Approximate degree lower bounds for oracle identification problems

Mark Bun∗ Nadezhda Voronova†

March 2023

Abstract

The approximate degree of a Boolean function is the minimum degree of real polynomial that approximates it pointwise. For any Boolean function, its approximate degree serves as a lower bound on its quantum query complexity, and generically lifts to a quantum communication lower bound for a related function.

We introduce a framework for proving approximate degree lower bounds for certain oracle identification problems, where the goal is to recover a hidden binary string \( x \in \{0,1\}^n \) given possibly non-standard oracle access to it. We apply this framework to the ordered search and hidden string problems, proving nearly tight approximate degree lower bounds of \( \Omega(n/\log^2 n) \) for each. These new lower bounds are driven by randomized communication upper bounds for the greater-than and equality functions.

1 Introduction

In an oracle identification problem, there is an unknown string \( x \in \{0,1\}^n \). A query algorithm is given possibly non-standard oracle access to \( x \), and its goal is to reconstruct \( x \) by making a minimal number of queries to this oracle. More specifically, an oracle identification problem is specified by a fixed family of Boolean functions \( a_1, \ldots, a_N \). A query algorithm may inspect any value \( a_i(x) \) of its choice at the cost of one query, and its goal is to determine \( x \). Many influential problems in the study of quantum algorithms and complexity can be viewed as oracle identification problems, including van Dam’s original oracle interrogation problem [vD98], the Bernstein-Vazirani problem [BV93], combinatorial group testing [AM14, Bel15], symmetric junta learning [Bel15], and more [BdW99, AIK+04, AIK+07, INRT12, CIG+12, Kot14]. In this work, we study two such oracle identification problems:

Ordered Search. Consider the following abstraction of the problem of searching an ordered list of \( N = 2^n \) elements. Given a list of \( N \) bits \( y_i \in \{0,1\} \) under the promise that \( y_0 \leq y_1 \leq \cdots \leq y_{N-1} \), find the (binary encoding of the) minimum index \( x \in \{0,1\}^n \) such that \( y_x = 1 \). Binary search yields a deterministic algorithm making \( n \) queries, and it is not hard to see that this is optimal for randomized algorithms as well. As for quantum algorithms, it turns out that a constant-factor speedup is possible [FGGS99, CLP07, BH08], but a lower bound of \( \Omega(n) \) holds in this model.
as well [BdW99, FGGS98, Amb99, HNS02, CL08]. Ordered search may be viewed as an oracle identification problem where the query algorithm is given oracle access to \( y_0 = \text{GT}_0(x), \ldots, y_{N-1} = \text{GT}_{N-1}(x) \), where each “greater-than” function \( \text{GT}_i(x) \) evaluates to 1 if \( i \geq x \) and to 0 otherwise.

**Hidden String.** In the hidden string problem, the goal is to reconstruct a hidden string \( x \in \{0, 1\}^n \) given information about the presence of absence of potential substrings of \( x \). That is, the goal is to determine \( x \) given “substring oracle” access, i.e., oracle access to \( y_s = \phi_s(x) \) for every binary string \( s \) of length at most \( n \), where \( \phi_s(x) \) evaluates to 1 iff \( s \) is a substring of \( x \). Building on a classical query algorithm of Skiena and Sundaram [SS95], Cleve et al. [CIG+12] gave a \( 3n/4 + o(n) \) quantum query algorithm for this problem, and proved a nearly matching quantum query lower bound of \( \Omega(n/\log^2 n) \).

The state-of-the-art quantum query lower bounds for both problems are proved via the quantum adversary method, which in its modern formulation [HLS07], characterizes the bounded-error quantum query complexity of every function up to a constant factor [Rei11]. The other major technique for proving quantum query lower bounds is the polynomial method [BBC+01], which lower bounds the quantum query complexity of a function by lower bounding its approximate degree. The approximate degree of a Boolean function is the least degree of a real polynomial that approximates it pointwise to error 1/3. Since the acceptance probability of a \( T \)-query quantum algorithm is a polynomial of degree \( 2T \), the approximate degree of a function is always at most (half of its) quantum query complexity, but it can be much smaller [Amb06, ABK16, She20, BKT20].

In this work, we prove lower bounds of \( \Omega(n/\log^2 n) \) on the approximate degree of (decision variants) of the ordered search and hidden string problems. These lower bounds are nearly optimal, as the known quantum (indeed, even classical) query algorithms for these problems automatically yield \( O(n) \) upper bounds on their approximate degree. Our lower bound on the approximate degree of the hidden string problem implies a quantum query lower bound matching the state-of-the-art [CIG+12].

Approximate degree is a fundamental measure of the complexity of Boolean functions that has been the subject of extensive study in its own right (see, e.g., [BT22] for a recent survey). And while nearly tight quantum query lower bounds for these problems were already known, we see two main quantum motivations for recovering these bounds via approximate degree. First, there are senses in which approximate degree is a more robust lower bound technique than the adversary method. For example, via Sherstov’s pattern matrix method [She11], any approximate degree lower bound for a Boolean function \( f \) can be “lifted” to give the same quantum communication lower bound for a related two-party function \( F \). Such a generic lifting result is not known for any other general quantum query lower bound technique. Moreover, variants of the polynomial method are capable of proving lower bounds against zero-, small-, and unbounded-error quantum algorithms [BBC+01, BCdWZ99], as well as time-space tradeoffs [KSdW07]. In the particular case of ordered search, Childs and Lee [CL08] explicitly posed the question of investigating approximate degree lower bounds to circumvent limitations of the adversary method.

Second, we believe that our approximate degree lower bounds shed additional light on what makes the ordered search and hidden string problems hard, and may be more transparent in this regard than existing adversary lower bounds. In particular, our lower bounds show that it is not only hard for quantum algorithms to reconstruct the hidden string \( x \), but even to simply compute its parity (a decision problem). The other nearly tight lower bounds for the problems we consider appear to make essential use of the fact that the query algorithm needs to reconstruct all of \( x \), and
it isn’t clear (at least to us) how to adapt them to hold for their decision variants. We believe that the technique we introduce, or at least the “indirect” method we use to prove our lower bounds, will be more broadly useful in understanding the approximate degree and quantum query complexity of other oracle identification problems.

1.1 Techniques

Here we give a brief summary of the ideas behind our lower bound for ordered search. A more detailed technical overview, including a discussion of how we apply our framework to the hidden string problem, appears in Section 2. Full proofs appear in Sections 3 and 4.

