Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 87 (2012)

The Query Complexity of Finding a Hidden Permutation

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

[1]MADALGO, Department of Computer Science, Aarhus University, Denmark
[2]Indian Institute of Technology Kanpur, India
[3]Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract

We study the query complexity of determining a hidden permutation. More specifically, we study the problem of learning a secret (z, π) consisting of a binary string z of length n and a permutation π of $[n]$. The secret must be unveiled by asking queries $x \in \{0,1\}^n$, and for each query asked, we are returned the score $f_{z,\pi}(x)$ defined as

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

i.e., the length of the longest common prefix of x and z with respect to π. The goal is to minimize the number of queries asked.

Our main result are matching upper and lower bounds for this problem, both for deterministic and randomized query schemes. The deterministic query complexity is $\Theta(n \log n)$, which, surprisingly, improves to $\Theta(n \log\log n)$ in the randomized setting.

For the randomized query complexity, both the upper and lower bound are stronger than what can be achieved by standard arguments like the analysis of random queries or information-theoretic considerations. Our proof of the $\Omega(n \log\log n)$ lower bound is based on a potential function argument, which seems to be uncommon in the query complexity literature. We find this potential function technique a very powerful tool in proving lower bounds for randomized query schemes and we expect it to find applications in many other query complexity problems.

Keywords: Query complexity; randomized algorithms.

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 (the 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. Surprisingly, for many simple and natural problems we do not know the number of queries necessary, both if randomization is allowed or not.

In the context of Boolean functions—see [BdW02] for a survey—the goal is to evaluate a given Boolean function f on a secret argument x by querying bits of x. For example, x could be the adjacency matrix of a graph and f is one (zero) if the graph is connected (not connected). For graph connectivity the deterministic query complexity is $\Theta(n^2)$ and the randomized complexity is only known to be $\Omega(n^{4/3} \log^{1/3} n)$, cf. [CK07].

It is an open question since Saks' and Wigderson's paper [SW86] from 1986 to determine the randomized query complexity of the recursive 3-Majority function. It asks to determine the value of the root of a complete ternary tree of height h. The value of every internal node in the

ISSN 1433-8092

tree is the majority value of its three children. We have query access to the 3^h leaves, which are labeled with binary values. Recent progress include the papers by Jayram, Kumar, and Sivakumar [JKS03] and Magniez, Nayak, Santha, and Xiao [MNSX11]. Magniez et al. provide the current best lower and upper bounds of $(5/2)^h$ and $(1.004) \cdot 2.64946^h$, respectively.

Numerous related—seemingly simple looking problems—of unknown query complexities exist. In the field of property testing, it is open to date how many queries to a Boolean function $f : \{0,1\}^n \to \{0,1\}$ we need to determine with probability $2/3$ whether f is a k-parity function or far from being one. For non-adaptive testing sequences, Buhrman, García-Soriano, Matsliah, and de Wolf [BGSMdW12] very recently could prove that the query complexity is $\Theta(k \log k)$. For the general, adaptive, case the best known lower bound [BBM12] is $\Omega(k)$, while the best known upper bound is $O(k \log k)$.

Another long-standing open question is the coin weighing problem. Here we are given n coins of two different weights and a spring scale. Our goal is to classify the coins into light and heavy ones. We may use the spring scale to weigh arbitrary subsets of the coins. What is the smallest number of weighings needed, in the worst-case, to classify the coins? This problem remains open since 1963 when Erdős and Rényi [ER63] showed a a lower bound of $(1 + o(1))n/\log_2 n$ and an upper bound of $(1 + o(1))(\log_2 9)n/\log_2 n$. The upper bound was subsequently improved to $(1 + o(1))2n/\log_2 n$ by Lindström [Lin65] and, independently, by Cantor and Mills [CM66], but no tight bound is known to date. A number of different versions of coin weighing problems exist. A good survey can be found in Bshouty's paper on polynomial time query-optimal algorithms for coin weighing problems [Bsh09].

A related open problem is the query complexity of the Mastermind game with n positions and $k = n$ colors. In Mastermind, the secrets and queries are strings of length n over a k-ary alphabet. If the secret is z, a query x returns the number $\mathrm{eq}(z, x) := |\{i \mid x_i = z_i\}|$ of positions in which x and the secret code z coincide. Although the asymptotically best upper bound of Chvátal [Chv83] was recently improved from $O(n \log n)$ to $O(n \log \log n)$, cf. [DSTW13], the best known lower bound is the trivial linear one. This problem is open for more than 30 years. For both the coin weighing problem of Erdős and Rényi as well as for the Mastermind problem of Chvátal it is widely believed that the current lower bounds are not optimal.

Our Contribution In this work, we shift the focus from searching for secret bits (or bit-strings) to hidden permutations. Apart from the classic result that $\Theta(n \log n)$ comparison queries are necessary and sufficient for sorting, we are not aware of any other query complexity theoretic work on finding permutations.

The query problem we regard originates from theoretical work on randomized search heuristics (see the end of this section), but seems to be a good first example for studying permutation query problems. In particular, it nicely captures the aspect of order/sequentiality in that $\pi(i)$ is easier to detect than $\pi(j)$ when $i < j$.

Let S_n denote the set of permutations of $[n] := \{1, \ldots, n\}$; let $[0..n] := \{0, 1, \ldots, n\}$. Our problem is that of learning a hidden permutation $\pi \in S_n$ together with a hidden bit-string $z \in \{0,1\}$ through (as few as possible) 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.

We show the following.

(1) We exhibit a *randomized strategy* that finds the secret with an expected number of $O(n \log \log n)$ queries (Section 4). This improves upon the $O(n \log n/\log \log n)$ bound

from [DW12].

The $O(n \log \log n)$ bound is quite remarkable. Putting it into an information-theoretic perspective, this result shows that we succeed in learning an average of $\Omega(\log n / \log \log n)$ bits of information in each query. This in particular means that our queries allow many different answers to show up with sufficiently high probability. Hence we partially overcome the slowness of learning one $(z_{\pi(i)}, \pi(i))$ after the other in the order $i = 1, \ldots, n$.

Our strategy is efficient; i.e., it can be implemented in polynomial time.

(2) We then show that this upper bound is asymptotically optimal (Section 5). The lower bound is derived from a potential function argument, which, as far as we know, seems uncommon in the query complexity literature. We believe this potential function approach is very interesting in its own right, and we expect that it may be applied to many other query complexity problems to yield improved lower bounds.

The lower bound is particularly interesting as for many related problems (sorting, Mastermind, and many coin weighing problems, the asymptotic query complexity or the best known lower bound equals the information-theoretic lower bound. This is not the case here, where the information-theoretic lower bound is linear (cf. Section 2 for a sketch). This also was the best lower bound known so far.

(3) Interestingly, and unlike for the problems mentioned in the previous paragraph, the deterministic query complexity of our problem is asymptotically larger than the randomized one. In Section 3, we prove a lower bound of $\Omega(n \log n)$ queries for any deterministic query scheme, which is easily proven tight by sequentially determining $(z_{\pi(i)}, \pi(i))$ for $i = 1, \ldots, n$.

Origin of the Problem The problem we regard in this work has its origins in the field of evolutionary algorithms. Here, the so-called LeadingOnes function

$$\{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\}$$

is a commonly used testbed both for experimental and theoretical analyses (e.g., [Rud97]). It turns out that many genetic and evolutionary algorithms as well as other general-purpose randomized search heuristics used in practice query the fitness of $\Theta(n^2)$ solution candidates until they find the maximum of LeadingOnes, see, e.g., [DJW02]. Naturally, the same is true when LeadingOnes is replaced by any function LeadingOnes $\circ \sigma$, where σ is an automorphism of the n-dimensional cube.

In an attempt to understand the intrinsic difficulty of a problem for general purpose search heuristics, Droste, Jansen and Wegener [DJW06] suggested a query complexity theoretic notion called black-box complexity, which roughly speaking (see [DJW06] for the details) is the number of search points a black-box optimization algorithm has to evaluate to find the optimum of an unknown optimization instance from a known problem. The connection to our work is made by the easy observation that the black-box complexity of the class of all LeadingOnes $\circ \sigma$ functions is asymptotically equal to the randomized query complexity of our HiddenPermutation problem. For this reason, the black-box complexity result in [DW12] implies an upper bound of $O(n \log n / \log \log n)$ for the randomized query complexity of our problem.

2 Preliminaries

For all positive integers $k \in \mathbb{N}$ we define $[k] := \{1, \ldots, k\}$ and $[0..k] := [k] \cup \{0\}$. By e_k^n we denote the kth unit vector $(0, \ldots, 0, 1, 0, \ldots, 0)$ 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 that we *create y from x by flipping I* or that we *create 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 is called the *target string* of $f_{z,\pi}$ and π is called the *target permutation*. In the HiddenPermutation problem we want to identify target string and target permutation by asking queries x^i 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 randomized strategy for the HiddenPermutation problem is a tree of outdegree $n+1$ in which 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. The complexity of a strategy on input (z,π) is the expected number of queries required to identify the secret, and the randomized query complexity of a strategy is the worst case over all secrets. Deterministic strategies are the special case in which the probability distributions are one-point distributions; i.e., a fixed query is associated with every node. The deterministic (randomized) query complexity of HiddenPermutation is the best possible (expected) complexity.

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 $\{0,1\}^n \times S_n$. That is, we need to "learn" $\Omega(n \log n)$ bits of information. Each *score* is a number between 0 and n, i.e., we learn at most $O(\log n)$ bits of information per query, and the $\Omega(n)$ bound follows.

