy once $s = \log n$. At this point, we move our focus from increasing s to reducing the size of the candidate sets $V_1, \ldots, V_s$, thus adding them to the second level. More precisely, we reduce their sizes to at most $n/\log^{2d} n = n/\alpha_2^d$. This reduction is carried out by `SizeReduction`, which we describe in Section 4.3. It reduces the sizes of the up to $\alpha_{\ell-1}$ candidate sets from some value $\le n/\alpha_{\ell-1}^d$ to the target size n/α_ℓ^d of level ℓ with an expected number of $O(1)\alpha_{\ell-1} d(\log(\alpha_\ell) - \log(\alpha_{\ell-1}))/\log(\alpha_{\ell-1})$

Algorithm 2: The $O(n \log\log n)$ strategy for the HiddenPermutation problem with n positions.

Input: Number of levels t. Capacities $\alpha_1, \ldots, \alpha_t \in \mathbb{N}$ of the levels $1, \ldots, t$. Score $q \in n - \Theta(n/\log n)$ that is to be achieved in the first phase. Positive integer $d \in \mathbb{N}$.
Main Procedure
$V_1, \ldots, V_n \leftarrow [n]$; // V_i is the set of candidates for $\pi(i)$
$s \leftarrow 0$; // s counts the number of successful iterations
$x \leftarrow 0^n$; $y \leftarrow 1^n$; $J \leftarrow \emptyset$; **if** $f(x) > 0$ **then** swap x and y; // $f(x) = s < f(y)$ and $J = \{1, \ldots, s\}$
while $|J| < q$ **do**
  // $J = [s]$, $V_j = \{\pi(j)\}$ for $j \in J$, $f(x) = s < f(y)$, and $\pi(s+1) \in [n] \setminus \bigcup_{j \le s} V_j$
  $J' \leftarrow \mathit{Advance}(t)$; // $J' \neq \emptyset$
  Reduce the size of the sets V_j with $j \in J'$ to 1 by calling `SizeReduction`$(\alpha_t, J', 1, x)$;
  $J \leftarrow J \cup J'$;
Part 2: Identify values $\pi(n-q+1), \ldots, \pi(n)$ and corresponding bits using `BinSearch`;
// where *Advance* is the following function.
Advance(level ℓ)
// $\pi(s+1) \notin \bigcup_{j=1}^{s} V_j$, $f(x) = s < f(y)$ and invariants (1) to (4) hold.
// returns a set J of up to α_ℓ indices such that $|V_j| \le n/\alpha_\ell^d$ for all $j \in J$
$J \leftarrow \emptyset$;
while $|J| \le \alpha_\ell$ **do**
  // $\pi(s+1) \notin \bigcup_{j=1}^{s} V_j$, $f(x) = s < f(y)$ and invariants (1) to (4) hold. $|V_j| \le n/\alpha_\ell^d$ for $j \in J$.
  if $\ell = 1$ **then**
    $V^*_{s+1} \leftarrow [n] \setminus \bigcup_{j=1}^{s} V_j$;
    $V_{s+1} \leftarrow$ `RandBinSearch`$(x, s+1, V^*_{s+1}, n/\alpha_1^d)$; // Reduce $|V_{s+1}|$ to n/α_1^d
    $s \leftarrow s + 1$;
    $J \leftarrow J \cup \{s\}$;
    $x \leftarrow y$; // establishes $f(x) \ge s$
  else
    $J' \leftarrow \mathit{Advance}(\ell - 1)$; // $J' \neq \emptyset$, and $s = \max J' \le f(x)$
    Reduce the sets V_j, $j \in J'$, to size n/α_ℓ^d using `SizeReduction`$(\alpha_{\ell-1}, J', n/\alpha_\ell^d, x)$;
    $J \leftarrow J \cup J'$;
  Create y from x by flipping all bits in $[n] \setminus \bigcup_{j=1}^{s} V_j$ and query $f(y)$;
  if ($f(x) > s$ and $f(y) > s$) or ($f(x) = s$ and $f(y) = s$) **then**
    break; // $\pi(s+1) \in \bigcup_{j=1}^{s} V_j$; failure on level ℓ
  if $f(x) > s$ **then** swap x and y; // $\pi(s+1) \notin \bigcup_{j=1}^{s} V_j$ and $f(x) = s < f(y)$
return J;

queries.

Once the sizes $|V_1|, \ldots, |V_s|$ have been reduced to at most $n/\log^{2d} n$, we move our focus back to increasing s. The probability that $\pi(s+1) \in V_1 \cup \ldots \cup V_s$ will now be small enough (details below), and we proceed as before by flipping in x the entries in the positions $[n] \setminus (V_1 \cup \ldots \cup V_s)$ and reducing the size of V_{s+1} to $n/\log^d n$. Again we iterate this process until the first level is filled; i.e., until we have $s = 2\log n$. As we did with $V_1, \ldots, V_{\log n}$, we reduce the sizes of

$V_{\log n+1}, \ldots, V_{2\log n}$ to $n/\log^{2d} n = n/\alpha_2^d$, thus adding them to the second level. We iterate this process of moving $\log n$ sets from level 0 to level 1 and then moving them to the second level until $\log^2 n = \alpha_2$ sets have been added to the second level. At this point the second level has reached its capacity and we proceed by reducing the sizes of $V_1, \ldots, V_{\log^2 n}$ to at most $n/\log^{4d} n = n/\alpha_3^d$, thus adding them to the *third level.*

In total we have $t = O(\log\log n)$ levels. For $1 \le i \le t$, the ith level has a capacity of $\alpha_i := \log^{2^{i-1}} n$ sets, each of which is required to be of size at most n/α_i^d. Once level i has reached its capacity, we reduce the size of the sets on the ith level to at most n/α_{i+1}^d, thus moving them from level i to level $i+1$. When α_t sets $V_{i+1}, \ldots, V_{i+\alpha_t}$ have been added to the last level, level t, we finally reduce their sizes to one. This corresponds to determining $\pi(i+j)$ for each $j \in [\alpha_t]$.

Failures: We say that a failure happens if we want to move some set V_{s+1} from level 0 to level 1, but $\pi(s+1) \in V_1 \cup \ldots \cup V_s$. In case of a failure, we immediately stop our attempt of increasing s. Rather, we *abort* the first level and move all sets on the first level to the second one. As before, this is done by calls to `SizeReduction` which reduce the size of the sets from at most $n/\log^d n$ to at most $n/\log^{2d} n$. We test whether we now have $\pi(s+1) \notin V_1 \cup \ldots \cup V_s$. Should we still have $\pi(s+1) \in V_1 \cup \ldots \cup V_s$, we continue by moving all level 2 sets to level 3, and so on, until we finally have $\pi(s+1) \notin V_1 \cup \ldots \cup V_s$. At this point, we proceed again by moving sets from level 0 to level 1, starting of course with set V_{s+1}. The condition $\pi(s+1) \notin V_1 \cup \ldots \cup V_s$ will certainly be fulfilled once we have moved V_1 to V_s to level $t+1$, i.e., have reduced them to singletons.

Part 2: In the second part of Algorithm 2 we determine the last $\Theta(n/\log n)$ entries of z and π. This can be done as follows. When we leave the first phase of Algorithm 2, we have $|V_1| = \ldots = |V_q| = 1$ and $f(x) \ge q$. We can now proceed as in deterministic algorithm (Algorithm 1) and identify each of the remaining entries with $O(\log n)$ queries. Thus the total number of queries in Part 2 is linear.

Our strategy is formalized by Algorithm 2. In what follows, we first present the two subroutines, `RandBinSearch` and `SizeReduction`. In Section 4.4, we present the full proof of Theorem 4.1.

4.2 Random Binary Search

`RandBinSearch` is called by the function *Advance*(1). It reduces the size of a candidate set from some value $v \le n$ to some value $\ell < v$ in $\log v - \log \ell$ queries.

Algorithm 3: A call `RandBinSearch`$(x, s+1, V, \ell)$ reduces the size of the candidate set V for V_{s+1} from v to ℓ in $\log v - \log \ell$ queries.

Input: A position s, a string $x \in \{0,1\}^n$ with $f(x) = s$, and a set V with $\pi(s+1) \in V$ and $\pi(1), \ldots, \pi(s) \notin V$, and a target size $\ell \in \mathbb{N}$.
while $|V| > \ell$ **do**
  // $\pi(s+1) \in V$, $\pi(1), \ldots, \pi(s) \notin V$, and $f(x) = s$
  Uniformly at random select a subset $F \subseteq V$ of size $|V|/2$;
  Create y' from x by flipping all bits in F and query $f(y')$;
  if $f(y') = s$ **then** $V \leftarrow V \backslash F$;
  else $V \leftarrow F$;
Output: Set V of size at most ℓ.

Lemma 4.2. *Let $x \in \{0,1\}^n$ with $f(x) = s$ and let V be any set with $\pi(s+1) \in V$ and $\pi(j) \notin V$ for $j \leq s$. Let $v := |V|$ and $\ell \in \mathbb{N}$ with $\ell < v$. Algorithm 3 reduces the size of V to ℓ using at most $\lceil \log v - \log \ell \rceil$ queries.*

Proof. Since $f(x) = s$, we have $x_{\pi(i)} = z_{\pi(i)}$ for all $i \in [s]$ and $x_{\pi(s+1)} \neq z_{\pi(s+1)}$. Also $\pi(s+1) \in V$ and $\pi(j) \notin V$ for $j \leq s$. Therefore, either we have $f(y') > s$ in line 4 or we have $f(y') = s$. In the former case, the bit $\pi(s+1)$ was flipped, and hence $\pi(s+1) \in F$ must hold. In the latter case the bit in position $\pi(s+1)$ bit was not flipped and we infer $\pi(s+1) \notin F$.

The runtime bound follows from the fact that the size of the set V halves in each iteration. □

We call `RandBinSearch` in Algorithm 2 (line 16) to reduce the size of V_{s+1} to n/α_1^d, or, put differently, to reduce the number of candidates for $\pi(s+1)$ to n/α_1^d. As the initial size of V_{s+1} is at most n, this requires at most $d \log \alpha_1$ queries by Lemma 4.2.

Lemma 4.3. *A call of Advance(1) in Algorithm 2 requires at most $\alpha_1 + \alpha_1 d \log \alpha_1$ queries.*

Proof. The two occasions where queries are made in Algorithm 2 are in line 24 and in line 16. Line 24 is executed at most α_1 times, each time causing exactly one query. Each call to `RandBinSearch` in line 16 causes at most $d \log \alpha_1$ queries, and `RandBinSearch` is called at most α_1 times. □

4.3 Size Reduction

We describe the second subroutine of Algorithm 2, `SizeReduction`. This routine is used to reduce the sizes of the up to $\alpha_{\ell-1}$ candidate sets returned by a recursive call *Advance*$(\ell - 1)$ from some value $\leq n/\alpha_{\ell-1}^d$ to at most the target size of level ℓ, which is n/α_ℓ^d. As we shall see below, this requires an expected number of $O(1)\alpha_{\ell-1} d(\log \alpha_\ell - \log \alpha_{\ell-1})/\log \alpha_{\ell-1}$ queries. The pseudo-code of `SizeReduction` is given in Algorithm 4. It repeatedly calls a subroutine `ReductionStep` that reduces the sizes of at most k candidate sets to a kth fraction of their original size using at most $O(k)$ queries, where k is a parameter. We use `ReductionStep` with parameter $k = \alpha_{\ell-1}$ repeatedly to achieve the full reduction of the sizes to at most n/α_ℓ^d.