The first lower bound for quantum ordered search was given by Buhrman and de Wolf [BdW99], who actually showed an $\Omega(\sqrt{n})$ lower bound on its approximate degree. The starting point for the proof of our lower bound is their ingenious indirect argument, so let us review it here. Recall that the ability to solve ordered search on inputs $y_0 \leq y_1 \leq \cdots \leq y_{N-1}$ enables recovering the string $x$ for which every $y_i = \text{GT}_i(x)$. This, in particular, enables the evaluation of any “hard” Boolean function of $x$, e.g., its parity. In light of this, define the partial Boolean function $\text{OS}_N(y_0, \ldots, y_{N-1}) := \text{parity}(x)$ whenever there exists an $x$ for which $y_i = \text{GT}_i(x)$ for every $i$. Let $p : \{0,1\}^N \to \mathbb{R}$ be a polynomial of degree $d$ approximating $\text{OS}_N$. It is known that every polynomial approximating parity must have degree $\Omega(n)$, so the goal now is to use this fact to prove a lower bound on the degree of $p$. To do so, we use the additional fact that the functions $\text{GT}_i$ can each be approximated by a degree $O(\sqrt{n})$ polynomial $q_i$ arising from, say, a variant of Grover search. By making $p$ “robust to noise” in its input without increasing its degree [BNRDrW07, She12], we get that the composed polynomial $p(q_0(x), \ldots, q_{N-1}(x)) \approx \text{parity}(x)$ and has degree $O(d\sqrt{n})$. Now the fact that the approximate degree of parity is $\Omega(n)$ implies that $d = \Omega(\sqrt{n})$.

In summary, the lower bound for $\text{OS}_N$ follows from the fact that we can express the function $\text{parity}(x) = \text{OS}_N(\text{GT}_0(x), \ldots, \text{GT}_{N-1}(x))$, where we have a lower bound on the approximate degree of parity and an upper bound on the approximate degree of GT. However, the lower bound gets stuck at degree $\Omega(\sqrt{n})$ because the functions $\text{GT}_i$ themselves require nontrivial degree $O(\sqrt{n})$ to approximate, and this is tight.

To get an improved lower bound of $\tilde{\Omega}(n)$ on the approximate degree of $\text{OS}_N$, we introduce the following idea to make GT behave as if it were easier to approximate by low degree polynomials, while preserving the hardness of parity. Given an input $x \in \{0,1\}^n$, we redundantly encode $x$ as a longer string $Y(x) \in \{0,1\}^m$ for some $m = \text{poly}(n)$. This encoding is chosen so that

- Access to $Y(x)$ instead of just $x$ itself makes each function $\text{GT}_i(x)$ approximable by a much lower degree polynomial. That is, for every $i$, there exists a polynomial $q_i$ of degree $\text{polylog}(n)$ such that $q_i(Y(x)) \approx \text{GT}_i(x)$ for every $x$.

- Even with access to $Y(x)$, the function $\text{parity}(x)$ remains hard to approximate. That is, for every polynomial $p$ of degree at most $n/\text{polylog}(n)$, we have that $p(Y(x))$ fails to approximate $\text{parity}(x)$.

We can now obtain our improved lower bound by applying Buhrman and de Wolf’s argument to the redundantly encoded inputs. Specifically, given a robust polynomial $p : \{0,1\}^N \to \mathbb{R}$ of degree $d$ approximating $\text{OS}_N$, we would have $p(q_0(Y(x)), \ldots, q_{N-1}(Y(x))) \approx \text{OS}_N(\text{GT}_0(x), \ldots, \text{GT}_{N-1}(x)) = \text{parity}(x)$ for every $x$. Our upper bound on the degrees of the $q_i$’s, together with our lower bound on the degree needed to approximate parity, imply that $d \text{ polylog } n \geq n/\text{polylog } n$, and hence $d \geq \tilde{\Omega}(n)$.
All that remains is to construct the appropriate encoding \( \mathcal{Y} \). Our approach is inspired by Nisan’s classic randomized communication protocol for computing the two-party greater-than function. The most helpful way to think about this protocol for our purposes is as follows. Suppose Alice and Bob hold strings \( a, b \in \{0, 1\}^n \) and their goal is to determine whether the natural number represented by \( a \) is at least that represented by \( b \). They may do so by performing binary search to identify the minimum index \( j \) for which \( a_j \neq b_j \), at which point the answer is determined by which of \( a_j \) or \( b_j \) is 1. Each step of this binary search can be conducted by testing the equality of a substring of \( a \) with a substring of \( b \). Each equality test, in turn, may be performed (with high success probability) by comparing the inner products of \( a \) and \( b \) with a shared random string. The protocol requires \( \log n \) steps of binary search, and each equality test should be repeated \( O(\log \log n) \) times to achieve high success probability, giving an overall communication cost of \( O(\log n) \).

Now let us see how to turn this communication protocol into a polynomial approximating \( \text{GT}_i(x) \). Think of \( x \) as Bob’s input to the communication protocol, and of Bob’s role as passively computing an encoding \( \mathcal{Y}(x) \) that consists of many inner products of \( x \) with random strings. Now thinking of \( i \) as Alice’s input, she can compute \( \text{GT}_i(x) \) (with high probability) by repeatedly querying \( \mathcal{Y}(x) \) at the locations that correspond to the appropriate inner products from the protocol described above. This results in a \( O(\log n) \) randomized query algorithm for computing \( \text{GT}_i(x) \) from \( \mathcal{Y}(x) \), the success probability of which is a degree-\( \tilde{O}(\log n) \) polynomial in \( \mathcal{Y}(x) \).

The final step is to argue that even given \( \mathcal{Y}(x) \), consisting of many inner products of random strings with \( x \), the parity function \( \text{parity}(x) \) remains hard to compute. To see why this is true, note that a single inner product of \( x \) with a random bit string is itself a parity on a random subset of indices. That is, \( \mathcal{Y}(x) = (\text{parity}(x|S_1), \ldots, \text{parity}(x|S_m)) \) for random subsets \( S_1, \ldots, S_m \subseteq [n] \). The key observation then is, that a degree-\( d \) polynomial of these random parities is able to approximate the full \( \text{parity}(x) \) if and only if some degree-\( d \) polynomial of these random parities exactly computes \( \text{parity}(x) \), which in turn happens if and only if a symmetric difference of at most \( d \) of the sets \( S_1, \ldots, S_m \) yields the entire set of indices \([n] \). As a result, as long as neither the degree \( d \) nor the number of random inner products \( m \) is too large, we obtain that \( \text{parity}(x) \) cannot be approximated using \( \mathcal{Y}(x) \).\(^1\)

### 1.2 Further discussion

One of our initial motivations for studying the approximate degree of ordered search came from the preliminary version of Chattopadhyay et al. [CKLM17]. They showed that \( \text{OS}_N \circ \text{IP}_m^N \) has randomized communication complexity \( \Omega(\log N \cdot m) \), where \( \text{IP}_m \) is a two-party inner product (mod 2) gadget on \( m \)-bit inputs. This was done via an involved simulation argument, showing how a communication protocol for \( \text{OS}_N \circ \text{IP}_m \) could be used to construct a randomized decision tree for \( \text{OS}_N \). The techniques were specialized to the both the outer function and the inner function. Subsequent work [CFK+21] recovered this result using a generic simulation theorem. A direct application of Sherstov’s pattern matrix method [She11] to our result yields a quantum communication lower bound of \( \Omega(\log N / \log^2 \log N) \) on \( \text{OS}_N \circ g^N \) even for a constant-sized gadget \( g \).