Let $\mathcal{H} := (x^i, s^i)_{i=1}^t$ be a vector 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.

Theorem 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 $j \leq s^h \leq s^\ell$ 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 $f + 1 \leq s^*$, 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 1. In the beginning, let $V_j := [n]$, $1 \leq j \leq n$. 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 objective value of x and let s^* be the objective value of x^*.

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

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

Rule 3: If $s > s^*$, then $V_i \leftarrow V_i \cap I$ for $1 \leq i \leq s^*$ 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 V_js 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 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. *It is decidable in polynomial time whether a guessing history is feasible. Furthermore, we can efficiently compute the number of pairs consistent with it.*

(sketch. A full proof is in the appendix). Let $V_1, \ldots, V_n$ be as in Theorem 1. We construct a bipartite graph $G = G(V_1, \ldots, V_n)$ with node set $[n]$ on both sides. Connect j to all elements 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 G. If there is no perfect matching, the history is infeasible. Otherwise, let π be any permutation with $\pi(j) \in V_j$ for all j. Set $z_{\pi(i)} := x^*_{\pi(i)}$ for $i \in [s^*]$, and $z_{\pi(i)} := 1 - x^*_{\pi(i)}$ for $i \notin [s^*]$. The pair (z, π) is consistent with $\mathcal{H}$.

For the counting problem we first observe that for any fixed consistent permutation π, the number of strings z such that (z, π) is consistent with $\mathcal{H}$ equals $2^{n-(s^*+1)}$ if $s^* < n$, and it equals one if $s^* = n$. By Hall's condition a perfect matching exists if and only if $|\bigcup_{j \in J} V_j| \geq |J|$ for every $J \subseteq [n]$. Constructing the permutations in a greedy fashion, one sees that the number of consistent permutations is equal to $\prod_{1 \leq i \leq n} (|V_i| - |\{j < i \mid V_j \subseteq V_i\}|)$.

The fact that the V_js are a laminar family is crucial for the counting result. Counting the number of perfect matching in a general bipartite graph is #P-complete. □

3 Deterministic Complexity

In this section, we present simple proofs that completely settle the deterministic query complexity of the HiddenPermutation problem. Specifically, we prove

Theorem 3. *The deterministic query complexity of the* HiddenPermutation *problem with n positions is $\Theta(n \log n)$.*

The lower bound is proved by examining the decision tree of the deterministic query scheme and finding 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 as follows: First consider the root node r of the decision tree. There is one child for every possible score $0, 1, \ldots, n$. The crucial observation in the following steps is that there is always one of the two children corresponding to scores 0 and 1 for which the size of the candidate set V_1 at most halves and $V_3, \ldots, V_n$ do not change when following the update rules from the preceding section. That is, for any query asked in the first round, there is an answer (either 0 or 1) to it such that the number of possible values of $\pi(1)$ at most halves. Thus we recurse into that child on the subset of inputs consistent with that answer. We may continue this for $\Omega(\log n)$ steps, until $|V_1| = 2$. At this point, let v be the node reached in the decision tree and let x^* be an arbitrary one of the queries asked which achieved the maximal score (remember the maximal score is either 0 or 1). Now choose $i' \in V_1$ and $i'' \in V_2$ arbitrarily and consider the subset of all inputs for which $i' = \pi(1)$, $i'' = \pi(2)$, $z_{i'} = x^*_{i'}$, and $z_{i''} = 1 - x^*_{i''}$. For all such inputs, the query path followed in the decision tree must start by descending 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' and i'', 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.

The upper bound is achieved by an algorithm that resembles binary search and iteratively identifies $\pi(1), \ldots, \pi(n)$ and the corresponding bit values $z_{\pi(1)}, \ldots, z_{\pi(n)}$: We start by querying the all-zeros string $(0, \ldots, 0)$ and the all-ones string $(1, \ldots, 1)$. The scores determine $z_{\pi(1)}$. By flipping a set I containing half of the bit positions in the better (the one achieving largest score) of the two strings, we can determine whether $\pi(1) \in I$ or not. This allows us to find $\pi(1)$ via a binary search strategy 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, cf. Section B.2.

4 The Randomized Strategy

We now show that the randomized query complexity is only $O(n \log \log n)$. This implies that we found a way to overcome the sequential learning process of the binary search strategy (typically revealing a constant amount of information per query) and instead have 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 V_is in parallel, that is, we gain information on several $\pi(i)$s despite the seemingly sequential way $f_{z,\pi}$ 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. *The randomized query complexity of the* HiddenPermutation *problem with n positions is $O(n \log \log n)$.*

The strategy has two parts. In the main 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. Part 1 is outlined below; the details are in the appendix.

4.1 The Main 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 $f_{z,\pi}$.

There is a trade-off between learning more information about π by reducing the sets $V_1, \ldots, V_{s^*+1}$ and increasing the score s^*. Our main task is to find the right balance between the two. In our $O(n \log\log n)$ strategy, we partition the sets $V_1, \ldots, V_n$ into several *levels*, each of which has a certain capacity. 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.

In the beginning, all candidate sets $V_1, \ldots, V_n$ belong to level 0. In the *first step* we aim at moving $V_1, \ldots, V_{\log n}$ to the first level. 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 y by flipping the bits in F_1. If $f(y') > f(x)$, we set $V_1 \leftarrow V_1 \setminus F_1$; we set $V_1 \leftarrow F_1$ otherwise. This ensures $\pi(1) \in V_1$. We stop this binary search once $\pi(2) \notin V_1$ is sufficiently likely; the analysis will show that $\Pr[\pi(2) \in V_1] \leq 1/\log^d n$ (and hence $|V_1| \leq n/\log^d n$) for some large enough constant d is a good choice.

We now start pounding on 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 now 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 we have $V_2 = [n] \setminus V_1$. At each step we chose a subset $F_2 \subseteq V_2$ of size $|V_2|/2$ and we create y' from y by flipping the bits in positions F_2. If $f(y') = 1$ we update V_2 by F_2 and we update V_2 by $V_2 \setminus F_2$ otherwise. We stop once $|V_2| \leq n/\log^d n$.

At this point we have $|V_1|, |V_2| \leq n/\log^d n$ and $V_1 \cap V_2 = \emptyset$. We hope that $\pi(3) \notin V_1 \cup V_2$, in which case we move set V_3 from level 0 to level 1 (we comment on the case $\pi(3) \in V_1 \cup V_2$ in the *failure* section, Section 4.2).

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 the current best score 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 $n/\log^{2d} n$. This reduction is carried out by `subroutine2`, which we describe in Section 4.3.

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 $[n] \setminus (V_1 \cup \ldots \cup V_{s^*})$ 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 $n/\log^{2d} n$, 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 $\log^2 n$ 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 $n/\log^{4d} n$, thus adding them to the *third level.*

In total we have $t = O(\log\log n)$ levels. For $1 \leq i \leq t$, the ith level has a capacity of $x_i := \log^{2^{i-1}} n$ sets, each of which is required to be of size at most n/x_i^d. Once level i has reached its capacity, we reduce the size of the sets on the ith level to at most n/x_{i+1}^d, thus moving them from level i to level $i+1$. When x_t sets $V_i, \ldots, V_{i+x_t}$ have been added to the last

level, level t, we finally reduce their sizes one. This corresponds to determining $\pi(i+j)$ for each $j \in [x_t]$.

This concludes the presentation of the main ideas of the first phase. We still need to discuss how to handle failures, how to move sets from some level i to level $i+1$ (`subroutine2`), and we need to compute the query complexity of our algorithm.

4.2 Failures

We say that a failure happens if we want to move some set V_{j+1} from level 0 to level 1, but either $f(y) = f(y \oplus 1_{[n]\setminus\{V_1\cup\ldots\cup V_j\}}) = j$ or $f(y), f(y \oplus 1_{[n]\setminus\{V_1\cup\ldots\cup V_j\}}) > j$ holds. In this case we can conclude that $\pi(j+1) \in V_1 \cup \ldots \cup V_j$. Therefore, we immediately *abort* the first level. That is, we move all sets on the first level to the second one. As before, this is done by calls to `subroutine2` 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 $\{f(y), f(y \oplus 1_{[n]\setminus\{V_1\cup\ldots\cup V_j\}}\} = \{j, j+\ell\}$ for some $\ell \geq 1$. Should we still have $\pi(j+1) \in V_1 \cup \ldots \cup V_j$, we continue by moving all level 2 sets to level 3, and so on, until we finally have $\pi(j+1) \notin V_1 \cup \ldots \cup V_j$. At this point, we proceed again by moving sets from level 0 to level 1, starting of course with set V_{j+1}.

The pseudo-code summarizing phase 1 can be found in the appendix, cf. Algorithm 1.

4.3 Subroutine2

We now describe how to reduce the sizes of the up to $x_{\ell-1}$ candidate sets from some value $\leq n/x_{\ell-1}^d$ to the target size n/x_ℓ^d of level ℓ with an expected number of $O(1)x_{\ell-1}d(\log(x_\ell) - \log(x_{\ell-1}))/\log(x_{\ell-1})$ queries. For this reduction we introduce procedure `subroutine2`: It reduces the sizes of at most k candidate sets to a kth fraction of their original size using at most $O(k)$ queries (k is a parameter of the routine). We use `subroutine2` with parameter $k = x_{\ell-1}$ repeatedly to achieve the full reduction of the sizes to at most n/x_ℓ^d.