Algorithm 4: A call `SizeReduction`(k, J, m, x) reduces the size of at most k sets V_j, $j \in J$, to size at most m. We use it twice in our main strategy. In line 22, we call `SizeReduction`$(\alpha_{\ell-1}, J', n/\alpha_\ell^d, x)$ to reduce the size of each V_j, $j \in J'$ to n/α_ℓ^d. In line 8, we call `SizeReduction`$(\alpha_t, J', 1, x)$ to reduce the size of each V_j, $j \in J'$, to one.

1 **Input:** Positive integer $k \in \mathbb{N}$, a set $J \subseteq [n]$ with $|J| \leq k$, $s = \max J$, a target size $m \in \mathbb{N}$, and a string $x \in \{0,1\}^n$ such that $f(x) \geq \max J$, and invariants (1) to (4) hold
2 Let α be such that $n/\alpha^d = \max_{j \in J} |V_j|$ and let β be such that $n/\beta^d = m$;
3 **for** $i = 1, \ldots, d(\log \beta - \log \alpha)/\log k$ **do**
4 ⌊ `ReductionStep`$(k, J, n/(k^i \alpha^d), x)$;
5 **Output:** Sets V_j with $|V_j| \leq m$ for all $j \in J$.

`ReductionStep` is given a set J of at most k indices and a string x with $f(x) \geq \max J$. The goal is to reduce the size of each candidate set V_j, $j \in J$, below a target size m where $m \geq |V_j|/k$ for all $j \in J$. The routine works in phases of several iterations each. Let J be the set of indices of the candidate sets that are still above the target size at the beginning of an iteration. For each $j \in J$, we randomly choose a subset $F_j \subseteq V_j$ of size $|V_j|/k$. We create a new bit-string y' from x by flipping the entries in positions $\bigcup_{j \in J} F_j$. Since the sets V_j, $j \leq s = \max J$, are

pairwise disjoint, we have either $f(y') \geq \max J$ or $f(y') = j - 1$ for some $j \in J$. In the first case, i.e., if $f(y') \geq \max J$, none of the sets V_j was *hit*, and for all $j \in J$ we can remove the subset F_j from the candidate set V_j. We call such queries "*off-trials*". An off-trial reduces the size of all sets V_j, $j \in J$, to a $(1 - 1/k)$th fraction of their original size. If, on the other hand, we have $f(y') = j - 1$ for some $j \in J$, we can replace V_j by set F_j as $\pi(j) \in F_j$ must hold. Since $|F_j| = |V_j|/k \leq m$ by assumption, this set has now been reduced to its target size and we can remove it from J.

We continue in this way until at least half of the indices are removed from J and at least ck off-trials occurred, for some constant c satisfying $(1 - 1/k)^{ck} \leq 1/2$. We then proceed to the next phase. Consider any j that is still in J. The size of V_j was reduced by a factor $(1 - 1/k)$ at least ck times. Thus its size was reduced to at most half its original size. We may thus halve k without destroying the invariant $m \geq |V_j|/k$ for $j \in J$. The effect of halving k is that the relative size of the sets F_j will be doubled for the sets V_j that still take part in the reduction process.

Algorithm 5: A call `ReductionStep`(k, J, m, x) reduces the size of at most k sets V_j, $j \in J$, to a kth fraction of their original size using only $O(k)$ queries.

1 **Input:** Positive integer $k \in \mathbb{N}$, a set $J \subseteq [n]$ with $|J| \leq k$, a target size $m \in \mathbb{N}$ with $|V_j| \leq km$ for all $j \in J$, and a string $x \in \{0,1\}^n$ with $f(x) \geq \max J$. Invariants (1) to (4) hold.
2 **for** $j \in J$ **do if** $|V_j| \leq m$ **then** delete j from J ; // V_j is already small enough
3 **while** $J \neq \emptyset$ **do**
4 $o \leftarrow 0$; // counts the number of off-trials
5 $\ell = |J|$; // $|V_j| \leq km$ for all $j \in J$
6 **repeat**
7 **for** $j \in J$ **do** Uniformly at random choose a subset $F_j \subseteq V_j$ of size $|V_j|/k$;
8 Create y' from x by flipping in x the entries in positions $\bigcup_{j \in J} F_j$ and query $f(y')$;
9 **if** $f(y') \geq \max J$ **then**
10 $o \leftarrow o + 1$; // "off"-trial
11 **for** $j \in J$ **do** $V_j \leftarrow V_j \backslash F_j$;
12 **else**
13 $V_{f(y')+1} \leftarrow F_{f(y')+1}$; // set $V_{f(y')+1}$ is hit
14 **for** $j \in J$ **do if** $j \leq f(y')$ **then** $V_j \leftarrow V_j \backslash F_j$;
15 **for** $j \in J$ **do if** $|V_j| \leq m$ **then** delete j from J;
16 **until** $o \geq c \cdot k$ *and* $|J| \leq \ell/2$ // c is chosen such that $(1 - 1/k)^{ck} \leq 1/2$;
17 $k \leftarrow k/2$;
18 **Output:** Sets V_j with $|V_j| \leq m$ for all $j \in J$.

Lemma 4.4. *Let $k \in \mathbb{N}$ and let $J \subseteq [n]$ be a set of at most k indices with $s = \max J$. Assume that invariants (1) and (4) hold. Let $x \in \{0,1\}^n$ be such that $f(x) \geq \max J$ and let $m \in \mathbb{N}$ be such that $m \geq |V_j|/k$ for all $j \in J$. In expectation it takes $O(k)$ queries until* `ReductionStep`(k, J, m, x) *has reduced the size of V_j to at most m for each $j \in J$.*

Proof. Let c be some constant. We show below that—for a suitable choice of c—after an expected number of at most ck queries both conditions in line 16 are satisfied. Assuming this to hold, we can bound the total expected number of queries until the size of each of the sets V_j

has been reduced to m by

$$\sum_{h=0}^{\log k} ck/2^h < 2ck\,,$$

as desired.

In each iteration of the repeat-loop we either hit an index in J and hence remove it from J or we have an off-trial. The probability of an off-trial is at least $(1-1/k)^k$ since $|J| \le k$ always. Thus the probability of an off-trial is at least $(2e)^{-1}$ and hence the condition $o \ge ck$ holds after an expected number of $O(k)$ iterations.

As long as $|J| \ge \ell/2$, the probability of an off-trial is at most $(1-1/k)^{\ell/2}$ and hence the probability that a set is hit is at least $1-(1-1/k)^{\ell/2}$. Since $\ln(1-1/k) \le -1/k$ we have $(1-1/k)^{\ell/2} = \exp(\ell/2\ln(1-1/k)) \le \exp(-\ell/(2k))$ and hence $1-(1-1/k)^{\ell/2} \ge 1-\exp(-\ell/(2k)) \ge \ell/(2k)$. Thus the expected number of iterations to achieve $\ell/2$ hits is $O(k)$.

If a candidate set V_j is hit in the repeat-loop, its size is reduced to $|V_j|/k$. By assumption, this is bounded by m. If V_j is never hit, its size is reduced at least ck times by a factor $(1-1/k)$. By choice of c, its size at the end of the phase is therefore at most half of its original size. Thus after replacing k by $k/2$ we still have $|V_j|/k \le m$ for $j \in J$. □

It is now easy to determine the complexity of `SizeReduction`.

Corollary 4.5. *Let $k \in \mathbb{N}$, J, and y be as in Lemma 4.4. Let further $d \in \mathbb{N}$ and $\alpha \in \mathbb{R}$ such that $\max_{j\in J}|V_j| = n/\alpha^d$. Let $\beta \in \mathbb{R}$ with $\beta > \alpha$.*

Using at most $d(\log\beta - \log\alpha)/\log k$ calls to Algorithm 5 we can reduce the maximal size $\max_{j\in J}|V_j|$ to n/β^d. The overall expected number of queries needed to achieve this reduction is $O(1)kd(\log\beta-\log\alpha)/\log k$.

Proof. The successive calls can be done as follows. We first call `ReductionStep`$(k, J, n/(k\alpha^d), x)$. By Lemma 4.4 it takes an expected number of $O(k)$ queries until the algorithm terminates. The sets V_j, $j \in J$, now have size at most $n/(k\alpha^d)$. We next call `ReductionStep`$(k, J, n/(k^2\alpha^d), x)$. After the hth such call we are left with sets of size at most $n/(k^h\alpha^d)$. For $h = d(\log\beta - \log\alpha)/\log k$ we have $k^h \ge (y/\alpha)^d$. The total expected number of queries at this point is $O(1)kd(\log\beta-\log\alpha)/\log k$. □

4.4 Proof of Theorem 4.1

It remains to show that the first phase of Algorithm 2 takes $O(n\log\log n)$ queries in expectation.

Theorem 4.6. *Let $q \in n - \Theta(n/\log n)$. Algorithm 2 identifies positions $\pi(1),\dots,\pi(q)$ and the corresponding entries $z_{\pi(1)},\dots,z_{\pi(q)}$ of z in these positions using an expected number of $O(n\log\log n)$ queries.*

We prove Theorem 4.6. The proof of the required probabilistic statements is postponed to Section 4.5. If no failure in any call of *Advance* happened, the expected number of queries is bounded by

$$\begin{aligned}
&\frac{q}{\alpha_t}\left(\frac{\alpha_t\log n}{\log\alpha_t} + \frac{\alpha_t}{\alpha_{t-1}}\left(\frac{\alpha_{t-1}dc(\log\alpha_t-\log\alpha_{t-1})}{\log\alpha_{t-1}}\right.\right.\\
&\qquad\left.\left. + \frac{\alpha_{t-1}}{\alpha_{t-2}}\left(\dots + \frac{\alpha_2}{\alpha_1}\left(\frac{\alpha_1 dc(\log\alpha_2-\log\alpha_1)}{\log\alpha_1} + \alpha_1 d\log\alpha_1\right)\right)\right)\right)\\
\le\;& ndc\left(\frac{\log n}{\log\alpha_t} + \frac{\log\alpha_t}{\log\alpha_{t-1}} + \dots + \frac{\log\alpha_2}{\log\alpha_1} + \log\alpha_1 - t\right), \qquad (1)
\end{aligned}$$

where c is the constant hidden in the $O(1)$-term in Lemma 4.4. To verify this formula, observe that we fill the $(i-1)$st level α_i/α_{i-1} times before level i has reached its capacity of α_i candidate sets. To add α_{i-1} candidate sets from level $i-1$ to level i, we need to reduce their sizes from n/α_{i-1}^d to n/α_i^d. By Corollary 4.5 this requires at most $\alpha_{i-1}dc(\log\alpha_i - \log\alpha_{i-1})/\log\alpha_{i-1}$ queries. The additional $\alpha_1 d\log\alpha_1$ term accounts for the queries needed to move the sets from level 0 to level 1; i.e., for the randomized binary search algorithm through which we initially reduce the sizes of the sets V_i to n/α_1^d—requiring at most $d\log\alpha_1$ queries per call. Finally, the term $\alpha_t \log n/\log\alpha_t$ accounts for the final reduction of the sets V_i to a set containing only one single element (at this stage we shall finally have $V_i = \{\pi(i)\}$). More precisely, this term is $(\alpha_t(\log n - d\log\alpha_t))\,/\log\alpha_t$ but we settle for upper bounding this expression by the term given in the formula.