Hoza [Hoz17] used ideas conceptually related to ours to nearly recover the known quantum query (but not approximate degree) lower bound for ordered search. Roughly, he used a Holevo-information argument to show that if an oracle identification problem specified by functions \( a_1, \ldots, a_N \)

\(^1\)In fact, this argument shows that it is impossible to approximate \( \text{parity}(x) \) to bounded error, but even to represent it in sign. This corresponds to a threshold degree lower bound.
can be solved with $T$ quantum queries, then $Q^*(A) \cdot T \gtrsim n$, where $A(i, x) = a_i(x)$ and $Q^*$ is the bounded-error two-party quantum communication complexity with shared entanglement. His quantum query lower bound for ordered search follows directly from the fact that the quantum communication complexity of the two-party greater-than function $\text{GT}$ on $n$-bit inputs is $O(\log n)$. However, without opening up the communication protocol for $\text{GT}$ as we do, it is not clear how to recover an approximate degree lower bound from his construction.

The idea of indirectly proving approximate degree lower bounds by combining a lower bound for one problem with an upper bound for another also appears in [BBGK18]. They gave a tight lower bound on the approximate degree of any function of the form $f \circ g^n$ where $f$ is an $n$-input symmetric function by combining a known lower bound for $\text{parity} \circ g^n$ [She12b] with a quantum query and approximate degree upper bound for the combinatorial group testing problem [Bel15].

We believe it should be possible to extend our techniques to prove new lower bounds for other oracle identification problems. A family of special cases of oracle identification is captured by the symmetric junta learning problem [AM14]. Here, there is a symmetric function $h : \{0, 1\}^k \to \{0, 1\}$ and each $f_S$ takes the form $f_S(x) = h(x|S)$. An important instance of this problem is the combinatorial group testing problem, wherein one takes $h = \text{OR}_k$. Belovs gave a tight upper bound of $O(\sqrt{k})$ [Bel15] for this problem. He also determined the query complexity for $h = \text{EXACT} - \text{HALF}$ to be $\Theta(k^{1/4})$ and gave an upper bound of $O(k^{1/4})$ for $h = \text{MAJ}$. These upper bounds were also (nearly) recovered algorithmically by Montanaro and Shao [MS20]. Despite its similarity to $\text{EXACT} - \text{HALF}$, no polynomial lower bound is known for the majority function $\text{MAJ}$.

In the counterfeit coin problem, there is a hidden string $x \in \{-1, 1\}^n$ with Hamming weight at most $k$. A query is parameterized by a balanced (i.e., having an equal number of 1’s and $-1$’s) string $y \in \{-1, 0, 1\}^n$, and indicates whether $\langle x, y \rangle$ is zero or non-zero. Iwama et al. [INRT12] gave a quantum algorithm making $O(k^{1/4})$ queries and conjectured this is tight, but no lower bound is known. Note that the oracle here is quite similar to the $\text{EXACT} - \text{HALF}$ oracle.

2 Technical ideas

2.1 Our lower bound framework

We begin with a somewhat more abstract description of our framework for proving approximate degree lower bounds for oracle identification problems. The main idea is to provide additional information about the hidden input to an oracle identification problem so as to selectively affect the ability of quantum query algorithms and approximating polynomials to compute the functions we wish to understand.

Recall that an oracle identification problem is specified by a family of functions $a_1, \ldots, a_N$. Given query access to the values $a_1(x), \ldots, a_N(x)$, the goal in our decision problems is to compute the function $\text{parity}(x)$. Suppose that we may identify $\text{parity}(x) = f(a_1(x), \ldots, a_N(x))$ for some function $f$. If we can construct a function $Y$ such that:

- Given $Y(x)$, every function $a_i(x)$ can be computed by a low-degree polynomial, but
- Given $Y(x)$, computing the parity of $x$ requires a high-degree polynomial,

Then by combining these two statements, we see that the function $f(a_1, \ldots, a_N)$ itself requires a high-degree polynomial. We apply this framework taking $f$ to be either the OS function or for
the “anchored hidden string” AHS function. The latter also implies a lower bound for the original (decisional) hidden string function HS described in the introduction.

In the following sections, we describe the main technical ideas that go into the proofs of our lower bounds. In order to provide more intuition about the structure of \( Y \), we describe the steps of constructing it for OS in detail before returning to the generalized framework.

### 2.2 Ordered search lower bound

First, notice that OS\(_N\) has the structure of an oracle identification problem since

\[
\text{OS}_N(\text{GT}_{0^n}(x), \text{GT}_{0^{n-1}}(x), \ldots, \text{GT}_{1^n}(x)) = \text{parity}(x)
\]

where \( N = 2^n \) and \( \text{GT}_i(x) = 1 \) if and only if \( x \leq i \) where \( i, x \in \{0,1\}^n \) if compared as numbers written in binary notation.

We want to show that there exists a function \( Y \) of \( x \) that we think of as revealing partial information about \( x \) such that:

- On one hand, for all \( i \in \{0,1\}^n \) there is an algorithm that makes a small number of queries to \( Y \) and can identify the value of \( \text{GT}_i(x) \) with constant probability of success. Note that a query-efficient algorithm automatically gives rise to a low-degree approximating polynomial.
- On the other hand, approximating the value of \( \text{parity}(x) \) given \( Y \) with a constant probability of success requires a lot of queries to \( Y \). Let us denote this auxiliary problem by \( \text{PUR}(Y) := \text{parity}(x) \).

It is helpful to think of \( Y \) itself as an oracle, whose output is given to a polynomial or to a query algorithm, whose goal is then to compute some other function of \( x \). We describe how we construct oracle \( Y \) through several attempts.

Let us first focus on constructing an oracle \( Y \) that meets the first condition. To do so, we can use the idea behind the \( O(\log n \log \log n) \)-bit communication protocol\(^2\) for the two-party communication problem \( \text{GT} \) to obtain an efficient randomized query algorithm for every function \( \text{GT}_i \). In the \( \text{GT} \) communication problem, Alice and Bob both get a string of \( n \) bits and the goal is to decide if the number represented by Alice’s string is greater than the number represented by Bob’s string.

In this randomized communication protocol for \( \text{GT} \), Alice checks if the first halves of the inputs are equal and depending on the answer, she either recursively continues on the first halves of the inputs or the second halves. By doing so, she finds the most significant bit where the inputs differ. To perform each equality check, both Alice and Bob compute the inner products modulo 2 of each of the inputs with the same set of some \( \alpha \) (publicly) random strings, Bob sends his values to Alice, and Alice compares these values to the values she obtained. If the original values were equal, then the inner products will be always equal, and otherwise, at least one pair of inner products will be unequal with high probability for sufficiently large \( \alpha \). This elementary operation (i.e., the ability to compute inner products with random strings) will be exactly what we want our oracle \( Y \) to be useful for.

\(^2\)A more efficient \( O(\log n) \)-bit communication protocol is known and underlies our sharpest result for ordered search. We discuss it in Sections 2.4 and 3.
First attempt. We will eventually give a randomized construction of the oracle $\mathcal{Y}$, and to this end, think of it as taking as input both the hidden string $x$ and a random input $r$. Let $\mathcal{Y}(r,x)$ be a function that takes a collection of $m$ $n$-bit strings $r \in \times_{i \in [m]} \{(0,1)^n\}$ and $x \in \{0,1\}^n$, and outputs $m$ bits, each representing the inner product of $r_i$ with $x$: $(\mathcal{Y}(r,x))_i = \langle r_i, x \rangle$. 

Our first attempt, however, will make no use of randomness at all. Let us consider $\mathcal{Y}(r,x)$ where $r$ consists of all possible strings of length $n$. That is, the output of the oracle consists of $\langle x, r_i \rangle$ for every $r_i \in \{0,1\}^n$.