`subroutine2` is given a set J of at most k indices and a string y with $f(y) \geq \max J$. 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 y by flipping the entries in positions $\cup_{j\in J} F_j$. A condition on the sets $V_j, j \in J$, (see Lemma 5) ensures that 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.

The pseudo-code of `subroutine2` can be found in the appendix, cf. Algorithm 4 in Section B.3.

Lemma 5. *Let $k \in \mathbb{N}$ and let $J \subseteq [n]$ be a set of at most k indices with $V_j \cap V_p = \emptyset$ for $j \in J$ and $p \in [\max J] \backslash \{j\}$. Let $y \in \{0,1\}^n$ be such that $f(y) \geq \max J$ 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 `subroutine2`(k, J, m, y) *has reduced the size of V_j to at most m for each $j \in J$.*

Using successive calls to `subroutine2` we get the following corollary.

Corollary 6. *Let $k \in \mathbb{N}$, J, and y be as in Lemma 4. Let further $d \in \mathbb{N}$ and $x \in \mathbb{R}$ such that $\max_{j \in J} |V_j| = n/x^d$. Let $y \in \mathbb{R}$ with $y > x$.*

Using at most $d(\log y - \log x)/\log k$ calls to `subroutine2` *we can reduce the maximal size $\max_{j \in J} |V_j|$ to n/y^d. The overall expected number of queries needed to achieve this reduction is $O(1)kd(\log y - \log x)/\log k$.*

4.4 Proof of Theorem 4

It remains to show that the first phase of our strategy requires at most $O(n \log\log n)$ queries. If no failure happened, the expected number of queries was bounded by

$$\begin{aligned}
&\frac{q}{x_t}\left(\frac{x_t \log n}{\log x_t} + \frac{x_t}{x_{t-1}}\left(\frac{x_{t-1} dc(\log x_t - \log x_{t-1})}{\log x_{t-1}}\right.\right. \\
&\qquad\qquad \left.\left. + \frac{x_{t-1}}{x_{t-2}}\left(\ldots + \frac{x_2}{x_1}\left(\frac{x_1 dc(\log x_2 - \log x_1)}{\log x_1} + x_1 d \log x_1\right)\right)\right)\right) \\
\leq\, & ndc\left(\frac{\log n}{\log x_t} + \frac{\log x_t}{\log x_{t-1}} + \ldots + \frac{\log x_2}{\log x_1} + \log x_1 - t\right), \qquad (1)
\end{aligned}$$

where c is the constant hidden in the $O(1)$-term in Lemma 5. To verify this formula, observe that we fill the $(i-1)$st level x_i/x_{i-1} times before level i has reached its capacity of x_i candidate sets. To add x_{i-1} candidate sets from level $i-1$ to level i, we need to reduce their sizes from n/x_{i-1}^d to n/x_i^d. By Corollary 6 this requires at most $x_{i-1} dc(\log x_i - \log x_{i-1})/\log x_{i-1}$ queries. The additional $x_1 d \log x_1$ 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 V_is to n/x_1^d—requiring at most $d \log x_1$ queries per call. Finally, the term $x_t \log n/\log x_t$ accounts for the final reduction of the V_is to a set containing only one single element (at this stage we shall finally have $V_i = \{\pi(i)\}$). More precisely, this term is $(x_t(\log n - d \log x_t))/\log x_t$ 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 $n^2/((n-q)(x_i^{d-1}-1))$. By Corollary 6, each such level-i failure causes an additional number of at most $1 + x_i dc(\log x_{i+1} - \log x_i)/\log x_i$ queries (the 1 counts for the query through which we discover that $\pi(s^*+1) \in V_1 \cup \ldots \cup V_{s^*}$). Thus,

$$\sum_{i=1}^{t} \frac{n^2}{(n-q)(x_i^{d-1}-1)}\left(1 + \frac{x_i dc(\log x_{i+1} - \log x_i)}{\log x_i}\right) \qquad (2)$$

bounds the expected number of additional queries caused by failures.

We recall the settings of the x_i. We have $x_1 = \log n$ and $x_i = \log^{2^{i-1}} n$. We further have $t \in \Theta(\log\log n)$, so that $\log x_t = \Omega(\log n)$. The parameter d is some constant ≥ 4. With these parameter settings, formula (1) evaluates to

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

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

$$\frac{n^2dc}{n-q}\sum_{i=1}^{t} x_i^{-(d-3)} = O(n\log n)\sum_{i=0}^{t-1} x_1^{-(d-3)2^i}$$
$$< O(n\log n)(x_1^{d-3}-1)^{-1} = O(n)\,.$$

This shows that the overall expected number of queries sums to $O(n\log\log n) + O(n) = O(n\log\log n)$.

5 The Lower Bound

In this section, we prove a tight lower bound for the randomized query complexity of the HiddenPermutation problem. The lower bound is stated in the following:

Theorem 7. *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.

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 and in fact, at any node v and for any ancestor u of v, conditioned on the event that the search path reaches v, the value of $F(x_u)$ is equal to the index of the child of u that lies on the path to v. Finally, we define S_v as the subset of inputs (as outlined above) that reach node v.

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:

$$V_i^{w_j} = \begin{cases} V_i^v \cap P_{i \bmod 2}^v & \text{if } i \leq j\,, \\ V_i^v \cap P_{j \bmod 2}^v & \text{if } i = j+1\,, \\ V_i^v & \text{if } i > j+1. \end{cases}$$

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 $V_{\max_v+1}^v$ *pseudo-active (at v)*.

For intuition on the requirement $i < n/3$, observe from the following lemma that $V_{\max_v+1}^v$ contains all sets V_i^v for $i \leq \max_v$ 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.

In the appendix, we show that the candidate sets satisfy the following:

Lemma 8. *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 $V_i^v \subset V_j^v$.

5.1 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

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

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 $V_{\max_v+1}^v$. Note that from Lemma 8, 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 $|V_{\max_v+1}^v|$, the distribution of $\Pi(\max_v+1)$ starts depending heavily on the candidate sets V_i^v for $i \leq \max_v$ and $i \equiv \max_v$. Thus we have in some sense determined $\Pi(\max_v+1)$ already when $|V_{\max_v+1}^v|$ approaches Con_v. Therefore, we have to take this into account in the potential function since otherwise changing $V_{\max_v+1}^v$ from being pseudo-active to being active could give a huge potential increase.

After some lengthy calculations, it is possible to prove the following lemma. The full proof is presented in Appendix C. Here instead, we give only a very high level overview.

[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 9. *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: $F(x_v) \leq \max_v$ 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 interesting part of the analysis. This is because the probability of suddenly increasing the score by 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^v_i, and a Boolean $b \in \{0, 1\}$, let $V^v_i(b) = \{j \in V^v_i \mid x_v[j] = b\}$. Clearly, $|V^v_i(0)| + |V^v_i(1)| = |V^v_i|$. For even a_i, $1 \leq i \leq |A_v|$, let

$$\varepsilon_i = |V^v_i(1)|/|V^v_i|,$$

and for odd i, let

$$\varepsilon_i = |V^v_i(0)|/|V^v_i|.$$

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

$$\varepsilon'_i := \Pr[a_i \leq F(x_v) < a_{i+1} - 1 | \Pi \in S_v \wedge F(x_v) \geq a_i].$$

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

With these definitions, it is clear that we have

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

The important fact is that we can also bound other probabilities using the ε_is and ε'_is: We can show that (see the appendix)

$$\Pr[F(x_v) = a_i - 1 | \Pi \in S_v] \leq \varepsilon_i \Pi_{j=1}^{i-1}(1 - \varepsilon_j)(1 - \varepsilon'_j) \tag{3}$$

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 ε_is. The probability of making the various changes is also determined from the ε_is and ε'_is and finally the potential function is defined in terms of the sizes of these active candidate sets. Thus proving Lemma 9 reduces to proving an inequality showing that any possible choice of the ε_is and ε'_is provide only little expected increase in potential.

Since the full calculations are too long to fit in this extended abstract, we will in the following provide intuition for the correctness by giving parts of the analysis for the seemingly best choice for the ε_is and simply assume all ε'_is are 0; i.e., $a_i = i$ 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 $\max_v = n/4$.

From an information theoretic perspective, queries reveal the most information when the answer is uniform amongst many possibilities; i.e., if the entropy of the answer is high. Examining the probability bounds above, this corresponds to the setting where we essentially have $\varepsilon_i = \prod_{j=1}^{i-1}(1-\varepsilon_j)$. This is roughly satisfied for the choice $\varepsilon_i = 1/\max_v = 4/n$. Therefore, we will analyze the expected potential increase for the choice $\varepsilon_i = 4/n$ for all $i \leq n/4$. For this choice, the above probability bound (3) becomes (using $a_i = i$) $\Pr[F(x_v) = i-1 | \Pi \in S_v] \leq \Theta(1/n)$. Thus the expected increase in potential provided by the cases where the score does not increase is bounded by

$$\sum_{j \leq n/4} \Pr[F(x_v) = j \mid \Pi \in S_v](\varphi(w_j) - \varphi(v)) \leq \sum_{j \leq n/4} \Theta\left(\frac{1}{n}\right)(\varphi(w_j) - \varphi(v)).$$

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 $n/4$, and there are j sets that change by a factor $(1 - 4/n)$. Here we only consider the potential change caused by the sets changing by a factor of $n/4$. This change is bounded by

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

To continue these calculations, we use Lemma 8 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 $[2^i : 2^{i+1}]$:

$$\begin{aligned} \sum_{j \leq n/4} \Theta\left(\frac{1}{n}\right) \log\left(1 + \frac{\log(n/4)}{\log(2n/|V_j^v|)}\right) &\leq \Theta\left(\sum_{i=0}^{\log n - 1} \frac{1}{2^i} \log\left(1 + \frac{\log(n/4)}{\log(n/2^i)}\right)\right) \\ &\leq \Theta\left(\sum_{i=0}^{\log n - 1} \frac{\log(n/4)}{2^i \log(n/2^i)}\right). \end{aligned}$$

The last inequality followed from $\log(1+x) \leq x$ for $x > 0$. 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'. For the complete proof see Appendix C.