Next we need to bound the number of queries caused by *failures*. We show that, on average, not too many failures happen. More precisely, we show that the expected number of level-i failures is at most $n^2/((n-q)(\alpha_i^{d-1}-1))$. By Corollary 4.5, each such level-i failure causes an additional number of at most $1 + \alpha_i dc(\log\alpha_{i+1} - \log\alpha_i)/\log\alpha_i$ queries (the 1 counts for the query through which we discover that $\pi(s+1) \in V_1 \cup \ldots \cup V_s$). Thus,

$$\sum_{i=1}^{t} \frac{n^2}{(n-q)(\alpha_i^{d-1}-1)} \left(1 + \frac{\alpha_i dc(\log\alpha_{i+1} - \log\alpha_i)}{\log\alpha_i}\right) \tag{2}$$

bounds the expected number of additional queries caused by failures.

Recall the choice of t and the capacities α_i. We have $\alpha_1 = \log n$ and $\alpha_i = \alpha_{i-1}^2 = (\log n)^{2^{i-1}}$; t is maximal such that $\alpha_t^d \le n$. Then $\alpha_t \ge \sqrt{n}$ and hence $\log\alpha_t = \Omega(\log n)$. The parameter d is any constant that is at least 4. With these parameter settings, formula (1) evaluates to

$$ndc\left(\frac{\log n}{\log\alpha_t} + 2(t-1) + \log\log n - t\right) = O(n\log\log n)$$

and, somewhat wastefully, we can bound formula (2) from above by

$$\frac{n^2 dc}{n-q}\sum_{i=1}^{t}\alpha_i^{-(d-3)} = O(n\log n)\sum_{i=0}^{t-1}\alpha_1^{-(d-3)2^i} < O(n\log n)(\alpha_1^{d-3}-1)^{-1} = O(n)\,,$$

where the first equation is by construction of the values of α_i, the inequality uses the fact that the geometric sum is dominated by the first term, and the last equality stems from our choice $\alpha_1 = \log n$ and $d \ge 4$. This shows, once Corollary 4.9 is proven, that the overall expected number of queries sums to $O(n\log\log n) + O(n) = O(n\log\log n)$. □

4.5 Failures

We derive a bound on the expected number of failures. We do so in two steps. We first bound the worst-case number of calls of *Advance*(ℓ) for any fixed ℓ and then bound the probability that any particular call may fail.

In order to bound the worst-case number of calls of *Advance*(ℓ), we observe that any such call returns a non-empty set and distinct calls return disjoint sets. Thus there can be at most q calls to *Advance*(ℓ) for any ℓ. Before a call to *Advance*(ℓ), we have $\pi(s+1) \notin \bigcup_{j\le s} V_j$, and hence any call increases s by at least 1. This is obvious for *Advance*(1) and holds for $\ell > 1$, since *Advance*(ℓ) immediately calls *Advance*$(\ell-1)$.

We turn to the probabilistic part of the analysis. Key to the failure analysis is the following observation.

Lemma 4.7. *Each V_j has the property that $V_j \setminus \{\pi(j)\}$ is a random subset of $V_j^* \setminus \{\pi(j)\}$ of size $|V_j| - 1$, where V_j^* is as defined in line 15 of Algorithm 2.*

Proof. V_j is initialized to V_j^*; thus the claim is true initially. In `RandBinSearch`, a random subset F of V_j is chosen and V_j is reduced to F (if $\pi(j) \in F$) or to $V_j \setminus F$ (if $\pi(j) \notin F$). In either case, the claim stays true. The same reasoning applies to `ReductionStep`. □

Lemma 4.8. *Let $q \in n - \Theta(n/\log n)$ be the number of indices i for which we determine $\pi(i)$ and $z_{\pi(i)}$ in the first phase. The probability that any particular call of Advance(ℓ) fails is less than $n(n-q)^{-1}(\alpha_\ell^{d-1} - 1)^{-1}$.*

Proof. A failure happens if both x and $y = x \oplus 1_{[n]\setminus\bigcup_{i=1}^{s} V_i}$ have a score of more than s or equal to s in line 25 of Algorithm 2. This is equivalent to $\pi(s+1) \in \bigcup_{i=1}^{s} V_i$.

Let k be the number of indices $i \in [n]$ for which V_i is on the last level; i.e., $k := |\{i \in [n] \mid |V_i| = 1\}|$ is the number of sets V_i which have been reduced to singletons already. Note that these sets satisfy $V_i = \{\pi(i)\}$. Therefore, they cannot contain $\pi(s+1)$ and we do not need to take them into account.

A failure on level ℓ occurs only if $\pi(s+1) \in \bigcup_{j=1}^{s} V_j$ and the size of each candidate set $V_1, \ldots, V_s$ has been reduced already to at most n/α_ℓ^d. There are at most α_j candidate sets on each level $j \geq \ell$. By construction, the size of each candidate set on level j is at most n/α_j^d. By Lemma 4.7, the probability that $\pi(s+1) \in \bigcup_{j=1}^{s} V_j$ is at most

$$\frac{1}{n-k} \sum_{j=\ell}^{t} \frac{n\alpha_j}{\alpha_j^d} = \frac{n}{n-k} \sum_{j=\ell}^{t} \frac{1}{\alpha_j^{d-1}}\,. \tag{3}$$

By definition we have $\alpha_j \geq \alpha_\ell^{(2^{j-\ell})}$ and in particular we have $\alpha_j \geq \alpha_\ell^{j-\ell}$. Therefore expression (3) can be bounded from above by

$$\frac{n}{n-k} \sum_{j=1}^{t-\ell} \left(\frac{1}{\alpha_\ell^{d-1}}\right)^j < \frac{n}{n-k} \left(\alpha_\ell^{d-1} - 1\right)^{-1}\,.$$

□

Since *Advance*(ℓ) is called at most q times we immediately get the following.

Corollary 4.9. *Let $\ell \in [t]$. The expected number of level ℓ failures is less than*

$$qn(n-q)^{-1}(\alpha_\ell^{d-1} - 1)^{-1} \leq n^2(n-q)^{-1} \left(\alpha_\ell^{d-1} - 1\right)^{-1}\,.$$

5 The Lower Bound

We prove a tight lower bound for the randomized query complexity of the HiddenPermutation problem. The lower bound is stated in the following:

Theorem 5.1. *The randomized query complexity of the* HiddenPermutation *problem with n positions is $\Omega(n \log\log n)$.*

To prove a lower bound for randomized query schemes, we appeal to Yao's principle. That is, we first define a hard distribution over the secrets and show that every deterministic query scheme for this hard distribution needs $\Omega(n \log\log n)$ queries in expectation. This part of the proof is done using a potential function argument.

Hard Distribution. Let Π be a permutation drawn uniformly among all the permutations of $[n]$ (in this section, we use capital letters to denote random variables). Given such a permutation, we let our target string Z be the one satisfying $Z_{\Pi(i)} = i(\bmod 2)$ for $i = 1, \ldots, n$. Since Z is uniquely determined by the permutation Π, we will mostly ignore the role of Z in the rest of this section. Finally, we use $F(x)$ to denote the value of the random variable $f_{Z,\Pi}(x)$ for $x \in \{0,1\}^n$. We will also use the notation $a \equiv b$ to mean that $a \equiv b(\bmod 2)$.

Deterministic Query Schemes. By fixing the random coins, a randomized solution with expected t queries implies the existence of a deterministic query scheme with expected t queries over our hard distribution. The rest of this section is devoted to lower bounding t for such a deterministic query scheme.

Recall that a deterministic query scheme is a decision tree T in which each node v is labeled with a string $x_v \in \{0,1\}^n$. Each node has $n+1$ children, numbered from 0 to n, and the ith child is traversed if $F(x_v) = i$. To guarantee correctness, no two inputs can end up in the same leaf.

For a node v in the decision tree T, we define $\max_v$ as the largest value of F seen along the edges from the root to v. Note that $\max_v$ is not a random variable. Also, we define S_v as the subset of inputs (as outlined above) that reach node v.

Update Rules for Candidate Sets. We use a potential function which measures how much "information" the queries asked have revealed about Π. Our goal is to show that the expected increase in the potential function after asking each query is small. Our potential function depends crucially on the candidate sets. The update rules for the candidate sets are slightly more specific than the ones in Section 2 because we now have a fixed connection between the two parts of the secret. We denote the candidate set for $\pi(i)$ at node v with V_i^v. At the root node r, we have $V_i^r = [n]$ for all i. Let v be a node in the tree and let $w_0, \ldots, w_n$ be its children (w_i is traversed when the score i is returned). Let P_0^v (resp. P_1^v) be the set of positions in x_v that contain 0 (resp. 1). Thus, formally, $P_0^v = \{i \mid x_v[i] = 0\}$ and $P_1^v = \{i \mid x_v[i] = 1\}$.[1] The precise definition of candidate sets is as follows:

$$V_i^{w_j} = \begin{cases} V_i^v \cap P^v_{i(\bmod 2)} & \text{if } i \le j\,, \\ V_i^v \cap P^v_{j(\bmod 2)} & \text{if } i = j+1\,, \\ V_i^v & \text{if } i > j+1. \end{cases}$$

As with the upper bound case, the candidate sets have some very useful properties. These properties are slightly different from the ones observed before, due to the fact that some extra information has been announced to the query algorithm. We say that a candidate set V_i^v is *active (at v)* if the following conditions are met: (i) at some ancestor node u of v, we have $F(x_u) = i-1$, (ii) at every ancestor node w of u we have $F(x_w) < i-1$, and (iii) $i < \min\{n/3, \max_v\}$. We call $V^v_{\max_v+1}$ *pseudo-active (at v)*.

For intuition on the requirement $i < n/3$, observe from the following lemma that $V^v_{\max_v+1}$ contains all sets V_i^v for $i \le \max_v$ and $i \equiv \max_v$. At a high level, this means that the distribution of $\Pi(\max_v+1)$ is not independent of $\Pi(i)$ for $i \equiv \max_v$. The bound $i < n/3$, however, forces the dependence to be rather small (there are not too many such sets). This greatly helps in the potential function analysis.

We prove the following lemma with arguments similar to those in the proof of Theorem 2.1.