Let us now see how to construct a query algorithm $C_i$ that, given oracle access to $\mathcal{Y}(r,x)$, computes $\text{GT}_i(x)$ with high probability. This algorithm emulates Alice’s side in the communication protocol, fixing his input to $i$. It samples random strings used in the communication protocol, computes the inner products of $i$ with these random strings on its own, and asks the oracle (emulating Bob) for the inner products of $x$ with the same random strings.

From the correctness of the communication protocol for $\text{GT}$ we can conclude that for all $x, i \in \{0,1\}^n$

$$\Pr_{r_1, \ldots, r_{\alpha \log n}} [C_i(\mathcal{Y}(r,x)) \neq \text{GT}_i(x)] < \log n \cdot 2^{-\alpha}$$

where $r_1, \ldots, r_{\alpha \log n}$ are the strings that $C_i$ sampled during the run, and $r$ is a collection of all $n$-bit strings. The number of queries is $\alpha \log n$.

Thus we see that this oracle satisfies the first condition: it helps to compute the $\text{GT}_i$ efficiently for every $i$ and $x$. But now there is a problem with the second condition: $\text{parity}(x) = \text{PUR}(\mathcal{Y})$ can be computed easily since $\text{parity}(x) = \text{PUR}(\mathcal{Y}(r,x)) = \langle x, 1^n \rangle$. So there is a 1-query algorithm (and hence a degree-1 polynomial) that exactly computes $\text{PUR}(\mathcal{Y}(r,x))$, violating our second condition.

Second attempt. Our goal now is to reduce the efficacy of the oracle $\mathcal{Y}$ in terms of how well it can be used by low-degree polynomials to approximate $\text{PUR}$. To do this, we instead consider a distribution over the potential oracles defined by the collection of strings used in the protocol. Let $r$ denote a sequence of the random strings that could appear in one run of $\text{GT}$ protocol described earlier. Let $\mathcal{R}$ denote the set of all such sequences. This allows us to define a distribution of oracles $\mathcal{Y}[\mathcal{R}](r,x)$, where $r \leftarrow \mathcal{R}$, and for us to consider a deterministic query algorithm. Let $B_{(r,i)}$ be a deterministic algorithm that is given access to the $\mathcal{Y}[\mathcal{R}](r,x)$ where $r \leftarrow \mathcal{R}$ is chosen uniformly at random, and which has the realization of $r$ and $i$ hardcoded into it. This algorithm is able to emulate the communication protocol (and the algorithm $C_i$), but now each time it needs a random string, it uses one provided in $r$.

From the correctness of the communication protocol for $\text{GT}$ we again can conclude that for all $x, i \in \{0,1\}^n$

$$\Pr_{r \leftarrow \mathcal{R}} [B_{(r,i)}(\mathcal{Y}[\mathcal{R}](r,x)) \neq \text{GT}_i(x)] < \log n \cdot 2^{-\alpha}.$$ 

So, with high probability, $B_{(r,i)}$ computes $\text{GT}_i(x)$ over the choice of the oracle $\mathcal{Y}[\mathcal{R}](r,x)$ for $r \leftarrow \mathcal{R}$.

Does this new oracle satisfy the second condition? Now an approximation to $\text{PUR}[\mathcal{R}](\mathcal{Y}[\mathcal{R}](r,x))$ needs to approximate $\text{parity}(x)$ when given a set of random parities from $\mathcal{R}$. Indeed, we show this requires high degree, as a consequence of the fact that high degree polynomial is necessary to construct the full parity of $x$ from random parities.
However, we need to add one more improvement to our structure. For every fixed $i, x$, the algorithm $B(r,i)$ when run on $Y[R](r,x)$ computes $GT_i(x)$ with high probability over $r \leftarrow \mathcal{R}$. But we need to switch quantifiers: we want an oracle that is “good” for all possible inputs for $GT$ simultaneously and, unfortunately, our current construction doesn’t give an algorithm computing $GT_i(x)$ for all $i, x \in \{0, 1\}^n$ using the same $r \leftarrow \mathcal{R}$.

**Third (and final) attempt.** So, is there a way to fix the source of randomness in a way that works for all possible inputs? Inspired by Newman’s theorem [New91] on simulating public randomness using private randomness in communication complexity, we show that there is. We show that by taking $t = O(n \delta^2)$ copies of $\hat{R}$, denoted $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_t$, we get a “good base” for the oracle. Consider a randomized algorithm $A(r,i)$ that, given access to $Y[R'](r,x)$ with $r \leftarrow \mathcal{R}' = \times_{j \in [t]} \mathcal{R}_j$, does the following:

- Sample $j \leftarrow [t]$ at random.
- Run $B(r,i)$ using the set $\mathcal{R}_j$ as the source of randomness.

Following the argument underlying Newman’s theorem, we show that this algorithm computes $GT_i(x)$ with $\log n \cdot 2^{-\alpha} + \delta$ failure probability. It works for every $i$ and $x$ and it still makes only $\alpha \log n$ queries to the oracle. If we put $\delta = \frac{1}{12}$ and $\alpha = O(\log \log n)$ then the probability of this algorithm failing for some input pair is at most $\frac{1}{6}$ with only $\alpha \log n = O(\log n \log \log n)$ queries to the oracle, i.e.,

$$\Pr_{r \leftarrow \mathcal{R}'}[A(r,i)(Y[R'](r,x)) \neq GT_i(x)] < \frac{1}{6}.$$ 

This change also doesn’t increase the “size” of the oracle (i.e., the number of queries it can answer) too much. This allows us to show that with high probability it is still impossible to combine the given partial parities to create the full parity using a low-degree polynomial, so the second condition is also satisfied. So there exists an oracle that allows computing the GT with low-degree polynomials but requires a high-degree polynomial to compute parity$(x)$ which is exactly what allows us to prove the lower bound on the approximate degree of $OS$.

### 2.3 Technical ideas behind the parity lower bound

Our technique relies on a lower bound on the approximate degree of parity$(x)$, or, more precisely, on the “Parity Under Randomness $\mathcal{R}$” function $\text{PUR}[\mathcal{R}](Y[\mathcal{R}](r,x))$ evaluates to to parity$(x)$ on input $Y[\mathcal{R}](r,x)$. We, in fact, prove a more general statement lower bounding the approximate degree of $\text{PUR}[\mathcal{R}]$ for a class of potential structures $\mathcal{R}$.

Specifically, we show that the parity function is hard, even to sign-represent, and even given access to $Y[\mathcal{R}]$ consisting of inner products of $x$ with random strings $r_i$ where each bit of $r_i$ is either fixed to zero or is an unbiased random bit. The only other restriction we need on $Y[\mathcal{R}]$ is that its “size”, i.e., the number of inner products it provides, is small. The bigger this number is, the worse our the lower bound becomes.