5.2 Potential at the End

Intuitively, if the maximum score value increases after a query, it increases, in expectation, only by an additive constant. In fact, in Appendix C, we prove that 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 8, 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 $\Omega(n \log\log n)$. 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, in Appendix C we prove 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

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

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 10. *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 $3/4$.*

5.3 Putting Things Together

Finally, we show how Lemma 9 and Lemma 10 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 9. 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 10, Markov's inequality and a union bound, we may now write

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

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

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

where T_{i^*} is the set of all nodes at depth i^* in T, $w_0, \ldots, w_n$ 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 9 this shows that $n \log\log n/t = O(1)$; i.e., $t = \Omega(n \log\log n)$. This concludes the proof of Theorem 7.

References

[BBM12] Eric Blais, Joshua Brody, and Kevin Matulef, *Property testing lower bounds via communication complexity*, Computational Complexity **21** (2012), 311–358.

[BdW02] Harry Buhrman and Ronald de Wolf, *Complexity measures and decision tree complexity: a survey*, Theoretical Computer Science **288** (2002), 21–43.

[BGSMdW12] Harry Buhrman, David García-Soriano, Arie Matsliah, and Ronald de Wolf, *The non-adaptive query complexity of testing k-parities*, CoRR **abs/1209.3849** (2012), Available at http://arxiv.org/abs/1209.3849.

[Bsh09] Nader H. Bshouty, *Optimal algorithms for the coin weighing problem with a spring scale*, Proc. of the 22nd Conference on Learning Theory (COLT'09), 2009.

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

[CK07] Amit Chakrabarti and Subhash Khot, *Improved lower bounds on the randomized complexity of graph properties*, Random Struct. Algorithms **30** (2007), 427–440.

[CM66] David G. Cantor and William H. Mills, *Determination of a subset from certain combinatorial properties*, Canadian Journal of Mathematics **18** (1966), 42–48.

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

[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** (2006), 525–544.

[DSTW13] Benjamin Doerr, Reto Spöhel, Henning Thomas, and Carola Winzen, *Playing Mastermind with many colors*, Proc. of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'13), SIAM, 2013, To appear. Available at http://arxiv.org/abs/1207.0773.

[DW12] Benjamin Doerr and Carola Winzen, *Black-box complexity: Breaking the $O(n \log n)$ barrier of LeadingOnes*, Proc. of Artificial Evolution (EA'11), Lecture Notes in Computer Science, vol. 7401, Springer, 2012, To appear. 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** (1963), 229–243.

[JKS03] T. S. Jayram, Ravi Kumar, and D. Sivakumar, *Two applications of information complexity*, Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC'03), ACM, 2003, pp. 673–682.

[Lin65] Bernt Lindström, *On a combinatorial problem in number theory*, Canadian Mathematical Bulletin **8** (1965), 477–490.

[MNSX11] Frédéric Magniez, Ashwin Nayak, Miklos Santha, and David Xiao, *Improved bounds for the randomized decision tree complexity of recursive majority*, Proccedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP'11), Lecture Notes in Computer Science, vol. 6755, Springer, 2011, pp. 317–329.

[Rud97] Günter Rudolph, *Convergence properties of evolutionary algorithms*, Kovac, 1997.

[SW86] Michael E. Saks and Avi Wigderson, *Probabilistic boolean decision trees and the complexity of evaluating game trees*, Proc. of the 27th Annual Symposium on Foundations of Computer Science (FOCS'86), IEEE Computer Society, 1986, pp. 29–38.

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

Appendix

A Details for Section 2

We repeat Theorem 2.

Theorem 2. *It is decidable in polynomial time whether a guessing history $\mathcal{H} = (x^i, s^i)_{i=1}^t$ is feasible. Furthermore, we can efficiently compute the number of consistent pairs $(z, \pi) \in \{0,1\}^n \times S_n$.*

We first show how to check feasibility in polynomial time.

of Theorem 2, Part I. We prove here the first part of Theorem 2; i.e., we show that consistency 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 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 $x^h, x^\ell \in \{0,1\}^n$

- setting z_i to opposite values by rule (a)
- setting z_i to opposite values by rule (b)
- 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 $|\cup_{j \in J} V_j| \geq |J|$ for every $J \subseteq [n]$.

of Theorem 2, Part II. We now prove the second part of Theorem 2; i.e., we show how to efficiently count the number of *consistent pairs*.

According to Theorem 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. When we choose $\pi(i)$, the number of choices left is $|V_i|$ 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

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

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

The proof of Theorem 2 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 $|\cup_{j \in J} V_j \setminus \{\ell\}| < |J|$. Since G contains a perfect matching (assuming a consistent history), this implies $|\cup_{j \in J} V_j| = |J|$; i.e., J is tight for Hall's condition. We have 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 $\ell \in \cup_{j \in J} V_j$. Since the V_i form a laminar family, 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., $|\cup_{j \in J} V_j| = |J|$, we can learn $\pi|J$ easily. 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 $\cup_{j \in J} V_j$ exactly once. The objective values determine the permutation.

B Details on the Randomized Strategy (Section 4)

For readability purposes, we repeat the material presented in the main text. The reader only interested in the proof details may skip Section B.1 and most parts of Section B.4.

We repeat the main theorem of this section.

Theorem 4. *The randomized query complexity of the* HiddenPermutation *problem with n positions is $O(n \log \log n)$.*

The strategy verifying this bound has two parts. In the beginning, we use Algorithm 1 to identify the positions $\pi(1), \ldots, \pi(q)$ and the corresponding bit values $z_{\pi(1)}, \ldots, z_{\pi(q)}$; for some $q \in n - \Theta(n/\log n)$. As we shall see below, this requires $O(n \log \log n)$ queries, cf. Theorem 6. Once $z_{\pi(1)}, \ldots, z_{\pi(q)}$ have been identified, we can find the remaining $n - q \in \Theta(n/\log n)$ positions and entries using the binary search algorithm described in the introduction (details will be given in Section B.2. This is the second phase of our algorithm. As the binary search strategy requires at most $O(\log n)$ queries per position, the second phase contributes only a linear number of queries—an effort that is negligible compared to the one of the first phase.

B.1 Outline of the Proof for Theorem 4

We remind the reader that the set V_i is the *candidate set* for $\pi(i)$, $1 \le i \le n$. That is, we have $j \in V_i$ if—based on the query history so far—we have not ruled out that $\pi(i)$ equals j.

As noted in the main text, there is a trade-off between increasing the current best score s^* and reducing the sizes of the candidate sets $V_1, \ldots, V_{s^*+1}$. The main idea behind Algorithm 1 is to alternate between the two. More precisely, we maintain a string x and an integer s with $f(x) \ge s$ and have already reduced some elements from V_1 to V_s. The sets V_{s+1} to V_n are still equal to $[n]$. Initially, $s = 0$ and x is arbitrary. The sets V_1 to V_s are arranged into $t + 2$ levels.

Algorithm 1: The $O(n \log\log n)$ strategy for the HiddenPermutation problem with n positions.

Input: Number of levels t. Capacities $x_1, \ldots, x_t \in \mathbb{N}$ 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
Choose $x \in \{0,1\}^n$ uniformly at random and query $f(x)$;
$J \leftarrow \emptyset$;
while $|J| < q$ **do**
    $J' \leftarrow$ *Advance*(t);
    Reduce the size of the sets V_j with $j \in J'$ to 1 calling `subroutine2`$(x_t, J', 1, x)$;
    $J \leftarrow J \cup J'$;
Identify the remaining q bits from x and $y := x \oplus \left([n] \setminus \cup_{j=1}^{s} V_j\right)$ using the binary search algorithm `subroutine1` ; //phase 2

where *Advance* is the following function.
Advance(level ℓ) //returns a set J of up to x_ℓ indices such that $|V_j| \leq n/x_\ell^d$ for all $j \in J$
$J \leftarrow \emptyset$;
while $|J| \leq x_\ell$ **do**
    if $\ell = 1$ **then**
        Create y from x by flipping all bits in $[n] \setminus \cup_{j=1}^{s} V_j$ and query $f(y)$;
        if $f(x) > s$ and $f(y) = s$ **then** swap x and y;
        if $f(x) = s$ and $f(y) > s$ **then**
            $s \leftarrow s + 1$;
            $V_s \leftarrow$ `subroutine1`$(x, y, n/x_1^d)$; //Reduce $|V_s|$ to n/x_1^d
            $J \leftarrow J \cup \{s\}$;
            $x \leftarrow y$; //establishes $f(x) \geq s$
        else
            break ; //failure on level 1
    else
        $J' \leftarrow$ *Advance*$(\ell - 1)$;
        if $J' \neq \emptyset$ **then**
            Reduce the size of the sets V_j with $j \in J'$ to n/x_ℓ^d using `subroutine2`$(x_{\ell-1}, J', n/x_\ell^d, x)$;
            $J \leftarrow J \cup J'$;
        else
            break ; //failure on level ℓ
return J;

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; all candidate sets on level $t+1$ are singletons; i.e., we have identified the corresponding $\pi(i)$ value. On level i, $1 \leq i \leq t$, we can have up to x_i sets. We also say that the *capacity* of level i is x_i. The size of any set on level i is at most n/x_i^d, where $d \geq 1$ is some parameter to

be fixed later. Once x_i sets have been added to the ith level, we reduce the size of each of them to at most n/x_{i+1}^d using `subroutine2`, which we describe in Section B.3. The reduced sets are added to the $(i+1)$st level. `subroutine2` essentially allows us to simultaneously reduce the sizes of k sets to a kth fraction of their original size using at most $O(k)$ queries. Successive calls to `subroutine2` will reduce the sizes from at most n/x_i^d to at most n/x_{i+1}^d. After moving the sets from level i to level $i+1$, we either start filling the ith level again (in case the $(i+1)$st level has some capacity left), or we reduce the sizes of the sets on the $(i+1)$st level, thus adding them to the $(i+2)$nd one. Once the tth level contains x_t sets, we reduce the size of each of these sets to one, again employing `subroutine2`, and move them to level $t+1$. Thus $V_j = \{\pi(j)\}$ for each set V_j on level $t+1$.