[1] To prevent our notations from becoming too overloaded, here and in the remainder of the section we write $x = (x[1], \ldots, x[n])$ instead of $x = (x_1, \ldots, x_n)$

Lemma 5.2. *The candidate sets have the following properties:*

(i) Two candidate sets V_i^v and V_j^v with $i < j \leq \max_v$ and $i \not\equiv j$ are disjoint.

(ii) An active candidate set V_j^v is disjoint from any candidate set V_i provided $i < j < \max_v$.

(iii) The candidate set V_i^v, $i \leq \max_v$ is contained in the set $V_{\max_v+1}^v$ if $i \equiv \max_v$ and is disjoint from it if $i \not\equiv \max_v$.

(iv) For two candidate sets V_i^v and V_j^v, $i < j$, if $V_i^v \cap V_j^v \neq \emptyset$ then $V_i^v \subset V_j^v$.

Proof. Let w be the ancestor of v where the function returns score $\max_v$.

To prove (i), observe that in w, one of V_i^w and V_j^w is intersected with P_0^w while the other is intersected with P_1^w and thus they are made disjoint.

To prove (ii), we can assume $i \equiv j$ as otherwise the result follows from the previous case. Let u be the ancestor of v such that $F(x_u) = j - 1$ and that in any ancestor of u, the score returned by the function is smaller than $j - 1$. At u, V_j^u is intersected with $P_{j-1 \bmod 2}^u$ while V_i^v is intersected with $P_{i \bmod 2}^u$. Since $i \equiv j$, it follows that they are again disjoint.

For (iii), the latter part follows as in (i). Consider an ancestor v' of v and let w_j be the jth child of v' that is also an ancestor of v. We use induction and we assume $V_i^{v'} \subset V_{\max_v+1}^{v'}$. If $j < \max_v$, then $V_{\max_v+1}^{v'} = V_{\max_v+1}^{w_j}$ which means $V_i^{w_j} \subset V_{\max_v+1}^{w_j}$. If $j = \max_v$, then $V_{\max_v+1}^{w_j} = V_{\max_v+1}^{v'} \cap P_{\max_v(\bmod 2)}^{v'}$ and notice that in this case also $V_i^{w_j} = V_i^{v'} \cap P_{i(\bmod 2)}^{v'}$ which still implies $V_i^{w_j} \subset V_{\max_v+1}^{w_j}$.

To prove (iv), first observe that the statement is trivial if $i \not\equiv j$. Also, if the function returns score $j - 1$ at any ancestor of v, then by the same argument used in (ii) it is possible to show that $V_i^v \cap V_j^v = \emptyset$. Thus assume $i \equiv j$ and the function never returns value $j - 1$. In this case, it is easy to see that an inductive argument similar to (iii) proves that $V_i^{v'} \subset V_j^{v'}$ for every ancestor v' of v. □

Corollary 5.3. *Every two distinct active candidate sets V_i^v and V_j^v are disjoint.*

Remember that the permutation Π was chosen uniformly and randomly. Soon, we shall see that this fact combined with the above properties imply that $\Pi(i)$ is uniformly distributed in V_i^v, when V_i^v is active. The following lemma is needed to prove this.

Lemma 5.4. *Consider a candidate set V_i^v and let $i_1 < \cdots < i_k < i$ be the indices of candidate sets that are subsets of V_i^v. Let $\sigma := (\sigma_1, \cdots, \sigma_i)$ be a sequence without repetition from $[n]$ and let $\sigma' := (\sigma_1, \cdots, \sigma_{i-1})$. Let n_σ and $n_{\sigma'}$ be the number of permutations in S_v that have σ and σ' as a prefix, respectively. If $n_\sigma > 0$, then $n_{\sigma'} = (|V_i^v| - k)n_\sigma$.*

Proof. Consider a permutation $\pi \in S_v$ that has σ as a prefix. This implies $\pi(i) \in V_i^v$. For an element $s \in V_i^v$, $s \neq i_j$, $1 \leq j \leq k$, let π_s be the permutation obtained from π by placing s at position i and placing $\pi(i)$ where s used to be. Since $s \neq i_j$, $1 \leq j \leq k$, it follows that π_s has σ' as prefix and since $s \in V_i^v$ it follows that $\pi_s \in S_v$. It is easy to see that for every permutation in S_v that has σ as a prefix we will create $|V_i^v| - k$ different permutations that have σ' as a prefix and all these permutations will be distinct. Thus, $n_{\sigma'} = (|V_i^v| - k)n_\sigma$. □

Corollary 5.5. *Consider a candidate set V_i^v and let $i_1 < \cdots < i_k < i$ be the indices of candidate sets that are subsets of V_i^v. Let $\sigma' := (\sigma_1, \cdots, \sigma_{i-1})$ be a sequence without repetition from $[n]$ and let $\sigma_1 := (\sigma_1, \cdots, \sigma_{i-1}, s_1)$ and $\sigma_2 := (\sigma_1, \cdots, \sigma_{i-1}, s_2)$ in which $s_1, s_2 \in V_i^v$. Let n_{σ_1} and n_{σ_2} be the number of permutations in S_v that have σ_1 and σ_2 as a prefix, respectively. If $n_{\sigma_1}, n_{\sigma_2} > 0$, then $n_{\sigma_1} = n_{\sigma_2}$.*

Proof. Consider a sequence $s_1, \cdots, s_i$ without repetition from $[n]$ such that $s_j \in V_j^v$, $1 \leq j \leq i$. By the previous lemma $\Pr[\Pi(1) = s_1 \wedge \cdots \wedge \Pi(i-1) = s_{i-1} \wedge \Pi(i) = s_i] = \Pr[\Pi(1) = s_1 \wedge \cdots \wedge \Pi(i-1) = s_{i-1}] \cdot (1/|V_i^v|)$. □

Corollary 5.6. *If V_i^v is active, then we have:*

(i) $\Pi(i)$ *is independent of* $\Pi(1), \cdots, \Pi(i-1)$.

(ii) $\Pi(i)$ *is uniformly distributed in* V_i^v.

5.1 The Potential Function

We define the potential of an active candidate set V_i^v as $\log\log(2n/|V_i^v|)$. This is inspired by the upper bound: a potential increase of 1 corresponds to a candidate set advancing one level in the upper bound context (in the beginning, a set V_i^v has size n and thus its potential is 0 while at the end its potential is $\Theta(\log\log n)$. With each level, the quantity n divided by the size of V_i is squared). We define the potential at a node v as

$$\varphi(v) = \log\log \frac{2n}{|V^v_{\max_v+1}| - \mathrm{Con}_v} + \sum_{j \in A_v} \log\log \frac{2n}{|V_j^v|},$$

in which A_v is the set of indices of active candidate sets at v and Con_v is the number of candidate sets contained inside $V^v_{\max_v+1}$. Note that from Lemma 5.2, it follows that $\mathrm{Con}_v = \lfloor \max_v/2 \rfloor$.

The intuition for including the term Con_v is the same as our requirement $i < n/3$ in the definition of active candidate sets, namely that once Con_v approaches $|V^v_{\max_v+1}|$, the distribution of $\Pi(\max_v+1)$ starts depending heavily on the candidate sets V_i^v for $i \le \max_v$ and $i \equiv \max_v$. Thus we have in some sense determined $\Pi(\max_v+1)$ already when $|V^v_{\max_v+1}|$ approaches Con_v. Therefore, we have to take this into account in the potential function since otherwise changing $V^v_{\max_v+1}$ from being pseudo-active to being active could give a huge potential increase.

The following is the main lemma that we wish to prove; it tells us that the expected increase of the potential function after each query is constant. The proof of this lemma is the main part of the proof of Theorem 5.1.

Lemma 5.7. *Let v be a node in T and let $\boldsymbol{i}_v$ be the random variable giving the value of $F(x_v)$ when $\Pi \in S_v$ and 0 otherwise. Also let $w_0, \ldots, w_n$ denote the children of v, where w_j is the child reached when $F(x_v) = j$. Then, $\mathbb{E}[\varphi(w_{\boldsymbol{i}_v}) - \varphi(v) \mid \Pi \in S_v] = O(1)$.*

Note that we have $\mathbb{E}[\varphi(w_{\mathbf{i}_v}) - \varphi(v) \mid \Pi \in S_v] = \sum_{a=0}^{n} \Pr[F(x_v) = a \mid \Pi \in S_v](\varphi(w_a) - \varphi(v))$. We consider two main cases: $F(x_v) \le \max_v$ and $F(x_v) > \max_v$. In the first case, the maximum score will not increase in w_a which means w_a will have the same set of active candidate sets. In the second case, the pseudo-active candidate set $V^v_{\max_v+1}$ will turn into an active set $V^{w_a}_{\max_v+1}$ at w_a and w_a will have a new pseudo-active set. While this second case looks more complicated, it is in fact the less critical part of the analysis. This is because the probability of suddenly increasing the score by a large α is extremely small (we will show that it is roughly $2^{-\Omega(\alpha)}$) which subsumes any significant potential increase for values of $a > \max_v$.

Let $a_1, \ldots, a_{|A_v|}$ be the indices of active candidate sets at v sorted in increasing order. We also define $a_{|A_v|+1} = \max_v + 1$. For a candidate set V_i^v, and a Boolean $b \in \{0,1\}$, let $V_i^v(b) = \{j \in V_i^v \mid x_v[j] = b\}$. Clearly, $|V_i^v(0)| + |V_i^v(1)| = |V_i^v|$. For even a_i, $1 \le i \le |A_v|$, let

$$\varepsilon_i = |V_i^v(1)|/|V_i^v|,$$

and for odd i, let

$$\varepsilon_i = |V_i^v(0)|/|V_i^v|.$$

Thus ε_i is the fraction of locations in V_i^v that contain values that does not match $Z_{\Pi(i)}$. Also, we define

$$\varepsilon_i' := \Pr[a_i \le F(x_v) < a_{i+1} - 1 | \Pi \in S_v \wedge F(x_v) \ge a_i].$$

Note that $\varepsilon'_i = 0$ if $a_{i+1} = a_i + 1$.

With these definitions, it is clear that we have

$$\begin{aligned} |V_{a_i}^{w_j}| &= (1-\varepsilon_i)|V_{a_i}^v| \quad \text{for} \quad a_i \leq j. \\ |V_{a_i}^{w_j}| &= \varepsilon_i |V_{a_i}^v| \quad \text{for} \quad a_i = j+1. \\ |V_{a_i}^{w_j}| &= |V_{a_i}^v| \quad \text{for} \quad a_i > j+1. \end{aligned}$$

The important fact is that we can also bound other probabilities using the values of the ε_i and ε'_i: We can show that (details follow)

$$\Pr[F(x_v) = a_i - 1 | \Pi \in S_v] \leq \varepsilon_i \Pi_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j) \tag{4}$$

and

$$\Pr[a_i \leq F(x_v) < a_{i+1} - 1 | \Pi \in S_v] \leq \varepsilon'_i (1-\varepsilon_i)\Pi_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j).$$

Thus we have bounds on the changes in the size of the active candidate sets in terms of the values ε_i. The probability of making the various changes is also determined from the values ε_i and ε'_i and finally the potential function is defined in terms of the sizes of these active candidate sets. Thus proving Lemma 5.7 reduces to proving an inequality showing that any possible choice of the values for the ε_i and ε'_i provide only little expected increase in potential.