The proof idea is based on the hardness of sign-representing parity as described in [ABFR91], combined with the following combinatorial observation: given a set of $n$-bit strings (corresponding to samples from $\mathcal{R}$, and in turn to random inner products) where in every string each bit is either zero or is an unbiased random bit, with high probability no small subset of them adds up to the all-ones string (which corresponds to the parity function).
2.4 Improved ordered search and anchored hidden string lower bounds

Our generalized lower bound for approximating \( \text{PUR}[\mathcal{R}] \) allows us to obtain other lower bounds for oracle identification problems. For example, we give a slightly stronger lower bound for OS than what is implied by the discussion above. There is, in fact, a more efficient randomized communication protocol for GT that uses \( O(\log n) \) bits of communication. This can be converted into randomized query algorithm and thus into a polynomial of degree \( O(\log n) \). Moreover, this more efficient protocol is still based on computing equalities of substrings of inputs, and so the appropriate \( \mathcal{Y} \) has a very similar structure to the one described above while still satisfying the conditions of the generalized lower bound for \( \text{PUR} \). Moreover, the necessary “size” of \( \mathcal{Y} \) barely blows up at all. Putting everything together gives our improved lower bound of \( \Omega \left( \frac{n}{\log^2 n} \right) \) on the approximate degree of OS.

Using the same framework, we can also obtain a nearly tight lower bound on the approximate degree of the anchored hidden string problem AHS. In the anchored hidden string problem, the goal is to determine the parity of \( x \) given oracle access to \( y_{i,s} = \phi_{i,s}(x) \) for every index \( i \) and every binary string \( s \) of length at most \( n \), where \( \phi_{i,s}(x) = 1 \) iff the substring of \( x \) starting at index \( i \) matches \( s \). This oracle identification problem has the right form for our framework since

\[
\text{AHS}_N((\phi_{i,s}(x))_{i\in[n],s\in\{0,1\}^{\leq n-i+1}}) = \text{parity}(x).
\]

Moreover, each function \( \phi_{i,s}(x) \) simply computes the equality function of \( s \) with a substring of \( x \) of length \( |s| \) starting from position \( i \). As we have already seen, we can compute the equality function very efficiently given an oracle \( \mathcal{Y} \) of the right random structure, and such a \( \mathcal{Y} \) meets the conditions of our generalized lower bound for \( \text{PUR}[\mathcal{Y}] \). This directly implies a lower bound of \( \Omega \left( \frac{n}{\log n} \right) \) on the approximate degree of AHS.

Finally, the last lower bound described in this work is on the approximate degree of HS. This lower bound follows via a reduction from AHS. This reduction was first introduced in [CIG+12] in the quantum query model, but it holds for polynomial approximation as well.

3 Ordered search and generalized lower bound

In this section we give the formal proof of our lower bound on the approximate degree of ordered search. We show how our framework is used for this function and prove the generalized lower bound on parity that we later reuse for the hidden string problem.

3.1 Preliminaries

Our lower bounds on the approximate degree of (a decision version) of ordered search and the hidden string problem require the following definition of polynomial approximations for promise problems.

**Definition 1.** Let \( f : D \to \{0, 1\} \) where \( D \subseteq \{0, 1\}^n \) for some \( n \in \mathbb{N} \) be a partial Boolean function. For \( \frac{1}{2} > \varepsilon > 0 \), a polynomial \( p : \{0, 1\}^n \to \mathbb{R} \) is an \( \varepsilon \)-approximation to \( f \) if \( |p(x) - f(x)| \leq \varepsilon \) for every \( x \in D \) and \( -\varepsilon \leq p(x) \leq 1 + \varepsilon \) for all \( x \in \{0, 1\}^n \). The \( \varepsilon \)-approximate degree of \( f \), denoted \( \deg_{\varepsilon}(f) \), is the the least degree of a polynomial \( p \) that \( \varepsilon \)-approximates \( f \). We use the convention \( \deg(f) = \deg_{1/3}(f) \) to refer to the “approximate degree of \( f \)” without qualification.
That is, we require a polynomial approximation to a partial function defined on a domain $D$ to approximate the function on $D$ and remain bounded outside of $D$. Note that this is the type of approximation that arises from quantum query algorithms for promise problems.

We also formally define the ordered search function $\text{OS}$ and the family of greater-than functions $\text{GT}$.

**Definition 2.** For all $i \in \{0,1\}^n$ define the function $\text{GT}_i : \{0,1\}^n \to \{0,1\}$ to be the indicator of whether the value of the input is smaller than $i$: $\text{GT}_i(x) = 1$ if and only if $x \leq i$ where $i$ and $x$ are compared as numbers written in binary notation.

**Definition 3.** The ordered search function $\text{OS}_{2^n} : \{0^k12^n-k \mid k \in [2^n]\} \to \{0,1\}$ is a partial function defined the following way: $\text{OS}_{2^n}(0^k12^n-k) = \text{parity}(x)$ where $x \in \{0,1\}^n$ is the binary representation of $k$.

### 3.2 The notion of a good base.

In order to formally define the oracle, i.e. the source of additional information about the input, we introduce the notion of a “good base” for the oracle. A set $\mathcal{R}$, consisting of tuples of strings, is a good base if it’s constructed as follows.

Let $\mathcal{R}'$ be a Cartesian product of $m'$ subsets of $\{0,1\}^n$ where each subset $\mathcal{R}_j^\tau$ is itself defined by an $n$-bit string-template $\tau = \tau_1\tau_2\ldots\tau_n \in \{0,1\}^n$

$$\mathcal{R}_j^\tau = S_{\tau_1}S_{\tau_2}S_{\tau_3}\ldots S_{\tau_n}$$

where $S_0 = \{0\}$ and $S_1 = \{0,1\}$.

For example, if $\tau = 00100010$ then $\mathcal{R}_j^\tau = S_{\tau_1}S_{\tau_2}S_{\tau_3}\ldots S_{\tau_n} = S_0S_0S_1S_0S_0S_0S_0 = \{0\}\{0\}\{0,1\}\{0\}\{0\}\{0,1\}\{0\} = \{00000000,00000010,00100000,00100100\}$.

Let $\mathcal{B} = \{1_1\} \times \{1_2\} \times \ldots \times \{1_n\}$ where $1_j = 0^{i-j}10^{n-j}$ is the string that has the value 1 in $j$-th position and has the value 0 everywhere else. Let $\mathcal{R} = \mathcal{B} \times \mathcal{R}'$, and thus $\mathcal{R}$ is a Cartesian product of $m = n + m'$ subsets of $\{0,1\}^n$. Note that every $r \in \mathcal{R}$ is a $m$-tuple of $n$-bit strings:

$$r = (r_1, r_2, \ldots, r_m) = (1_1, 1_2, \ldots, 1_{n-1}, 1_n, r_{n+1}, r_{n+2}, \ldots, r_{n+m'})$$

where each $r_j$ is a string of length $n$, the first $n$ strings are fixed for all $r \in \mathcal{R}$, and the last $m'$ strings are from some sets $\mathcal{R}_j^\tau$ each for some template $\tau$. If $r \leftarrow \mathcal{R}$ is chosen u.a.r. then each $r_j, n < j \leq m'$ is chosen u.a.r. from some $\mathcal{R}_j^\tau$ and thus the subsequence of bits of $r_j$ corresponding to ones in $\tau$ is a uniformly random string, and the subsequence of bits of $r_j$ corresponding to zeros in $\tau$ is the all-zero string.