At level one (calls *Advance*(1)) we attempt to advance the score s. Let s^* denote the current value of s and let x^* be the current bit string x. We ensure $f(x^*) \geq s^*$. We want to start working on V_{s^*+1}. We hope (this hope will be true with sufficiently high probability, as the analysis shows) that $\pi(s^*+1) \notin \cup_{j \leq s^*} V_j$. We create from x^* a new bit string y by flipping in x^* all bits that are *known* not to be the image of $1, \dots, s^*$ under π. That is, we set $y := x^* \oplus \left([n] \setminus \cup_{j=1}^{s^*} V_j\right)$ and query $f(y)$. If $f(y) = s^*$ and $f(x) > s^*$, we swap the two strings. This step is needed to ensure correctness of `subroutine1`.

If (after the potential swapping of the names) $f(y) > s^*$ and $f(x^*) = s$ holds, we know that $\pi(s^*+1) \in [n] \setminus \cup_{j=1}^{s^*} V_j$. In this case, we immediately update $V_{s^*+1} \leftarrow [n] \setminus \cup_{j=1}^{s^*} V_j$ and we reduce the size of the set V_{s^*+1} to n/x_1^d. This can be done by the binary search like algorithm, `subroutine1`, in at most $d \log x_1$ queries. This subroutine is described in Section B.2.

If, on the other hand, a *failure* happens; i.e., if in line 18 $f(y) = f(x^*) = s^*$ or $f(x^*) > s^*$ holds, then we know that $\pi(s^*+1)$ has not been flipped. Hence $\pi(s^*+1) \in \cup_{j=1}^{s^*} V_j$ must hold. Therefore, we must reduce the size of $\cup_{j=1}^{s^*} V_j$ to "free" $\pi(s^*+1)$. We immediately abort the first level by reducing the size of each of the V_is currently on that level to n/x_2^d, thus adding them to the second level. Should we still not make progress by flipping all bits in $[n] \setminus \cup_{j=1}^{s^*} V_j$, we continue by reducing the size of each level-2 set to n/x_3^d, and so on, until we eventually have $\pi(s^*+1) \notin \cup_{j=1}^{s^*} V_j$, in which case we can continue with the "increase the score by one, then reduce the size of the candidate sets"-procedure described above.

As such failures cause queries that neither help us to advance s^* nor to reduce the sizes of the candidate sets, it is essential to design the levels in such a way that the probability for such failures is small. As we shall see below, we achieve this by designing the levels such that their capacities grow exponentially. More precisely, we set $x_{i+1} = x_i^2$ for all $i = 1, \dots, t-1$. The first level has a capacity of $x_1 = \log n$ sets. Thus, $t = O(\log \log n)$ levels are sufficient if we want the last level, level t, to have a capacity that is linear in n. Our choice of the x_is also guarantees that x_i divides x_{i+1} and hence a call *Advance*(ℓ) returns no more than x_ℓ elements (exactly x_ℓ elements if the call ends without a failure).

Our strategy is formalized by Algorithm 1. In what follows, we first present the two subroutines, `subroutine1` and `subroutine2`. In Section B.4, we present the full proof of Theorem 4.

B.2 Subroutine 1

`subroutine1` is called by the function *Advance*(1). Simulating a randomized binary search, it allows us to reduce the number of potential candidates for $\pi(i)$ from some value $v < n$ to some value $\ell < v$ in $\log v - \log \ell$ queries.

Lemma 11. *Let $x, y \in \{0,1\}^n$ with $f(x) < f(y)$. Set $V := \{j \in [n] \mid x_j \neq y_j\}$ and $v := |V|$. Let $\ell \in \mathbb{N}$, $\ell < v$. Algorithm 2 reduces the size of V to ℓ using at most $\lceil \log v - \log \ell \rceil$ queries.*

Algorithm 2: The algorithm `subroutine1`(x, y, ℓ) reduces the size of the set $V_{f(x)+1}$ from v to ℓ in $\log v - \log \ell$ queries.

1 **Input:** Two strings $x, y \in \{0,1\}^n$ with $f(x) < f(y)$. An integer $\ell \in \mathbb{N}$.
2 **Initialization:** $V \leftarrow \{j \in [n] \mid x_j \neq y_j\}$; //potential candidates for $\pi(f(x)+1)$
3 **while** $|V| > \ell$ **do**
4 Uniformly at random select a subset $F \subseteq V$ of size $|V|/2$;
5 Create y' from y by flipping all bits in F and query $f(y')$;
6 **if** $f(y') > f(x)$ **then** $V \leftarrow V \backslash F$;
7 **else** $V \leftarrow F$;
8 **Output:** Set V of size at most ℓ.

Proof. For the correctness of the algorithm we note that, by definition, we have $x_{\pi(i)} = y_{\pi(i)} = z_{\pi(i)}$ for all $i \in [f(x)]$. Therefore, either we have $f(y') > f(x)$ in line 5 or we have $f(y') = f(x)$. In the former case, the bit $y_{\pi(f(x)+1)}$ was not flipped, and hence $\pi(f(x)+1) \notin F$ must hold. In the latter case the bit in position $\pi(f(x)+1)$ bit must have been flipped and we can infer $\pi(f(x)+1) \in F$.

The runtime bound is easily verified using the fact that the size of the set V halves in each iteration. □

It is now obvious how `subroutine1` (with the "select uniformly at random select"-statement in line 4 replaced by "arbitrarily select") yields a deterministic $O(n \log n)$ query algorithm for the HiddenPermutation problem. This is Algorithm 3.

Algorithm 3: A deterministic $O(n \log n)$ strategy for the HiddenPermutation problem.

1 **Initialization:** $x \leftarrow (0, \ldots, 0)$, $y \leftarrow (1, \ldots, 1)$;
2 Query $f(x)$ and $f(y)$;
3 **for** $i = 1, ..., n-1$ **do**
4 **if** $f(y) < f(x)$ **then** rename x and y;
5 $V \leftarrow$ `subroutine1`$(x, y, 1)$; //thereafter $V = \{\pi(i)\}$
6 Update x by flipping $\pi(i)$ and query $f(x)$;

For the correctness of this algorithm, observe that after the execution of the ith **for**-loop of Algorithm 3, the two strings x and y agree only in the positions $\pi(1), \ldots, \pi(i)$. Hence, either we have $f(x) = i$ and $f(y) > i$ or we have $f(y) = i$ and $f(x) > i$. In the latter case, x and y will be swapped in the next execution of the **for**-loop.

Note also that we apply this algorithm in the second phase of Algorithm 1 for identifying the last $\Theta(n/\log n)$ entries of z. This can be done as follows. When we leave the first phase of Algorithm 1, we have $|V_1| = \ldots = |V_q| = 1$ and $f(x) \geq q$. Create y from x by flipping all bits in $[n] \setminus \cup_{i=1}^{q} V_i$ and query $f(y)$. Then jump into the **for**-loop of Algorithm 3. This shows how to determine the remaining $q \in \Theta(n/\log n)$ bits of the target string z in at most a linear number of additional queries.

As mentioned, we call `subroutine1` also in the first level of Algorithm 1 (line 20) to reduce the size of $V_{f(x)+1}$ to n/x_1^d, or, put differently, to reduce the number of candidates for $\pi(f(x)+1)$ to n/x_1^d. As the initial size of $V_{f(x)+1}$ is at most n, this requires at most $d \log x_1$ queries by Lemma 11.

B.3 Subroutine 2

We describe the second subroutine of Algorithm 1, `subroutine2`. This routine is used to reduce the sizes of the up to $x_{\ell-1}$ candidate sets returned by a recursive call $Advance(\ell-1)$ from some value $\leq n/x_{\ell-1}^d$ to at most the target size of level ℓ, which is n/x_ℓ^d. As we shall see below, this requires an expected number of $O(1)x_{\ell-1}d(\log x_\ell - \log x_{\ell-1})/\log x_{\ell-1}$ queries. The pseudo-code of `subroutine2` is given in Algorithm 4. `subroutine2` performs a subtask of this reduction: It 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 `subroutine2` with parameter $k = x_{\ell-1}$ repeatedly to achieve the full reduction.