Proof Sketch. Since the full calculations are rather lengthy, in the following paragraphs we will provide the heart of the analysis by making simplifying assumptions that side-step some uninteresting technical difficulties that are needed for the full proof. We assume that all values ε'_i are 0 or in other words, $a_i = i$ for all $i \leq \max_v$. Also, we ignore the term in $\varphi(v)$ involving Con_v and consider only the case where the score returned is no larger than $\max_v$ and $\max_v = n/4$.

Thus the expected increase in potential provided by the cases where the score does not increase is bounded by

$$\sum_{j \leq n/4} \Pr[F(x_v) = j \mid \Pi \in S_v](\varphi(w_j) - \varphi(v)) \leq \sum_{j \leq n/4} \varepsilon_j (\varphi(w_j) - \varphi(v)).$$

Also, we get that when a score of j is returned, we update

$$\begin{aligned} |V_i^{w_j}| &= (1 - \Theta(1/n))|V_i^v| \quad \text{for} \quad i \leq j. \\ |V_i^{w_j}| &= \Theta(|V_i^v|/n) \quad \text{for} \quad i = j+1. \\ |V_i^{w_j}| &= |V_i^v| \quad \text{for} \quad i > j+1. \end{aligned}$$

Examining $\varphi(w_j) - \varphi(v)$ and the changes to the candidate sets, we get that there is one candidate set whose size decreases by a factor ε_j, and there are j sets that change by a factor $1 - \varepsilon_j$. Here we only consider the potential change caused by the sets changing by a factor of ε_j. This change is bounded by

$$\sum_{j \leq n/4} \varepsilon_j \log\left(\frac{\log(2n/(\varepsilon_j|V_j^v|)}{\log(2n/|V_j^v|)}\right) = \sum_{j \leq n/4} \varepsilon_j \log\left(1 + \frac{\log(1/\varepsilon_j)}{\log(2n/|V_j^v|)}\right) \leq \sum_{j \leq n/4} \varepsilon_j \log \frac{\log(1/\varepsilon_j)}{\log(2n/|V_j^v|)},$$

where we used $\log(1+x) \leq x$ for $x > 0$ for the last inequality. The function $\varepsilon_j \to (1/\varepsilon_j)^{\varepsilon_j}$ is decreasing for $0 < \varepsilon \leq 1$. Also, if $\varepsilon > 0$ then $\varepsilon \geq 1/n$ by definition of ε. We can therefore upper bound the sum by setting $\varepsilon_j = 4/n$ for all j. To continue these calculations, we use Lemma 5.2 to conclude that the active candidate sets are disjoint and hence the sum of their

sizes is bounded by n. We now divide the sum into summations over indices where $|V_j^v|$ is in the range $[2^i : 2^{i+1}]$ (there are at most $n/2^i$ such indices):

$$\sum_{j \le n/4} \Theta\left(\frac{1}{n}\right) \frac{\log(n/4)}{\log(2n/|V_j^v|)} \le \Theta\left(\sum_{i=0}^{\log n - 1} \frac{\log(n/4)}{2^i \log(n/2^i)}\right).$$

Now the sum over the terms where $i > \log\log n$ is clearly bounded by a constant since the 2^i in the denominator cancels the $\log(n/4)$ term and we get a geometric series of this form. For $i < \log\log n$, we have $\log(n/4)/\log(n/2^i) = O(1)$ and we again have a geometric series summing to $O(1)$.

The full proof is very similar in spirit to the above, just significantly more involved due to the unknown values ε_i and ε_i'. The complete proof is given in the next section.

5.2 Formal Proof of Lemma 5.7

As we did in the proof sketch we write

$$\mathbb{E}[\varphi(w_{\mathbf{i}_v}) - \varphi(v) \mid \Pi \in S_v] = \sum_{a=0}^{n} \Pr[F(x_v) = a \mid \Pi \in S_v](\varphi(w_a) - \varphi(v)).$$

As discussed, we divide the above summation into two parts: one for $a \le \max_v$ and another for $a > \max_v$:

$$\mathbb{E}[\varphi(w_{\mathbf{i}_v}) - \varphi(v) \mid \Pi \in S_v] =$$

$$\sum_{a=0}^{\max_v} \Pr[F(x_v) = a \mid \Pi \in S_v](\varphi(w_a) - \varphi(v)) + \tag{5}$$

$$\sum_{a=1}^{n/3-\max_v} \Pr[F(x_v) = \max_v + a \mid \Pi \in S_v](\varphi(w_a) - \varphi(v)). \tag{6}$$

To bound the above two summations, it is clear that we need to handle $\Pr[F(x_v) = a | \Pi \in S_v]$. In the next section, we will prove lemmas that will do this.

Bounding Probabilities Let $a_1, \ldots, a_{|A_v|}$ be the indices of active candidate sets at v sorted in increasing order. We also define $a_{|A_v|+1} = \max_v + 1$. For a candidate set V_i^v, and a Boolean $b \in \{0, 1\}$, let $V_i^v(b) = \{j \in V_i^v \mid x_v[j] = b\}$. Clearly, $|V_i^v(0)| + |V_i^v(1)| = |V_i^v|$. For even a_i, $1 \le i \le |A_v|$, let $\varepsilon_i = |V_i^v(1)|/|V_i^v|$ and for odd i let $\varepsilon_i = |V_i^v(0)|/|V_i^v|$. This definition might seem strange but is inspired by the following observation.

Lemma 5.8. *For $i \le |A_v|$, $\Pr[F(x_v) = a_i - 1 | \Pi \in S_v \wedge F(x_v) > a_i - 2] = \varepsilon_i$.*

Proof. Note that $F(x_v) = a_i - 1$ happens if and only if $F(x_v) > a_i - 2$ and $x_v[\Pi(a_i)] \not\equiv a_i$. Since $V_{a_i}^v$ is an active candidate set, the lemma follows from Corollary 5.6 and the definition of ε_i. □

Let $\varepsilon_i' := \Pr[a_i \le F(x_v) < a_{i+1} - 1 | \Pi \in S_v \wedge F(x_v) \ge a_i]$. Note that $\varepsilon_i' = 0$ if $a_{i+1} = a_i + 1$.

Lemma 5.9. *For $i \le |A_v|$ we have $|V_{a_i}^{w_j}| = |V_{a_i}^v|$ for $j < a_i - 1$, $|V_{a_i}^{w_j}| = \varepsilon_i |V_{a_i}^v|$ for $j = a_i - 1$, and $|V_{a_i}^{w_j}| = (1 - \varepsilon_i)|V_{a_i}^v|$ for $j > a_i - 1$. Also,*

$$\Pr[F(x_v) = a_i - 1 | \Pi \in S_v] = \varepsilon_i \Pi_{j=1}^{i-1}(1 - \varepsilon_j)(1 - \varepsilon_j'), \tag{7}$$

$$\Pr[a_i \le F(x_v) < a_{i+1} - 1 | \Pi \in S_v] = \varepsilon_i'(1 - \varepsilon_i)\Pi_{j=1}^{i-1}(1 - \varepsilon_j)(1 - \varepsilon_j'), \tag{8}$$

Proof. Using Lemma 5.8, it is verified that

$$\begin{aligned}\Pr[F(x_v) > a_i - 1 | \Pi \in S_v] &= \Pr[F(x_v) > a_i - 2 \wedge F(x_v) \neq a_i - 1 | \Pi \in S_v] \\ &= \Pr[F(x_v) \neq a_i - 1 | F(x_v) > a_i - 2 \wedge \Pi \in S_v] \cdot \\ \Pr[F(x_v) > a_i - 2 | \Pi \in S_v] &= (1 - \varepsilon_i) \Pr[F(x_v) > a_i - 2 | \Pi \in S_v].\end{aligned}$$

Similarly, using the definition of ε'_i we can see that

$$\begin{aligned}&\Pr[F(x_v) > a_i - 2 | \Pi \in S_v] \\ &= \Pr[F(x_v) \notin \{a_{i-1}, \ldots, a_i - 2\} \wedge F(x_v) > a_{i-1} - 1 | \Pi \in S_v] \\ &= \Pr[F(x_v) \notin \{a_{i-1}, \ldots, a_i - 2\} \, | F(x_v) > a_{i-1} - 1 \wedge \Pi \in S_v] \Pr[F(x_v) > a_{i-1} - 1 | \Pi \in S_v] \\ &= (1 - \varepsilon'_{i-1}) \Pr[F(x_v) > a_{i-1} - 1 | \Pi \in S_v].\end{aligned}$$

Using these equalities, we find that

$$\Pr[F(x_v) > a_i - 1 | \Pi \in S_v] = (1 - \varepsilon_i) \Pi_{j=1}^{i-1} (1 - \varepsilon_j)(1 - \varepsilon'_j)$$

and

$$\Pr[F(x_v) > a_i - 2 | \Pi \in S_v] = \Pi_{j=1}^{i-1} (1 - \varepsilon_j)(1 - \varepsilon'_j).$$

Equalities (7) and (8) follow from combining these bounds with Lemma 5.8. The rest of the lemma follows directly from the definition of ε_i and the candidate sets. □

Lemma 5.10. *Let $b \in \{0, 1\}$ be such that $b \equiv \max_v$ and let $k := |V^v_{\max_v + 1}(b)|$. Then,*

$$\Pr[F(x_v) = \max_v | \Pi \in S_v] = \frac{k - \mathrm{Con}_v}{|V^v_{\max_v + 1}| - \mathrm{Con}_v} \prod_{i=1}^{|A_v|} (1 - \varepsilon_i)(1 - \varepsilon'_i).$$

Proof. Conditioned on $F(x_v) > \max_v - 1$, $F(x_v)$ will be equal to $\max_v$ if $x_v[\Pi(\max_v + 1)] = b$. By definition, the number of positions in $V^v_{\max_v + 1}$ that satisfy this is k. However, $V^v_{\max_v + 1}$ contains Con_v candidate sets but since $V^v_{\max_v + 1}$ can only contain a candidate set V^v_i if $i \equiv \max_v$ (by Lemma 5.2), it follows from Lemma 5.4 that $\Pr[F(x_v) = \max_v | \Pi \in S_v \wedge F(x_v) > \max_v - 1] = (k - \mathrm{Con}_v)/(|V^v_{\max_v + 1}| - \mathrm{Con}_v)$. The lemma then follows from the previous lemma. □

Lemma 5.11. *Let $b \in \{0, 1\}$ be such that $b \equiv \max_v$ and let $k := |V^v_{\max_v + 1}(b)|$. Then,*

$$\Pr[F(x_v) > \max_v | \Pi \in S_v] \leq \frac{|V^v_{\max_v + 1}| - k}{|V^v_{\max_v + 1}| - \mathrm{Con}_v} \prod_{i=1}^{|A_v|} (1 - \varepsilon_i)(1 - \varepsilon'_i).$$

Proof. From the previous lemma we have that $\Pr[F(x_v) = \max_v | \Pi \in S_v \wedge F(x_v) > \max_v - 1] = (k - \mathrm{Con}_v)/(|V^v_{\max_v + 1}| - \mathrm{Con}_v)$. Thus,

$$\Pr[F(x_v) > \max_v | \Pi \in S_v \wedge F(x_v) > \max_v - 1] = \frac{|V^v_{\max_v + 1}| - k}{|V^v_{\max_v + 1}| - \mathrm{Con}_v}.$$

The claim now follows from Lemma 5.9. □

Remember that P^v_0 (resp. P^v_1) are the set of positions in x_v that contain 0 (resp. 1).