Any set $\mathcal{R}$ with the above structure will be called a good base of size $m$. Such an $\mathcal{R}$ is helpful for building our oracles as follows.

Let $\mathcal{Y}[\mathcal{R}] : \mathcal{R} \times \{0,1\}^n \to \{0,1\}^m$ be the following function: $(\mathcal{Y}[\mathcal{R}](r,x))_j = \langle r_j, x \rangle$ where $r_j$ is an $n$-bit string from the collection $r \in \mathcal{R}$ and the inner product is taken modulo 2. Note that $\mathcal{Y}[\mathcal{R}]$ is parameterized by $\mathcal{R}$, so for each good base $\mathcal{R}$ the function $\mathcal{Y}[\mathcal{R}]$ will be different. We will omit the parameter $\mathcal{R}$ later in places where it is clear from context.

Notice the following properties of this function $\mathcal{Y}[\mathcal{R}](r,x)$ that hold whenever $\mathcal{R}$ is a good base:

- For every $r \in \mathcal{R}$, the values $\mathcal{Y}(r,x)$ completely determine $x$. Since the first $n$ strings of $r$ are $1_1, 1_2, \ldots, 1_{n-1}, 1_n$, the first $n$ bits of $\mathcal{Y}(r,x)$ are exactly bits of $x$. 

10
• Given \( \mathcal{Y}(r, x) \) for \( r \leftarrow \mathcal{R} \) and \( r \) itself, one can compute (with some probability of error) if a subsequence of \( x \) specified by some pattern \( \tau \) agrees with some fixed string \( s \) in those indices.

To be more specific, if given \( \mathcal{Y}(r, x) \) for \( r \leftarrow \mathcal{R} \) and \( r \) itself, then one can check whether the strings \( x \land \tau \) (where \( \land \) denotes bitwise AND) and \( s \land \tau \) are equal for any \( s \in \{0, 1\}^n \) with one-sided error probability \( \frac{1}{2} \).

So, \( \mathcal{Y}[\mathcal{R}](r, x) \) could be used as an equality oracle for a fixed set of subsequences of \( x \) (predefined by \( \mathcal{R} \)) when \( r \) is chosen uniformly at random from \( \mathcal{R} \). Thus, \( \mathcal{Y}[\mathcal{R}](r, x) \) might give more information about \( x \) than \( x \) alone and might make some computations on \( x \) more efficient.

On the other hand, some functions of \( x \) remain “hard” even when given \( \mathcal{Y}(r, x) \). We will later show that \( \text{parity}(x) \) remains hard to compute even with this additional information.

3.3 Approximating polynomials for \( \text{GT}_i \)

We start our proof by showing that for some good base \( \mathcal{R}_{\text{OS}} \) the oracle \( \mathcal{Y}[\mathcal{R}_{\text{OS}}] \) could be used to make the computation of \( \text{GT} \) functions more efficient.

Claim 4. There exists a good base \( \mathcal{R}_{\text{OS}} \) of size \( m = O(n^2 \log \log n) \) such that if \( r \leftarrow \mathcal{R}_{\text{OS}} \) is sampled uniformly at random, then with probability at least \( \frac{2}{3} \) over the choice of \( r \) there exists a family of \( 2^n \) polynomials \( \{q_{(r,i)} : \{0,1\}^m \rightarrow \{0,1\} \mid i \in \{0,1\}^n \} \), each of degree at most \( 2 \log n \log \log n \), such that given \( \mathcal{Y}[\mathcal{R}_{\text{OS}}](r, x) \) as the input, each polynomial \( q_{(r,i)}(\mathcal{Y}[\mathcal{R}_{\text{OS}}](r, x)) \) approximates the corresponding \( \text{GT}_i(x) \) with error at most \( \frac{1}{6} \). That is,

\[
\Pr_{r \leftarrow \mathcal{R}_{\text{OS}}} \left[ \exists i, x \in \{0,1\}^n : \left| q_{(r,i)}(\mathcal{Y}(r, x)) - \text{GT}_i(x) \right| > \frac{1}{6} \right] < \frac{1}{3}.
\]

Proof. This proof consists of two parts: constructing a good base \( \mathcal{R}_{\text{OS}} \) and showing that it actually helps to compute every \( \text{GT}_i \).

Constructing the good base \( \mathcal{R}_{\text{OS}} \). We are going to construct \( \mathcal{R}_{\text{OS}} \) based on what random strings are useful in the communication protocol computing \( \text{GT} \) of two \( n \)-bit strings, \( x \) and \( i \). Intuitively, in this protocol, we first need to check if the first half of \( x \) and \( i \) are equal using a randomized communication protocol for equality. To do that we need to compute and compare \( \langle x, r \rangle \) and \( \langle i, r \rangle \), for some number \( \alpha \) of random strings \( r \) to be determined later, where each \( r \) is sampled from \( \{0,1\}^{\frac{3n}{4}} \). If the computed values \( \langle i, r \rangle = \langle x, r \rangle \) for all \( r \) we have considered, then we repeat this procedure on the second half of \( x \) and \( i \), which corresponds to computing and comparing \( \langle x, r \rangle \) and \( \langle i, r \rangle \) for \( \alpha \) random strings \( r \) sampled from \( \{0,1\}^{\frac{n}{2}} \). If, on the other hand, the values were not equal then we repeat this procedure on the first half of \( x \) and \( i \), which corresponds to computing and comparing \( \langle x, r \rangle \) and \( \langle i, r \rangle \) for \( \alpha \) random strings \( r \) sampled from \( \{0,1\}^{\frac{n}{4}} \). Since we want our oracle to be useful to emulate this procedure to compute \( \text{GT}_i(x) \), it should “contain” all the random strings used in this protocol.

Let \( \check{\mathcal{R}} = \mathcal{R}^{n/2} \times \mathcal{R}^{n/4} \times \mathcal{R}^{n/2} \times \cdots \times \mathcal{R}^{n/4} \times \mathcal{R}^{n/2} \times \mathcal{R}^{n/4} \). See Figure 1 for an illustration.

This \( \check{\mathcal{R}} \) describes all the strings as the source of randomness used in the \( O(\log n \log \log n) \) communication protocol for \( \text{GT} \), but each of the strings appears in the structure only once instead of \( \alpha \) times. So, we need to duplicate this structure \( \alpha \) times to properly simulate the protocol.
To finish the structure, we are going to add two other steps to the structure. First, we are going to have some number \( t \) of individual “prepackaged” copies to be determined later for the GT protocol. Let \( \mathcal{R}_1 = \ldots = \mathcal{R}_t = \times_{\alpha} \hat{\mathcal{R}} \). Each of the copies has enough randomness and the right structure of that randomness to simulate one full run of the GT protocol. Let \( \mathcal{R}' = \times_{j \in [t]} \mathcal{R}_j \) which allows us to handle \( t \) runs. Secondly, we want to be able to obtain the value of any specific index of \( x \), so we add a set of “basis” strings to the structure: \( \mathcal{B} = \{1_1\} \times \{1_2\} \times \ldots \times \{1_n\} = \{10\ldots0\} \times \{010\ldots0\} \times \ldots \times \{000\ldots010\} \times \{00\ldots01\} \).