`subroutine2` is given a set J of at most k indices and a string y with $f(y) \geq \max J$. 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 kth fraction of the potential candidates for $\pi(j)$. That is, we randomly choose a subset $F_j \subseteq V_j$ of size $|V_j|/k$. We create a new bit string y' from y by flipping all candidate positions $\cup_{j\in J} F_j$. A condition on the sets $(V_j)_{j\in J}$ ensures that 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 V_j as the elements in F_j are no longer candidates for $\pi(j)$. 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 the candidate set V_j by the 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 further know that for all $h \in J$ with $h < j$ the set V_h was not hit. Thus, we can safely remove F_h from V_h, for $h < j$.

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 $(1-1/k)^{ck} \leq 1/2$. Once this is achieved, we are done with the current phase, and we move on to the next phase. To this end, let us 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.

We repeat Lemma 5.

Lemma 4. *Let $k \in \mathbb{N}$, let $J \subseteq [n]$ be a set of at most k indices with $V_j \cap V_p = \emptyset$ for $j \in J$ and $p \in [\max J]\backslash\{j\}$. Let $y \in \{0,1\}^n$ be such that $f(y) \geq \max J$, 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 Algorithm 4 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 V_js has been reduced to m by

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

as desired.

Algorithm 4: `subroutine2`(k, J, m, y). This subroutine is used in the main program to reduce 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.

Input: Positive integer $k \in \mathbb{N}$, a set $J \subseteq [n]$ with $|J| \leq k$, target size $m \in \mathbb{N}$ with $|V_j|/k \leq m$ for all $j \in J$, and a string $y \in \{0,1\}^n$ with $f(y) \geq \max J$. The sets $(V_j)_{j \in J}$ are pairwise disjoint. V_j, $j \in J$, is also disjoint from V_p for any $p \in [\max J] \setminus \{j\}$.
for $j \in J$ **do if** $|V_j| \leq m$ **then** delete j from J ; //V_j is already small enough
while $J \neq \emptyset$ **do**
    $o \leftarrow 0$; //counts the number of off-trials
    $\ell = |J|$; //$|V_j|/k \leq m$ for all $j \in J$
    repeat
        for $j \in J$ **do** Uniformly at random choose a subset $F_j \subseteq V_j$ of size $|V_j|/k$;
        Create y' from y by flipping in y the entries in positions $\cup_{j \in J} F_j$ and query $f(y')$;
        if $f(y') \geq \max J$ **then**
            $o \leftarrow o + 1$; //"off"-trial
            for $j \in J$ **do** $V_j \leftarrow V_j \setminus F_j$;
        else
            $V_{f(y')+1} \leftarrow F_{f(y')+1}$; //set $V_{f(y')+1}$ is hit
            for $j \in J$ **do if** $j \leq f(y')$ **then** $V_j \leftarrow V_j \setminus F_j$;
        for $j \in J$ **do if** $|V_j| \leq m$ **then** delete j from J;
    until $o \geq c \cdot k$ *and* $|J| \leq \ell/2$ //c is chosen such that $(1-1/k)^{ck} \leq 1/2$;
    $k \leftarrow k/2$;
Output: Sets V_j with $|V_j| \leq m$ for all $j \in J$.

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 $|J| \leq k$ always. Thus the probability of an off-trial is at least $(2e)^{-1}$ and hence the condition $o \geq ck$ holds after an expected number of $O(k)$ iterations.

As long as $|J| \geq \ell/2$, 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 $\ln(1-1/k) \leq -1/k$ we have $(1-1/k)^{\ell/2} = \exp(\ell/2 \ln(1-1/k)) \leq \exp(-\ell/(2k))$ and hence $1-(1-1/k)^{\ell/2} \geq 1 - \exp(-\ell/(2k)) \geq \ell/(2k)$. 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, this is at most half of its original size. Thus after replacing k by $k/2$ we still have $|V_j|/k \leq m$ for $j \in J$. □

Corollary 5. *Let $k \in \mathbb{N}$, J, and y be as in Lemma 4. Let further $d \in \mathbb{N}$ and $x \in \mathbb{R}$ such that $\max_{j \in J} |V_j| = n/x^d$. Let $y \in \mathbb{R}$ with $y > x$.*
Using at most $d(\log y - \log x)/\log k$ calls to Algorithm 4 we can reduce the maximal size $\max_{j \in J} |V_j|$ to n/y^d. The overall expected number of queries needed to achieve this reduction is $O(1)kd(\log y - \log x)/\log k$.

Proof. The successive calls can be done as follows. We first call `subroutine2`$(k, J, n/(kx^d), y)$. By Lemma 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 $n/(kx^d)$. We next call `subroutine2`$(k, J, n/(k^2x^d), y)$. After the hth such call we are left with sets of size at most $n/(k^h x^d)$. For $h = d(\log y -$

$\log x)/\log k$ we have $k^h \geq (y/x)^d$. The total expected number of queries at this point is $O(1)kd(\log y - \log x)/\log k$. □

Convention: In line 28 the pseudo-code of our main strategy, Algorithm 1, we write `subroutine2`$(x_{\ell-1}, J', n/x_\ell^d, x)$ if we appeal to `subroutine2` repeatedly, until the size of each V_j, $j \in J'$ has been reduced to n/x_ℓ^d. Similarly in line 9 we write `subroutine2`$(x_t, J', 1, x)$ to appeal to `subroutine2` repeatedly until the size V_j, $j \in J'$ has been reduced to one.

B.4 Proof of Theorem 4

It remains to show that the first phase of Algorithm 1 takes at most $O(n \log\log n)$ queries.

Theorem 6. *Let $q \in n - \Theta(n/\log n)$. Using Algorithm 1, we can identify positions $\pi(1), \ldots, \pi(q)$ and the corresponding entries $z_{\pi(1)}, \ldots, z_{\pi(q)}$ of z in these positions using at most $O(n \log\log n)$ queries.*

We prove Theorem 6. The proof of the required probabilistic statements is postponed to Sections B.5 and B.6.

If there is no failure in any call of *Advance*, the expected number of queries is bounded by

$$\begin{aligned}
&\frac{q}{x_t}\left(\frac{x_t \log n}{\log x_t} + \frac{x_t}{x_{t-1}}\left(\frac{x_{t-1} dc(\log x_t - \log x_{t-1})}{\log x_{t-1}}\right.\right.\\
&\qquad\left.\left. + \frac{x_{t-1}}{x_{t-2}}\left(\ldots + \frac{x_2}{x_1}\left(\frac{x_1 dc(\log x_2 - \log x_1)}{\log x_1} + x_1 d \log x_1\right)\right)\right)\right)\\
\leq & ndc\left(\frac{\log n}{\log x_t} + \frac{\log x_t}{\log x_{t-1}} + \ldots + \frac{\log x_2}{\log x_1} + \log x_1 - t\right), \qquad (4)
\end{aligned}$$

where c is the constant hidden in the $O(1)$-term in Corollary 5. To verify this formula, observe that each call of *Advance*(i) makes x_i/x_{i-1} calls to *Advance*$(i-1)$. After each such call we reduce the size of x_{i-1} candidate sets from n/x_{i-1}^d to n/x_i^d. By Corollary 5, this requires at most $x_{i-1} dc(\log x_i - \log x_{i-1})/\log x_{i-1}$ queries. The additional $x_1 d \log x_1$ term accounts for the queries needed to move the sets from level 0 to level 1; i.e., it accounts for the queries caused by the calls *Advance*(1) through which we reduce the sizes of the V_is by the randomized binary search algorithm, `subroutine1`, to n/x_1^d—requiring $d \log x_1$ queries per call. Finally, the term $x_t \log n/\log x_t$ accounts for the final reduction of the V_is to a set containing only one single element (at this stage we shall finally have $V_i = \{\pi(i)\}$). More precisely, this term is $(x_t(\log n - d \log x_t))/\log x_t$ 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*. In Sections B.5 and B.6 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 $n^2/((n-q)(x_i^{d-1}-1))$. By Corollary 5, each such level-i failure causes an additional number of at most $1 + x_i dc(\log x_{i+1} - \log x_i)/\log x_i$ queries; the 1 counts for the failured query in line 16; i.e., for the query through which we discover that $\pi(s^*+1) \in V_1 \cup \ldots \cup V_{s^*}$. Thus, in total we get an additional number of at most

$$\sum_{i=1}^{t} \frac{n^2}{(n-q)(x_i^{d-1}-1)}\left(1 + \frac{x_i dc(\log x_{i+1} - \log x_i)}{\log x_i}\right) \qquad (5)$$

expected queries caused by failures.

We recall the settings of the x_i. We set $x_1 := \log n$ and we choose the x_j such that $x_j = x_{j-1}^2$, $j = 2, \ldots, t$. We further require that $\log x_t = \Omega(\log n)$, which can be achieved by choosing

$t \in \Theta(\log\log n)$. In what follows, we do not specify the choice of d, but note that any choice $d \geq 4$ is good enough. With this parameter setting, formula (4) evaluates to

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

and, somewhat wasteful, we can bound formula (5) from above by

$$\frac{n^2dc}{n-q}\sum_{i=1}^{t} x_i^{-(d-3)} = O(n\log n)\sum_{i=0}^{t-1} x_1^{-(d-3)2^i} < O(n\log n)(x_1^{d-3}-1)^{-1} = O(n)\,,$$

where the first equation is by construction of the x_is, the inequality uses the fact that the geometric sum is dominated by the first term, and the last equality stems from our choice $x_1 = \log n$. This shows that the overall expected number of queries sums to $O(n\log\log n) + O(n) = O(n\log\log n)$.