Lemma 5.12. *Let $x_0 = |P^v_0|$ and $x_1 = |P^v_1|$. Let b_i be a Boolean such that $b_i \equiv i$. For $a \geq \max_v + 2$*

$$\Pr[F(x_v) = a \mid \Pi \in S_v] \leq \left(\prod_{i=\max_v + 2}^{a} \frac{x_{b_i} - \lfloor i/2 \rfloor}{n - i + 1} \right) \left(1 - \frac{x_{b_{a+1}} - \lfloor (a+1)/2 \rfloor}{n - a} \right).$$

Proof. Notice that we have $V_i^v = [n]$ for $i \geq \max_v + 2$ which means $i-1$ is the number of candidates sets V_j^v contained in V_i^v and among those $\lfloor i/2 \rfloor$ are such that $i \equiv j$. Consider a particular prefix $\sigma = (\sigma_1, \cdots, \sigma_{i-1})$ such that there exists a permutation $\pi \in S_v$ that has σ as a prefix. This implies that $\sigma_j \in P_{b_j}^v$. Thus, it follows that there are $x_{b_i} - \lfloor i/2 \rfloor$ elements $s \in P_{b_i}^v$ such that the sequences $(\sigma_1, \cdots, \sigma_{i-1}, s)$ can be the prefix of a permutation in S_v. Thus by Corollary 5.5, and for $i \geq \max_v + 2$,

$$\Pr[F(x_v) = i-1 | \Pi \in S_v \wedge F(x_v) \geq i-1] = 1 - \frac{x_{b_i} - \lfloor i/2 \rfloor}{n-i+1}$$

and

$$\Pr[F(x_v) \geq i | \Pi \in S_v \wedge F(x_v) \geq i-1] = \frac{x_{b_i} - \lfloor i/2 \rfloor}{n-i+1}.$$

□

Corollary 5.13. *For* $\max_v + 1 \leq a \leq n/3$ *we have*

$$\Pr[F(x_v) = a \mid \Pi \in S_v] = 2^{-\Omega(a - \max_v)} \cdot \left(1 - \frac{x_{b_{a+1}} - \lfloor (a+1)/2 \rfloor}{n-a}\right).$$

Proof. Since $x_{b_i} + x_{b_{i+1}} = n$, it follows that

$$\left(\frac{x_{b_i} - \lfloor i/2 \rfloor}{n-i+1}\right)\left(\frac{x_{b_{i+1}} - \lfloor i/2 \rfloor}{n-i+2}\right) \leq \left(\frac{x_{b_i} - \lfloor i/2 \rfloor}{n-i+1}\right)\left(\frac{x_{b_{i+1}} - \lfloor i/2 \rfloor}{n-i+1}\right) \leq \frac{1}{2}.$$

□

Now we analyze the potential function.

Bounding (5) We have,

$$\varphi(w_a) - \varphi(v) = \log \frac{\log \frac{2n}{|V^{w_a}_{\max_{w_a}+1}| - \mathrm{Con}_{w_a}}}{\log \frac{2n}{|V^{v}_{\max_v+1}| - \mathrm{Con}_v}} + \sum_{j \in A_v} \log \frac{\log \frac{2n}{|V_j^{w_a}|}}{\log \frac{2n}{|V_j^v|}}. \tag{9}$$

When $a \leq \max_v$ we have, $\max_v = \max_{w_a}$ and $\mathrm{Con}_v = \mathrm{Con}_{w_a}$. For $a < \max_v$, we also have $V^{w_a}_{\max_v+1} = V^v_{\max_v+1}$. It is clear from (9) that for $a_i \leq a < a_{i+1} - 1$, all the values of $\varphi(w_a) - \varphi(v)$ will be equal. Thus, (5) is equal to

$$\sum_{i=1}^{|A_v|} \Pr[F(x_v) = a_i - 1 | \Pi \in S_v](\varphi(w_{a_i - 1}) - \varphi(v)) + \tag{10}$$

$$\sum_{i=1}^{|A_v|} \Pr[a_i \leq F(x_v) < a_{i+1} - 1 | \Pi \in S_v](\varphi(w_{a_i}) - \varphi(v)) + \tag{11}$$

$$\Pr[F(x_v) = \max_v | \Pi \in S_v](\varphi(w_{\max_v}) - \varphi(v)). \tag{12}$$

Analyzing (10) We write (10) using (9) and Lemma 5.9(7). Using inequalities, $1-x \le e^{-x}$, for $0 \le x \le 1$, $\log(1+x) \le x$ for $x \ge 0$, and $\sum_{1\le i\le k} y_i \log 1/y_i \le Y \log(k/Y)$ for $y_i \ge 0$ and $Y = \sum_{1\le i\le k} y_i$, we obtain that (10) equals

$$
\begin{aligned}
&= \sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j)(1-\varepsilon'_j) \left(\sum_{j=1}^{i} \log \frac{\log \frac{2n}{|V_{a_j}^{w_{a_i-1}}|}}{\log \frac{2n}{|V_{a_j}^v|}} \right) \\
&= \sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j)(1-\varepsilon'_j) \left(\log \frac{\log \frac{2n}{|V_{a_i}^{w_{a_i-1}}|}}{\log \frac{2n}{|V_{a_i}^v|}} + \sum_{j=1}^{i-1} \log \frac{\log \frac{2n}{|V_{a_j}^{w_{a_i-1}}|}}{\log \frac{2n}{|V_{a_j}^v|}} \right) \\
&= \sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j)(1-\varepsilon'_j) \left(\log \frac{\log \frac{2n}{|V_{a_i}^v|} + \log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V_{a_i}^v|}} + \sum_{j=1}^{i-1} \log \frac{\log \frac{2n}{|V_{a_j}^v|} + \log \frac{1}{1-\varepsilon_j}}{\log \frac{2n}{|V_{a_j}^v|}} \right) \\
&= \sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j)(1-\varepsilon'_j) \left(\log \left(1 + \frac{\log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V_{a_i}^v|}} \right) + \sum_{j=1}^{i-1} \log \left(1 + \frac{\log \frac{1}{1-\varepsilon_j}}{\log \frac{2n}{|V_{a_j}^v|}} \right) \right) \\
&\le \sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j) \left(\log \left(1 + \frac{\log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V_{a_i}^v|}} \right) + \sum_{j=1}^{i-1} \log \left(1 + \log \frac{1}{1-\varepsilon_j} \right) \right) \\
&\le \sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j) \left(\log \left(1 + \frac{\log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V_{a_i}^v|}} \right) + \sum_{j=1}^{i-1} \log \frac{1}{1-\varepsilon_j} \right) \\
&= \sum_{i=1}^{|A_v|} \varepsilon_i \log \left(1 + \frac{\log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V_{a_i}^v|}} \right) \prod_{j=1}^{i-1} (1-\varepsilon_j) + \qquad (13) \\
&\quad \sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j) \left(\sum_{j=1}^{i-1} \log \frac{1}{1-\varepsilon_j} \right). \qquad (14)
\end{aligned}
$$

To bound (13), we use the fact that any two active candidate sets are disjoint. We break the summation into smaller chunks. Observe that $\prod_{j=1}^{i-1}(1-\varepsilon_j) \le e^{-\sum_{j=1}^{i-1}\varepsilon_j}$. Thus, let J_t, $t \ge 0$, be the set of indices such that for each $i \in J_t$ we have $2^t - 1 \le \sum_{j=1}^{i-1} \varepsilon_j < 2^{t+1}$. Now define $J_{t,k} = \left\{ i \in J_t \mid n/2^{k+1} \le |V_{a_i}^v| \le n/2^k \right\}$, for $0 \le k \le \log n$ and let $s_{t,k} = \sum_{i\in J_{t,k}} \varepsilon_i$. Observe that by the disjointness of two active candidate sets, $|J_{t,k}| \le 2^{k+1}$. The sum (13) is thus equal

to

$$\sum_{t=0}^{\log n}\sum_{k=1}^{\log n}\sum_{i\in J_{t,k}}\varepsilon_i\log\left(1+\frac{\log\frac{1}{\varepsilon_i}}{\log\frac{2n}{|V_{a_i}^v|}}\right)\prod_{j=1}^{i-1}(1-\varepsilon_j)\leq$$

$$\sum_{t=0}^{n}\sum_{k=1}^{\log n}\sum_{i\in J_{t,k}}\varepsilon_i\log\left(1+\frac{\log\frac{1}{\varepsilon_i}}{k}\right)e^{-\sum_{j=1}^{i-1}\varepsilon_j}\leq$$

$$\sum_{t=0}^{n}\sum_{k=1}^{\log n}\sum_{i\in J_{t,k}}\varepsilon_i\frac{\log\frac{1}{\varepsilon_i}}{k}e^{-2^t+1}\leq\sum_{t=0}^{n}\sum_{k=1}^{\log n}\frac{s_{t,k}\log\frac{|J_{t,k}|}{s_{t,k}}}{k}e^{-2^t+1}\leq$$

$$\sum_{t=0}^{n}\sum_{k=1}^{\log n}\frac{s_{t,k}(k+1)+s_{t,k}\log\frac{1}{s_{t,k}}}{k}e^{-2^t+1}\leq$$

$$\sum_{t=0}^{n}2^{t+2}e^{-2^t+1}+\sum_{t=0}^{n}\sum_{k=1}^{\log n}\frac{s_{t,k}\log\frac{1}{s_{t,k}}}{k}e^{-2^t+1}\leq$$

$$O(1)+\sum_{t=0}^{n}\sum_{r=1}^{\log\log n}\sum_{k=2^{r-1}}^{2^r}\frac{s_{t,k}\log\frac{1}{s_{t,k}}}{2^{r-1}}e^{-2^t+1}.$$