The final underlying structure of the oracle will be a Cartesian product of \( \mathcal{R}' \) and \( \mathcal{B} \): \( \mathcal{R}_{\text{OS}} = \mathcal{B} \times \mathcal{R}' = \mathcal{B} \times (\times_{j \in [t]} \mathcal{R}_j) \). See Figure 2 for an illustration.

We also set the parameters to be \( \alpha = 2 \log (\log n), t = 250n \ln 2 \). Notice that this set \( \mathcal{R}_{\text{OS}} \) is a good base by construction and has size \( m = n + \alpha tn = n + ct^2 \log (\log n) \) for some constant \( c \).

**Constructing the family of approximating polynomials.** In order to prove this claim, we first describe a randomized query algorithm that computes \( \text{GT}_i(x) \) correctly for all \( i \) and \( x \) with high probability given \( \mathcal{Y}(\mathcal{R}_{\text{OS}})(r, x) \) as input. We then explain how to convert this query algorithm into a polynomial. The algorithm construction itself consists of two parts. In the first part, for all \( j \in [t] \) we show the existence of a deterministic algorithm \( B_{(r,j)} \) that, given \( \mathcal{Y}(r, x) \), can compute \( \text{GT}_i(x) \) for every specific \( x, i \in \{0,1\}^n \) with good probability over the choice of \( r \leftarrow \mathcal{R}_{\text{OS}} \), and this
algorithm is only going to use the parts of the input that correspond to $\mathcal{R}_j$ and $\mathcal{B}$. In the second part, we show that the algorithm $A_{(r,i)}$ that chooses a copy $j$ to use randomly and runs $B_{(r,i,j)}$, computes $\text{GT}_i(x)$ correctly for all $i$ and $x$ with high probability given $\mathcal{Y}(r,x)$ as input.

For all $i \in \{0,1\}^n$, $j \in [t]$, $r \in \mathcal{R}_{\text{OS}}$ let $B_{(r,i,j)}(\mathcal{Y}(r,x))$ be the following deterministic algorithm.

1. Set $\ell = 0$, $u = n/2$
2. While $\ell < u$:
3. Set $\tau = 0^\ell 1^{u-\ell} 0^{n-u}$
4. For all indices $v \in [m]$ corresponding to $n$-bit strings drawn from $\mathcal{R}^*$ within the $j$-th copy $\mathcal{R}_j$:
5. Compute $\langle i, r_v \rangle$ and compare it to $\langle \mathcal{Y}(r,x) \rangle_v = \langle x, r_v \rangle$.
6. If for all such $v$ the inner products are equal, i.e., $\langle i, r_v \rangle = \langle \mathcal{Y}(r,x) \rangle_v$, then set $\text{tmp} = u$, $u = u + (u - \ell)/2$, $\ell = \text{tmp}$ and go step 2.
7. Otherwise, set $u = (u + \ell)/2$ and go step 2.
8. Compare $i_\ell = \langle i, 1_\ell \rangle$ and $\langle \mathcal{Y}(r,x) \rangle_\ell = \langle x, 1_\ell \rangle = x_\ell$. If $x_\ell \leq i_\ell$ then accept. Otherwise, reject.

The last step is possible specifically because of $\mathcal{B}$ in the structure of $\mathcal{R}_{\text{OS}}$: $r_\ell = 1_\ell$ for all $\ell \leq n$ and for all $r \in \mathcal{R}_{\text{OS}}$. Notice that this algorithm emulates the randomized communication protocol for the GT communication problem.

In general, the algorithm emulates the randomized communication protocol for equality on the first half of the segment $[\ell, u + (u - \ell)]$ in $x$ and $i$, and depending on the result it splits the inputs into smaller segments and continues recursively. In the end, if all the runs of equality protocols were correct, the algorithm finds and compares the most significant bit where $x$ and $i$ differ.

By [NS03] we know that this algorithm computes $\text{GT}_i(x)$ with probability at least

$$1 \left(1 - (\log n)2^{-\alpha} = 1 - (\log n)2^{-2\log(\log n)} = 1 - \frac{1}{\log n} \geq \frac{11}{12}\right)$$

for sufficiently large $n$ independently of the choice of $j \in [t]$. That is, for all $j \in [t]$ and for all $i, x \in \{0,1\}^n$,

$$\Pr_{r \leftarrow \mathcal{R}_{\text{OS}}} [B_{(r,i,j)}(\mathcal{Y}(r,x)) = \text{GT}_i(x)] \geq \frac{11}{12}.$$

This algorithm makes at most $\alpha \log n = 2 \log n \log \log n$ queries to the oracle $\mathcal{Y}(r, x)$. Note that this algorithm needs access to the specific $r$ needed to compute every $\langle i, r_v \rangle$ and we enable this by “hardcoding” this $r$ into the algorithm and creating a separate algorithm for each possible $r$.

We have shown that for every fixed $i, x \in \{0,1\}^n$ there are many $r \in \mathcal{R}_{\text{OS}}$ that if used as a first input for the oracle $\mathcal{Y}$ allow $B_{(r,i,j)}$ to compute $\text{GT}_i(x)$. Unfortunately, this is not enough: our algorithm should be universal, i.e., we want a single algorithm that with high probability over $r$ succeeds on all $i$ and $x$. On the other hand, $B_{(r,i,j)}$ only uses one fixed “package” of random strings, namely the $j$-th package.

Let $W(i,x,r,j)$ be the indicator that the $j$-th package of random strings in $r$ defines a set of “bad” random strings for $(i,x)$: $W(i,x,r,j) = 1$ if and only if $B_{(r,i,j)}(\mathcal{Y}(r,x)) \neq \text{GT}_i(x)$. We established that $B_{(r,i,j)}(\mathcal{Y}(r,x))$ works well if given a random $r \leftarrow \mathcal{R}_{\text{OS}}$ for every $j \in [t]$. 
the probability of this algorithm outputting an incorrect answer is at most \( \frac{1}{12} \). So for all \( i, x \in \{0, 1\}^n, j \in [t] \), we have

\[
\Pr_{r \leftarrow \mathcal{R}_{\text{OS}}} [W(i, x, r, j) = 1] = \mathbb{E}_{r \leftarrow \mathcal{R}_{\text{OS}}}[W(i, x, r, j)] \leq \frac{1}{12}.
\]

We can’t immediately get a useful upper bound on the probability of \( r \leftarrow \mathcal{R} \) working out for all \( i \) and \( x \) at the same time. To achieve this, we’ll design a new algorithm that uses all \( t \) packages of random strings. Its construction and analysis are inspired by Newman’s classic argument used for simulating public randomness by private randomness in communication protocols.

For all \( i \in \{0, 1\}^n, r \in \mathcal{R}_{\text{OS}} \) let \( A(r, i) (\mathcal{Y}(r, x)) \) be the following randomized algorithm:

- Choose \( j \leftarrow [t] \) uniformly at random.
- Run \( B(r, i, j) (\mathcal{Y}(r, x)) \).