Key to the failure analyses in Sections B.5 and B.6 is the following observation.

Lemma 7. *Each V_j has the property that $V_j \setminus \{\pi(j)\}$ is random; i.e., is a random subset of $[n] \setminus \{\pi(j)\}$ of size $|V_j| - 1$.*

Proof. V_j is initialized to $[n]$; thus the claim is true initially. In `subroutine1`, 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 `subroutine2`. □

B.5 Failures on Level One

Lemma 8. *A call of Advance(1) (i.e., lines 15 to 24) requires at most $x_1 + x_1 d\log x_1$ queries.*

Proof. The two occasions where queries are made are in line 16 and in line 20. Line 16 is executed at most x_1 times, each time causing exactly one query. As shown in Lemma 11, each call to `subroutine1` in line 20 causes at most $d\log x_1$ queries. The subroutine is called at most x_1 times. □

Lemma 9. *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(1) fails is at most $n(n-q)^{-1}\sum_{i=1}^{t} x_i^{-(d-1)}$.

Proof. A failure happens if we cannot increase the score of x by flipping the bits in $[n] \setminus \cup_{i=1}^{s} V_i$; i.e., if $f(y) \leq s$ holds in line 16 of Algorithm 1. This is equivalent to $\pi(s+1) \in \cup_{i=1}^{s} V_i$.

Let ℓ be the number of indices $i \in [n]$ for which V_i is on the last level; i.e., $\ell := |\{i \in [n] \mid |V_i| = 1\}|$ 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.

By our random construction of the V_is (cf. Lemma 7), the probability that $\pi(s+1) \in \cup_{i=1}^{s} V_i$ is at most $|\cup_{i=\ell+1}^{s} V_i|/(n-\ell)$ and hence bounded by $\sum_{i=\ell+1}^{s} |V_i|/(n-\ell)$.

At any time during the run of Algorithm 1, there are at most x_i sets V_j on the ith level. By construction, the size of each such level-i set is at most nx_i^{-d}. Hence, we can bound the probability that $\pi(s+1) \in \cup_{i=1}^{s} V_i$ from above by

$$\frac{n}{n-\ell}\sum_{i=1}^{t} x_i^{-(d-1)}\,.$$

□

By the exponential growth of the values $x_1, \ldots, x_t$ and the fact that we call line 16 a total number of $q < n$ times to improve upon the current best score, from Lemma 9 we immediately get the following.

Corollary 10. *The expected number of level* 1 *failures is less than*

$$qn(n-q)^{-1}(x_1^{d-1}-1)^{-1} \leq n^2(n-q)^{-1}\left(x_1^{d-1}-1\right)^{-1}.$$

B.6 Failures at Higher Levels

Lemma 11. *As in Lemma 9 let q be the number of indices i for which we determine $\pi(i)$ and $z_{\pi(i)}$ in the first phase. Let $i \in [t]$.*
The probability that a particular call of Advance(i) fails (level i failure) is at most $n(n-q)^{-1}\left(x_i^{d-1}-1\right)^{-1}$.

Proof. This proof is similar to the one of Lemma 9: A failure on level i occurs only if $\pi(s+1) \in \cup_{j=1}^{s} V_j$ and the size of each candidate set $V_1, \ldots, V_s$ has been reduced already to at most n/x_i^d. There are at most x_j candidate sets on each level $j \geq i$. By construction, the size of each candidate set on level j is at most n/x_j^d. By Lemma 7, the probability that $\pi(s+1) \in \cup_{j=1}^{s} V_j$ is at most

$$\frac{n}{n-\ell}\sum_{j=i}^{t}\frac{1}{x_j^{d-1}}, \tag{6}$$

where ℓ denotes again the number of sets that have been reduced to singletons already (i.e., the number of sets on level $t+1$).

By definition we have $x_j \geq x_i^{(2^{j-i})}$ and in particular we have $x_j \geq x_i^{j-i}$. Therefore expression (6) can be bounded from above by

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

□

By the same reasoning as in Section B.5, from Lemma 11 we immediately get the following.

Corollary 12. *Let $i \in [t]$.*
The expected number of level i failures is less than

$$nq(n-q)^{-1}(x_i^{d-1}-1)^{-1} \leq n^2(n-q)^{-1}(x_i^{d-1}-1)^{-1}.$$

C Details on the Lower Bound, Section 5

C.1 Candidate Sets in the Lower Bound Context

At node v of the decision tree, the candidate set V_i^v, intuitively corresponds to the possible values of $\pi(i)$. 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 $P_1^v = \{i \mid x_v[i] = 1\}$.[2] The precise definition of candidate sets is as follows:

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

Note that, as was the case with the candidate sets defined in Section 2, not all values in a candidate set could be valid. But as with the upper bound case, the candidate sets have some very useful properties. These properties are also slightly different from the ones observed before, due to the fact that some extra information has been announced to the query algorithm. We reiterate the definition of an active candidate set. 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 $F(x_u) = i-1$ and (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)*. The following theorem can be proved using similar ideas as in Theorem 1

Lemma 8. *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 $V^v_{\max_v+1}$ 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 by P_0^w while the other is intersected by 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 $P^u_{j-1 \bmod 2}$ while V_i^v is intersected with $P^u_{i \bmod 2}$. 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 $V_i^{v'} \subset V^{v'}_{\max_v+1}$. If $j < \max_v$, then $V^{v'}_{\max_v+1} = V^{w_j}_{\max_v+1}$ which means $V_i^{w_j} \subset V^{w_j}_{\max_v+1}$. If $j = \max_v$, then $V^{w_j}_{\max_v+1} = V^{v'}_{\max_v+1} \cap P^{v'}_{\max_v \bmod 2}$ and notice that in this case also $V_i^{w_j} = V_i^{v'} \cap P^{v'}_{i \bmod 2}$ which still implies $V_i^{w_j} \subset V^{w_j}_{\max_v+1}$.

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

Lemma 14. *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 $n_{\sigma'} = (|V_i^v| - k)n_\sigma$.*

[2] 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 $x = (x_1, \ldots, x_n)$

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 15. *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 16. *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.*

C.2 Potential Function Analysis

The main result of this section is the following.

Lemma 9. *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, $\mathbb{E}[\varphi(w_{\boldsymbol{i}_v}) - \varphi(v) \mid \Pi \in S_v] = O(1)$.*

We write

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

Before presenting the formal analysis of this potential function, we note that we have two main cases: $F(x_v) \leq \max_v$ 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 interesting part of the analysis since the probability of suddenly increasing the score by α is extremely small (we will show that it is roughly $O(2^{-\Omega(\alpha)})$) which subsumes any significant potential increase for values of $a > \max_v$. As discussed, we divide the above summation into two parts: one for $a \leq \max_v$ and another for $a > \max_v$:

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

$$\sum_{a=0}^{\max_v} \Pr[F(x_v) = a \mid \Pi \in S_v](\varphi(w_a) - \varphi(v)) + \tag{7}$$

$$\sum_{a=1}^{n/3-\max_v} \Pr[F(x_v) = \max_v + a \mid \Pi \in S_v](\varphi(w_a) - \varphi(v)). \tag{8}$$

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.

C.2.1 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, $1 \leq i \leq |A_v|$, 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 17. *For $i \leq |A_v|$, $\Pr[F(x_v) = a_i - 1 | \Pi \in S_v \wedge F(x_v) > a_i - 2] = \varepsilon_i$.*

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 16 and the definition of ε_i. □

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

Lemma 18. *For $i \leq |A_v|$ 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,*

$$\Pr[F(x_v) = a_i - 1 | \Pi \in S_v] = \varepsilon_i \Pi_{j=1}^{i-1}(1 - \varepsilon_j)(1 - \varepsilon_j'), \tag{9}$$

$$\Pr[a_i \leq F(x_v) < a_{i+1} - 1 | \Pi \in S_v] = \varepsilon_i'(1 - \varepsilon_i)\Pi_{j=1}^{i-1}(1 - \varepsilon_j)(1 - \varepsilon_j'), \tag{10}$$

Proof. Using Lemma 17, 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, we get 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 (9) and (10) follow from combining these bounds with Lemma 17. The rest of the lemma follows directly from the definition of ε_i and the candidate sets. □

Lemma 19. *Let $b \in \{0,1\}$ be such that $b \equiv \max_v$ and let $k := |V_{\max_v + 1}^v(b)|$. Then,*

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

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 $V_{\max_v + 1}^v$ that satisfy this is k. However, $V_{\max_v + 1}^v$ contains Con_v candidate sets but since $V_{\max_v + 1}^v$ can only contain a candidate set V_i^v if $i \equiv \max_v$ (by Lemma 8), it follows from Lemma 14 that $\Pr[F(x_v) = \max_v | \Pi \in S_v \wedge F(x_v) > \max_v - 1] = (k - \mathrm{Con}_v)/(|V_{\max_v + 1}^v| - \mathrm{Con}_v)$. The lemma then follows from the previous lemma. □

Lemma 20. *Let $b \in \{0,1\}$ be such that $b \equiv \max_v$ and let $k := |V^v_{\max_v+1}(b)|$. Then,*

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

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,

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

Using induction, it is possible to prove that

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

which implies the lemma. □

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

Lemma 21. *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 $V^v_i = [n]$ for $i \geq \max_v + 2$ which means $i-1$ is the number of candidates sets V^v_j contained in V^v_i 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 $\sigma_j \in P^v_{b_j}$. Thus, it follows that there are $x_{b_i} - \lfloor i/2 \rfloor$ elements $s \in P^v_{b_i}$ such that the sequences $(\sigma_1, \cdots, \sigma_{i-1}, s)$ can be the prefix of a permutation in S_v. Thus by Corollary 15, 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 22. *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.