Now define $S_{t,r}=\sum_{2^{r-1}\leq k<2^r}s_{t,k}$. Remember that $\sum_{r=1}^{\log\log n}S_{t,r}<2^{t+1}$. (13) is thus at most

$$O(1)+\sum_{t=0}^{n}\sum_{r=1}^{\log\log n}\frac{S_{t,r}\log\frac{2^{r-1}}{S_{t,r}}}{2^{r-1}}e^{-2^t+1}=$$

$$O(1)+\sum_{t=0}^{n}\sum_{r=1}^{\log\log n}\frac{S_{t,r}(r-1)+S_{t,r}\log\frac{1}{S_{t,r}}}{2^{r-1}}e^{-2^t+1}\leq$$

$$O(1)+\sum_{t=0}^{n}\sum_{r=1}^{\log\log n}\frac{2^{t+1}(r-1)}{2^{r-1}}e^{-2^t+1}+\sum_{t=0}^{n}\sum_{r=1}^{\log\log n}\frac{1}{2^{r-1}}e^{-2^t+1}=O(1).$$

To bound (14), define J_t as before and let $p_i=\prod_{j=1}^{i-1}1/(1-\varepsilon_j)$. Observe that the function $\log(1/(1-x))$ is a convex function which means if $s_i:=\sum_{j=1}^{i-1}\varepsilon_j$ is fixed, then $\prod_{j=1}^{i-1}1/(1-\varepsilon_j)$ is minimized when $\varepsilon_j=s_i/(i-1)$. Thus,

$$p_i\geq\left(\frac{1}{1-\frac{s_i}{i-1}}\right)^{i-1}\geq\left(1+\frac{s_i}{i-1}\right)^{i-1}\geq 1+\binom{i-1}{j}\left(\frac{s_i}{i-1}\right)^j$$

in which j can be chosen to be any integer between 0 and $i-1$. We pick $j:=\max\{\lfloor s_i/8\rfloor,1\}$. Since $i\geq s_i$, we get for $i\in J_t$,

$$p_i\geq 1+\frac{\left(\frac{i-1}{2}\right)^j}{j^j}\left(\frac{s_i}{i-1}\right)^j\geq\ 1+\left(\frac{s_i}{2j}\right)^j\geq 2^{s_i/8}\geq 2^{2^{t-4}}.$$

Thus, we can write (14) as

$$\sum_{i=1}^{|A_v|}\varepsilon_i\frac{\log p_i}{p_i}=\sum_{t=0}^{\log n}\sum_{i\in J_t}\varepsilon_i\frac{\log p_i}{p_i}=\sum_{t=0}^{\log n}\sum_{i\in J_t}O\left(\frac{\varepsilon_i 2^t}{2^{2^{t-4}}}\right)\leq\sum_{t=0}^{\log n}O\left(\frac{2^{2t+1}}{2^{2^{t-4}}}\right)=O(1).$$

Analyzing (11) The analysis of this equation is very similar to (10). We can write (11) using (9) and (8). We also use the same technique as in analyzing (14). We find that the sum (11) equals

$$\sum_{i=1}^{|A_v|}\varepsilon'_i(1-\varepsilon_i)\prod_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j)\left(\sum_{j=1}^{i}\log\frac{\log\frac{2n}{|V^{w_{a_i}}_{a_j}|}}{\log\frac{2n}{|V^v_{a_j}|}}\right)=$$
$$\sum_{i=1}^{|A_v|}\varepsilon'_i(1-\varepsilon_i)\prod_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j)\left(\sum_{j=1}^{i}\log\frac{\log\frac{2n}{|V^v_{a_j}|}+\log\frac{1}{1-\varepsilon_j}}{\log\frac{2n}{|V^v_{a_j}|}}\right)=$$
$$\sum_{i=1}^{|A_v|}\varepsilon'_i(1-\varepsilon_i)\prod_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j)\left(\sum_{j=1}^{i}\log\left(1+\frac{\log\frac{1}{1-\varepsilon_j}}{\log\frac{2n}{|V^v_{a_j}|}}\right)\right)\leq$$
$$\sum_{i=1}^{|A_v|}\varepsilon'_i(1-\varepsilon_i)\prod_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j)\left(\sum_{j=1}^{i}\log\frac{1}{1-\varepsilon_i}\right).$$

Let $s_i := \sum_{j=1}^{i}\varepsilon_j$, and $s'_i := \sum_{j=1}^{i-1}\varepsilon'_j$. Similarly to the previous case, let J_t, $t \geq 0$, be the set of indices such that for each $i \in J_t$ we have $2^t - 1 \leq s_i + s'_i \leq 2^{t+1}$. Also define $p_i = \prod_{j=1}^{i} 1/(1-\varepsilon_j)$ and $p'_i = \prod_{j=1}^{i-1} 1/(1-\varepsilon'_j)$. Using the previous techniques we can get that $p_i \geq 2^{s_i/8}$ and $p'_i \geq 2^{s'_i/8}$. We get that (11) is at most

$$\leq \sum_{t=0}^{\log n}\sum_{i\in J_t}\varepsilon'_i\frac{\log p_i}{p_i p'_i} \leq \sum_{t=0}^{\log n}\sum_{i\in J_t}\varepsilon'_i\frac{s_i}{8\cdot 2^{s_i/8}2^{s'_i/8}}$$
$$\leq \sum_{t=0}^{\log n}\sum_{i\in J_t}\varepsilon'_i\frac{2^{t+1}}{8\cdot 2^{(2^t-1)/8}} \leq \sum_{t=0}^{\log n}\frac{2^{2t+2}}{8\cdot 2^{(2^t-1)/8}} = O(1).$$

Analyzing (12) Let $b \in \{0,1\}$ be such that $b \equiv \max_v$ and let $k := |V^v_{\max_v+1}(b)|$. We use Lemma 5.10. Observe that we still have $\mathrm{Con}_v = \mathrm{Con}_{w_{\max_v}}$. Thus we can bound (12) from above by

$$\frac{(k-\mathrm{Con}_v)}{|V^v_{\max_v+1}|-\mathrm{Con}_v}\prod_{i=1}^{|A_v|}(1-\varepsilon_i)(1-\varepsilon'_i)(\varphi(w_{\max_v})-\varphi(v)) =$$
$$\frac{(k-\mathrm{Con}_v)}{|V^v_{\max_v+1}|-\mathrm{Con}_v}\prod_{i=1}^{|A_v|}(1-\varepsilon_i)(1-\varepsilon'_i)\left(\log\frac{\log\frac{2n}{|V^{w_{\max_v}}_{\max_v+1}|-\mathrm{Con}_{w_{\max_v}}}}{\log\frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}}+\sum_{i=1}^{|A_v|}\log\frac{\log\frac{2n}{|V_i^{w_{\max_v}}|}}{\log\frac{2n}{|V_i^v|}}\right)\leq$$
$$\frac{(k-\mathrm{Con}_v)}{|V^v_{\max_v+1}|-\mathrm{Con}_v}\prod_{i=1}^{|A_v|}(1-\varepsilon_i)(1-\varepsilon'_i)\left(\log\frac{\log\frac{2n}{k-\mathrm{Con}_v}}{\log\frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}}+\sum_{i=1}^{|A_v|}\log\left(1+\log\frac{1}{1-\varepsilon_i}\right)\right)\leq$$
$$\frac{(k-\mathrm{Con}_v)}{|V^v_{\max_v+1}|-\mathrm{Con}_v}\log\frac{\log\frac{2n}{k-\mathrm{Con}_v}}{\log\frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}}+O(1)=O(1).$$

Bounding (6) The big difference here is that the candidate set $V^{w_a}_{\max_v+1}$ becomes an active candidate set (if of course $\max_v+1 < n/3$) at w_a while $V^v_{\max_v+1}$ was not active at v. Because

of this, we have

$$\varphi(w_a) - \varphi(v) \leq \log\log \frac{2n}{|V_{a+1}^{w_a}| - \mathrm{Con}_{w_a}} + \log\log \frac{2n}{|V_{\max_v+1}^{w_a}|} - \log\log \frac{2n}{|V_{\max_v+1}^{v}| - \mathrm{Con}_v} + \sum_{j \in A_v} \log \frac{\log \frac{2n}{|V_j^{w_a}|}}{\log \frac{2n}{|V_j^v|}}.$$

Thus, we get that

$$\sum_{a>\max_v}^{n/3} \Pr[F(x_v) = a \mid \Pi \in S_v](\varphi(w_a) - \varphi(v)) =$$

$$\sum_{a>\max_v}^{n/3} \Pr[F(x_v) = a \mid \Pi \in S_v] \log\log \frac{2n}{|V_{a+1}^{w_a}| - \mathrm{Con}_{w_a}} + \tag{15}$$

$$\sum_{a>\max_v}^{n/3} \Pr[F(x_v) = a \mid \Pi \in S_v] \log \left(\frac{\log \frac{2n}{|V_{\max_v+1}^{w_a}|}}{\log \frac{2n}{|V_{\max_v+1}^{v}| - \mathrm{Con}_v}} \right) + \tag{16}$$

$$\sum_{a>\max_v}^{n/3} \Pr[F(x_v) = a \mid \Pi \in S_v] \sum_{j \in A_v} \log \frac{\log \frac{2n}{|V_j^{w_a}|}}{\log \frac{2n}{|V_j^v|}}. \tag{17}$$

Using the previous ideas, it is easy to see that we can bound (17) from above by

$$\leq \sum_{a>\max_v}^{n/3} \big(\Pr[F(x_v) = a \mid \Pi \in S_v \wedge F(x_v) > \max_v] \cdot \Pr[F(x_v) > \max_v \mid \Pi \in S_v] \cdot \sum_{j \in A_v} \log \frac{\log \frac{2n}{|V_j^{w_a}|}}{\log \frac{2n}{|V_j^v|}} \big)$$

$$\leq \sum_{a>\max_v}^{n/3} \big(\Pr[F(x_v) = a \mid \Pi \in S_v \wedge F(x_v) > \max_v] \cdot \prod_{i=1}^{|A_v|} (1-\varepsilon_i)(1-\varepsilon_i') \sum_{j \in A_v} \log \frac{\log \frac{2n}{|V_j^{w_a}|}}{\log \frac{2n}{|V_j^v|}} \big)$$

$$\leq \prod_{i=1}^{|A_v|} (1-\varepsilon_i)(1-\varepsilon_i') \sum_{j \in A_v} \log \frac{\log \frac{2n}{|V_j^{w_a}|}}{\log \frac{2n}{|V_j^v|}}$$

$$\leq \prod_{i=1}^{|A_v|} (1-\varepsilon_i)(1-\varepsilon_i') \sum_{j \in A_v} \log \left(1 + \log \frac{1}{1-\varepsilon_j} \right)$$

$$\leq \prod_{i=1}^{|A_v|} (1-\varepsilon_i) \sum_{j \in A_v} \log \frac{1}{1-\varepsilon_j}$$

$$\leq \prod_{i=1}^{|A_v|} (1-\varepsilon_i) \log \left(\frac{1}{\prod_{j \in A_v}(1-\varepsilon_j)} \right) = O(1).$$