C.2.2 Bounding (7)

We have,

$$\varphi(w_a) - \varphi(v) = \log \frac{\log \frac{2n}{|V^{w_a}_{\max_{w_a}+1}|-\mathrm{Con}_{w_a}}}{\log \frac{2n}{|V^{v}_{\max_v+1}|-\mathrm{Con}_v}} + \sum_{j\in A_v} \log \frac{\log \frac{2n}{|V^{w_a}_j|}}{\log \frac{2n}{|V^v_j|}}. \tag{11}$$

When $a \le \max_v$ 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 (11) that for $a_i \le a < a_{i+1}-1$, all the values of $\varphi(w_a)-\varphi(v)$ will be equal. Thus,

$$\begin{aligned}
(7) \quad = \quad & \sum_{i=1}^{|A_v|} \Pr[F(x_v) = a_i - 1 | \Pi \in S_v](\varphi(w_{a_i-1}) - \varphi(v)) + & (12)\\
& \sum_{i=1}^{|A_v|} \Pr[a_i \le F(x_v) < a_{i+1} - 1 | \Pi \in S_v](\varphi(w_{a_i}) - \varphi(v)) + & (13)\\
& \Pr[F(x_v) = \max_v | \Pi \in S_v](\varphi(w_{\max_v}) - \varphi(v)). & (14)
\end{aligned}$$

Analyzing (12) We write (12) using (11) and Lemma 18(9). 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 get the following:

$$(12) = \sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j) \left(\sum_{j=1}^{i} \log \frac{\log \frac{2n}{|V^{w_{a_i-1}}_{a_j}|}}{\log \frac{2n}{|V^v_{a_j}|}} \right) =$$

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

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

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

$$\sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1}(1-\varepsilon_j) \left(\log \left(1 + \frac{\log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V^v_{a_i}|}} \right) + \sum_{j=1}^{i-1} \log \left(1 + \log \frac{1}{1-\varepsilon_j} \right) \right) \le$$

$$\sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1}(1-\varepsilon_j) \left(\log \left(1 + \frac{\log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V^v_{a_i}|}} \right) + \sum_{j=1}^{i-1} \log \frac{1}{1-\varepsilon_j} \right) =$$

$$\sum_{i=1}^{|A_v|} \varepsilon_i \log \left(1 + \frac{\log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V^v_{a_i}|}} \right) \prod_{j=1}^{i-1}(1-\varepsilon_j) + \tag{15}$$

$$\sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1}(1-\varepsilon_j) \left(\sum_{j=1}^{i-1} \log \frac{1}{1-\varepsilon_j} \right). \tag{16}$$

To bound (15), 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, $|J_{t,k}| \le 2^{k+1}$.

$$(15) = \sum_{t=0}^{\log n}\sum_{k=1}^{\log n}\sum_{i\in J_{t,k}}\varepsilon_i \log\left(1+\frac{\log\frac{1}{\varepsilon_i}}{\log\frac{2n}{|V_{a_i}^v|}}\right)\prod_{j=1}^{i-1}(1-\varepsilon_j) \le$$

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

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

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

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

Now define $S_{t,r} = \sum_{2^{r-1}\le k<2^r} s_{t,k}$. Remember that $\sum_{r=1}^{\log\log n} S_{t,r} < 2^{t+1}$. Thus,

$$(15) \le O(1) + \sum_{t=0}^{n}\sum_{r=1}^{\log\log n}\frac{S_{t,r}\log\frac{2^{r-1}}{S_{t,r}}}{2^{r-1}}e^{-2^t+1} =$$

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

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

To bound (16), 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,

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

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 $i \ge s_i$, we get for $i \in J_t$,

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

Thus, we can write

$$(16) = \sum_{i=1}^{|A_v|}\varepsilon_i\frac{\log p_i}{p_i} = \sum_{t=0}^{\log n}\sum_{i\in J_t}\varepsilon_i\frac{\log p_i}{p_i} = \sum_{t=0}^{\log n}\sum_{i\in J_t} O\left(\frac{\varepsilon_i 2^t}{2^{2^{t-4}}}\right) \le \sum_{t=0}^{\log n} O\left(\frac{2^{2t+1}}{2^{2^{t-4}}}\right) = O(1).$$

To bound (16), 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

$$(16) = \sum_{i=1}^{|A_v|} \varepsilon_i \frac{\log p_i}{p_i} = \sum_{t=0}^{\log n}\sum_{i\in J_t} \varepsilon_i \frac{\log p_i}{p_i} = \sum_{t=0}^{\log n}\sum_{i\in J_t} O\left(\frac{\varepsilon_i 2^t}{2^{2^{t-4}}}\right) \leq \sum_{t=0}^{\log n} O\left(\frac{2^{2t+1}}{2^{2^{t-4}}}\right) = O(1).$$

Analyzing (13) The analysis of this equation is very similar to (12). We can write (13) using (11) and (10). We also use the same technique as in analyzing (16).

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

Let $s_i := \sum_{j=1}^{i} \varepsilon_j$, and $s'_i := \sum_{j=1}^{i-1} \varepsilon'_j$. Similar 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

$$(13) \leq \sum_{t=0}^{\log n}\sum_{i\in J_t} \varepsilon'_i \frac{\log p_i}{p_i p'_i} \leq \sum_{t=0}^{\log n}\sum_{i\in J_t} \varepsilon'_i \frac{s_i}{8\cdot 2^{s_i/8}2^{s'_i/8}} \leq \sum_{t=0}^{\log n}\sum_{i\in J_t} \varepsilon'_i \frac{2^{t+1}}{8\cdot 2^{(2^t-1)/8}} \leq \sum_{t=0}^{\log n} \frac{2^{2t+2}}{8\cdot 2^{(2^t-1)/8}} = O(1).$$

Analyzing (14) Let $b \in \{0,1\}$ be such that $b \equiv \max_v$ and let $k := |V^v_{\max_v+1}(b)|$. We use Lemma 19. Observe that we still have $\mathrm{Con}_v = \mathrm{Con}_{w_{\max_v}}$. Thus we have,

$$(14) \leq \frac{(k-\mathrm{Con}_v)}{|V^v_{\max_v+1}|-\mathrm{Con}_v}\prod_{i=1}^{|A_v|}(1-\varepsilon_i)(1-\varepsilon'_i)(\varphi(w_{\max_v})-\varphi(v)) =$$

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

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

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

C.2.3 Bounding (8)

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

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

Thus, we get that

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

$$\sum_{a>\max_v}^{n/3}\Pr[F(x_v)=a\mid\Pi\in S_v] \quad \log\log\frac{2n}{|V^{w_a}_{a+1}|-\mathrm{Con}_{w_a}}+ \tag{17}$$

$$\sum_{a>\max_v}^{n/3}\Pr[F(x_v)=a\mid\Pi\in S_v] \quad \log\left(\frac{\log\frac{2n}{|V^{w_a}_{\max_v+1}|}}{\log\frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}}\right)+ \tag{18}$$

$$\sum_{a>\max_v}^{n/3}\Pr[F(x_v)=a\mid\Pi\in S_v] \quad \sum_{j\in A_v}\log\frac{\log\frac{2n}{|V_j^{w_a}|}}{\log\frac{2n}{|V_j^v|}}. \tag{19}$$

Using the previous ideas, it is easy to see that we can bound (19) as

$$(19) \leq \sum_{a>\max_v}^{n/3} \Pr[F(x_v)=a \mid \Pi \in S_v \wedge F(x_v)>\max_v] \cdot \Pr[F(x_v)>\max_v \mid \Pi \in S_v] \sum_{j\in A_v} \log \frac{\log \frac{2n}{|V_j^{w_a}|}}{\log \frac{2n}{|V_j^v|}} \leq$$

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

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

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

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

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

To analyze (18) by Lemma 20 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

$$(18) \leq \sum_{a>\max_v}^{n} \Pr[F(x_v)=a \mid \Pi \in S_v] \log \left(\frac{\log \frac{2n}{|V^v_{\max_v+1}|-k}}{\log \frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}}\right) =$$

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

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

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

It is left to analyze (17). Let $x_0 = |P_0^v|$ and $x_1 = |P_1^v|$. 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 22 we can write

(17) as follows:

$$
\begin{aligned}
(17) &\le \sum_{a=1}^{n/3-\max_v} \Pr[F(x_v) = \max_v + a \mid \Pi \in S_v] \log\log \frac{2n}{|V^{w_{\max_v+a}}_{\max_v+a+1}| - \mathrm{Con}_{w_{\max_v+a}}} \\
&\le \sum_{a=1}^{n/3-\max_v} 2^{-\Omega(a)} \Big(1 - \frac{x_{b_{\max_v+a+1}} - \lfloor(\max_v + a + 1)/2\rfloor}{n - \max_v - a}\Big) \log\log \frac{2n}{n - x_{b_{\max_v+a+1}} - \lfloor(\max_v + a)/2\rfloor} \\
&= O(1).
\end{aligned}
$$

ECCC ISSN 1433-8092

http://eccc.hpi-web.de