To analyze (16) by Lemma 5.11 we know that $\Pr[F(x_v) > \max_v \mid \Pi \in S_v] \leq \frac{|V^v_{\max_v+1}|-k}{|V^v_{\max_v+1}|-\mathrm{Con}_v}$ in which k is as defined in the lemma. Note that in this case $|V^{w_a}_{\max_v+1}| = |V^v_{\max_v+1}| - k$. This implies that (16) is at most

$$\sum_{a>\max_v}^{n} \Pr[F(x_v) = a \mid \Pi \in S_v] \log\left(\frac{\log \frac{2n}{|V^v_{\max_v+1}|-k}}{\log \frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}}\right) =$$

$$\left(\sum_{a>\max_v}^{n} \Pr[F(x_v) = a \mid \Pi \in S_v]\right) \log\left(\frac{\log \frac{2n}{|V^v_{\max_v+1}|-k}}{\log \frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}}\right) =$$

$$\Pr[F(x_v) > \max_v \mid \Pi \in S_v] \log\left(\frac{\log \frac{2n}{|V^v_{\max_v+1}|-k}}{\log \frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}}\right) \leq$$

$$\frac{|V^v_{\max_v+1}| - k}{|V^v_{\max_v+1}| - \mathrm{Con}_v} \log\left(\frac{\log \frac{2n}{|V^v_{\max_v+1}|-k}}{\log \frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}}\right) = O(1).$$

It is left to analyze (15). Let $x_0 = |P^v_0|$ and $x_1 = |P^v_1|$. Let b_i be a Boolean such $b_i \equiv i$. Note that we have $|V^{w_{\max_v+a}}_{\max_v+a+1}| = x_{b_{\max_v+a}} = n - x_{b_{\max_v+a+1}}$. Using Corollary 5.13 we the term (15) is at most

$$\sum_{a=1}^{n/3-\max_v} \Pr[F(x_v) = \max_v + a \mid \Pi \in S_v] \log\log \frac{2n}{|V^{w_{\max_v+a}}_{\max_v+a+1}| - \mathrm{Con}_{w_{\max_v+a}}}$$

$$\leq \sum_{a=1}^{n/3-\max_v} 2^{-\Omega(a)}\Big(1 - \frac{x_{b_{\max_v+a+1}} - \lfloor(\max_v + a + 1)/2\rfloor}{n - \max_v - a}\Big) \cdot \log\log \frac{2n}{n - x_{b_{\max_v+a+1}} - \lfloor(\max_v + a)/2\rfloor}$$

$$= O(1).$$

5.3 Concluding the Proof of Theorem 5.1

Intuitively, if the maximum score value increases after a query, it increases, in expectation, only by an additive constant. In fact, as shown in Corollary 5.13, the probability of increasing the maximum score value by α after one query is $2^{-\Omega(\alpha)}$. Thus, it follows from the definition of the active candidate sets that when the score reaches $n/3$ we expect $\Omega(n)$ active candidate sets. However, by Lemma 5.2, the active candidate sets are disjoint. This means that a fraction of them (again at least $\Omega(n)$ of them), must be small, or equivalently, their total potential is $\Omega(n \log\log n)$, meaning, at least $\Omega(n \log\log n)$ queries have been asked. In the rest of this section, we prove this intuition.

Given an input (z, π), we say an edge e in the decision tree T is *increasing* if e corresponds to an increase in the maximum score and it is traversed given the input (z, π). We say that an increasing edge is *short* if it corresponds to an increase of at most c in the maximum function score (in which c is a sufficiently large constant) and we call it *long* otherwise. Let N be the random variable denoting the number of increasing edges seen on input Π before reaching a node with score greater than $n/3$. Let L_j be the random indicator variable taking the value

0 if the jth increasing edge is short, and taking the value equal to the amount of increase in the score along this edge if not. If $j > N$, then we define $L_j = 0$. Also let W_j be the random variable corresponding to the node of the decision tree where the jth increase happens. As discussed, we have shown that for every node v, $\Pr[L_j \geq \alpha | W_j = v] \leq 2^{-\Omega(\alpha)}$. We want to upper bound $\sum_{j=1}^{n} \mathbb{E}[L_j]$ (there are always at most n increasing edges). From the above, we know that

$$\mathbb{E}[L_j] \leq \mathbb{E}[L_j \mid N \geq j] = \sum_{v \in T} \sum_{i=c+1}^{n} i \cdot \Pr[L_j = i \wedge W_j = v \mid N \geq j]$$
$$= \sum_{v \in T} \sum_{i=c+1}^{n} i \cdot \Pr[L_j = i \wedge W_j = v] = \sum_{v \in T} \sum_{i=c+1}^{n} i \cdot \Pr[L_j = i \mid W_j = v] \Pr[W_j = v]$$
$$\leq \sum_{v \in T} \sum_{i=c+1}^{n} \frac{i}{2^{\Omega(i)}} \Pr[W_j = v] \leq \sum_{v \in T} \frac{1}{2^{\Omega(c)}} \Pr[W_j = v] \leq \frac{1}{2^{\Omega(c)}},$$

where the summation is taken over all nodes v in the decision tree T. The computation shows $\sum_{j=1}^{n} \mathbb{E}[L_j] \leq n/2^{\Omega(c)}$. By Markov's inequality, we get that with probability at least $3/4$, we have $\sum_{j=1}^{n} L_j \leq n/2^{\Omega(c)}$. Thus, when the function score reaches $n/3$, short edges must account for $n/3 - n/2^{\Omega(c)}$ of the increase which is at least $n/6$ for a large enough constant c. Since any short edge has length at most c, there must be at least $n/(6c)$ short edges. As discussed, this implies existence of $\Omega(n)$ active candidate sets that have size $O(1)$, meaning, their contribution to the potential function is $\Omega(\log \log n)$ each. We have thus shown:

Lemma 5.14. *Let ℓ be the random variable giving the leaf node of T that the deterministic query scheme ends up in on input Π. We have $\varphi(\ell) = \Omega(n \log \log n)$ with probability at least $3/4$.*

Finally, we show how Lemma 5.7 and Lemma 5.14 combine to give our lower bound. Essentially this boils down to showing that if the query scheme is too efficient, then the query asked at some node of T increases the potential by $\omega(1)$ in expectation, contradicting Lemma 5.7. To show this explicitly, define $\mathbf{t}$ as the random variable giving the number of queries asked on input Π. We have $\mathbb{E}[\mathbf{t}] = t$, where t was the expected number of queries needed for the deterministic query scheme. Also let $\ell_1, \ldots, \ell_{4t}$ be the random variables giving the first $4t$ nodes of T traversed on input Π, where $\ell_1 = r$ is the root node and ℓ_i denotes the node traversed at the ith level of T. If only $m < 4t$ nodes are traversed, define $\ell_i = \ell_m$ for $i > m$; i.e., $\varphi(\ell_i) = \varphi(\ell_m)$. From Lemma 5.14, Markov's inequality and a union bound, we may now write

$$\mathbb{E}[\varphi(\ell_{4t})] = \mathbb{E}\left[\varphi(\ell_1) + \sum_{i=1}^{4t-1} \varphi(\ell_{i+1}) - \varphi(\ell_i)\right] = \mathbb{E}[\varphi(r)] + \mathbb{E}\left[\sum_{i=1}^{4t-1} \varphi(\ell_{i+1}) - \varphi(\ell_i)\right]$$
$$= \sum_{i=1}^{4t-1} \mathbb{E}[\varphi(\ell_{i+1}) - \varphi(\ell_i)] = \Omega(n \log \log n).$$

Hence there exists a value i^*, where $1 \leq i^* \leq 4t - 1$, such that

$$\mathbb{E}[\varphi(\ell_{i^*+1}) - \varphi(\ell_{i^*})] = \Omega(n \log \log n / t).$$

But

$$\mathbb{E}[\varphi(\ell_{i^*+1}) - \varphi(\ell_{i^*})] = \sum_{v \in T_{i^*}, v \text{ non-leaf}} \Pr[\Pi \in S_v] \, \mathbb{E}[\varphi(w_{\mathbf{i}_v}) - \varphi(v) \mid \Pi \in S_v],$$

where T_{i^*} is the set of all nodes at depth i^* in T, $w_0, \dots, w_n$ are the children of v and $\mathbf{i}_v$ is the random variable giving the score of $F(x_v)$ on an input $\Pi \in S_v$ and 0 otherwise. Since the events $\Pi \in S_v$ and $\Pi \in S_u$ are disjoint for $v \neq u$, we conclude that there must exist a node $v \in T_{i^*}$ for which

$$\mathbb{E}[\varphi(w_{\mathbf{i}_v}) - \varphi(v) \mid \Pi \in S_v] = \Omega(n \log\log n/t).$$

Combined with Lemma 5.7 this shows that $n \log\log n/t = O(1)$; i.e., $t = \Omega(n \log\log n)$. This concludes the proof of Theorem 5.1.

Acknowledgments

Most of this work was done while Benjamin Doerr was with the Max Planck Institute for Informatics (MPII) in Saarbrücken, Germany, and Carola Doerr was with the MPII and Université Diderot, Paris, France.

Carola Doerr acknowledges support from a Feodor Lynen postdoctoral research fellowship of the Alexander von Humboldt Foundation and the Agence Nationale de la Recherche under the project ANR-09-JCJC-0067-01.

References

[AAD+13] Peyman Afshani, Manindra Agrawal, Benjamin Doerr, Carola Doerr, Kasper Green Larsen, and Kurt Mehlhorn. The query complexity of finding a hidden permutation. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2013.

[Chv83] Vasek Chvátal. Mastermind. *Combinatorica*, 3:325–329, 1983.

[DDST16] Benjamin Doerr, Carola Doerr, Reto Spöhel, and Henning Thomas. Playing mastermind with many colors. *Journal of the ACM*, 63:42:1–42:23, 2016.

[DJK+11] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Per Kristian Lehre, Markus Wagner, and Carola Winzen. Faster black-box algorithms through higher arity operators. In *Proc. of Foundations of Genetic Algorithms (FOGA'11)*, pages 163–172. ACM, 2011.

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

[DJW06] Stefan Droste, Thomas Jansen, and Ingo Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, 39:525–544, 2006.

[DL18] Carola Doerr and Johannes Lengler. The (1+1) elitist black-box complexity of LeadingOnes. *Algorithmica*, 80:1579–1603, 2018.

[Doe18] Carola Doerr. Complexity theory for discrete black-box optimization heuristics. *CoRR*, abs/1801.02037, 2018.

[DW12] Benjamin Doerr and Carola Winzen. Black-box complexity: Breaking the $O(n \log n)$ barrier of LeadingOnes. In *Proc. of Artificial Evolution (EA'11)*, volume 7401 of *LNCS*, pages 205–216. Springer, 2012. Available online at http://arxiv.org/abs/1210.6465.

[ER63] Paul Erdős and Alfréd Rényi. On two problems of information theory. *Magyar Tudományos Akadémia Matematikai Kutató Intézet Közleményei*, 8:229–243, 1963.

[LW12] Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. *Algorithmica*, 64:623–642, 2012.

[Rud97] Günter Rudolph. *Convergence Properties of Evolutionary Algorithms*. Kovac, 1997.

[Val79] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.

[Yao77] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. of Foundations of Computer Science (FOCS'77)*, pages 222–227. IEEE, 1977.