Let us now analyse \( A(r, i) \). The number of queries that \( A(r, i) \) makes to the oracle is the same as \( B(r, i, j) \) which is \( \alpha \log n = 2 \log n \log \log n \). We fix a pair \( (i, x) \) and evaluate the following probability.

\[
\Pr_{r \leftarrow \mathcal{R}_{\text{OS}}} \left[ \Pr_{j \leftarrow [t]} [B(r, i, j) \neq GT_i(x)] > \frac{1}{6} \right] = \Pr_{r \leftarrow \mathcal{R}_{\text{OS}}} \left[ \frac{1}{t} \sum_{j \in [t]} W(i, x, r, j) > \frac{1}{6} \right].
\]

We established that \( \mathbb{E}_{r \leftarrow \mathcal{R}_{\text{OS}}}[W(i, x, r, j)] \leq \frac{1}{12} \) and so by Hoeffding’s inequality,

\[
\Pr_{r \leftarrow \mathcal{R}_{\text{OS}}} \left[ \frac{1}{t} \sum_{j \in [t]} W(i, x, r, j) > \frac{1}{6} + \frac{1}{12} \right] \leq e^{-2 \frac{t}{144}} \leq 2^{-\frac{500n}{144}}.
\]

By a union bound over all possible \( i, x \in \{0, 1\}^n \),

\[
\Pr_{r \leftarrow \mathcal{R}_{\text{OS}}} \left[ \exists i, x \in \{0, 1\}^n : \frac{1}{t} \sum_{j \in [t]} W(i, x, r, j) > \frac{1}{6} \right] \leq 2^{2n} 2^{-\frac{500n}{144}} \leq 2^{-n} < \frac{1}{3}.
\]

Therefore, we have proven that

\[
\Pr_{r \leftarrow \mathcal{R}_{\text{OS}}} \left[ \exists i, x \in \{0, 1\}^n : \Pr_{j \leftarrow [t]} [A(r, i) (\mathcal{Y}(r, x)) \neq GT_i(x)] > \frac{1}{6} \right] < \frac{1}{3}.
\]

The last step is to convert this family of query algorithms into a family of approximating polynomials. Let \( q(r, i) \) denote the acceptance probability of \( A(r, i) \). A standard argument (e.g., [BdW02, Theorem 15]) implies that this is a polynomial of degree at most \( 2 \log n \log \log n \) such that

\[
\Pr_{r \leftarrow \mathcal{R}_{\text{OS}}} \left[ \exists i, x \in \{0, 1\}^n : |q(r, i) (\mathcal{Y}(r, x)) - GT_i(x)| > \frac{1}{6} \right] < \frac{1}{3},
\]

which is exactly what we were looking for.
We successfully converted the most well-known communication protocol for GT that requires $O(\log n \log \log n)$ bits of communication into a family of polynomials of degree $O(\log n \log \log n)$ that approximates GT$_i$. It’s known that there is a better communication protocol for GT that requires only $O(\log n)$ bits of communication, as observed by Nisan [Nis93]. The next claim establishes that this more efficient protocol can be converted into a family of polynomials as well.

**Claim 5.** There exists a good base $R_{\text{OS}++}$ of size $m = O(n^3 \log n)$ such that if $r \leftarrow R_{\text{OS}++}$ is sampled uniformly at random, then with probability at least $\frac{2}{3}$ over the choice of $r$ there exists a family of polynomials $\{q_{(r,i)} : \{0,1\}^m \rightarrow \{0,1\} \mid i \in \{0,1\}^n\}$, each of degree at most $O(\log n)$, such that given $Y(r,x)$ as the input, each polynomial $q_{(r,i)}(Y[r_{\text{OS}++}])(r,x)$ approximates the corresponding GT$_i(x)$ with error at most $\frac{1}{6}$. That is,

$$\Pr_{r \leftarrow R_{\text{OS}++}} \left[ \exists i, x \in \{0,1\}^n : |q_{(r,i)}(Y(r,x)) - \text{GT}_i(x)| > \frac{1}{6} \right] < \frac{1}{3}.$$ 

The proof of Claim 5 is similar to the proof of Claim 4 and can be found in Appendix A.

### 3.4 General lower bound

To complete the framework and to obtain the lower bound for Ordered Search we need to show why computing the parity is hard even given $Y[R_{\text{OS}}]$ or $Y[R_{\text{OS}++}]$. We will show a stronger lower bound that would allow us to reuse this lower bound for other applications. Specifically, we will show that computing the parity of input $x$ remains hard given $Y[R]$ for any good base $R$ of small size.

#### 3.4.1 Combinatorial claim

The hardness of parity in this model is based on the following statement. For every good base $R$ of small size with high probability over the sample $r \leftarrow R$ for every set of $n$-bit strings taken from the collection $r$ of size at most $O(\frac{n}{\log n})$, the bitwise parity of these strings is not equal to the all-ones string.

**Claim 6.** For every good base $R$ of size $m$ with probability at least $\frac{2}{3}$ over the choice of $r \leftarrow R$ for every set of elements $T \subseteq [m]$ of size at most $d = \frac{n}{4\log m} - 1$, the bitwise parity of $n$-bit strings $r_i, i \in T$ from the collection $r \leftarrow R$ is not equal to the all-ones string:

$$\Pr_{r \leftarrow R} \left[ \forall T \subseteq [m], |T| \leq d : \bigoplus_{i \in T} r_i \neq 1^n \right] \geq \frac{2}{3}.$$ 

**Proof.** Fix an arbitrary good base $R$ of size $m$. Fix a set $T \subseteq [m]$ where $|T| \leq d$. We want to bound the probability $\Pr_{r \leftarrow R}[\bigoplus_{i \in T} r_i = 1^n]$ that for this $r$ and for this $T$ the strings corresponding to the indices in $T$ sum up to the string of all ones. Fix a specific index $k \in [n]$. We compute the probability that index $k$ is set to 1 in $\bigoplus_{i \in T} r_i$. To do this we need to understand how the candidate strings $r_i, i \in T$ can influence this value.

There are three possible scenarios for each index $k$:

- (Type I) There is at least one string $r_i \in \{0,1\}^n$ with $i \in T$ such that it is chosen from $R^r$ where $\tau_k = 1$. Then in each such string, the bit at index $k$ is sampled independently at random with probability $\frac{1}{2}$. Thus $\Pr_{r \leftarrow R}[\bigoplus_{i \in T} r_i, 1_k] = \frac{1}{2}$. 

15
• (Type II) There are no strings $r_i, i \in T$ such that $r_i$ is chosen from $\mathcal{R}^T$ and $\tau_k = 1$, but there is $r_i, i \in T$ that is chosen from $\mathcal{B}$, such that $r_i = 1_k$. Then the value of $\langle \bigoplus_{i \in T} r_i, 1_k \rangle$ is one since there is exactly one string in this sum with the $k$th index value set to one. Thus $\Pr_{r \leftarrow \mathcal{R}}[\langle \bigoplus_{i \in T} r_i, 1_k \rangle = 1] = 1$.

• (Type III) There are no strings $r_i, i \in T$ such that $r_i$ is chosen from $\mathcal{R}^T$ and $\tau_k = 1$, and there is no $r_i, i \in T$ that is chosen from $\mathcal{B}$, such that $r_i = 1_k$. Then for all strings $r_i$ the index $k$ is $0$, so $\Pr_{r \leftarrow \mathcal{R}}[\langle \bigoplus_{i \in T} r_i, 1_k \rangle = 1] = 0$.

<table>
<thead>
<tr>
<th>Index $k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_{i_1}$</td>
<td>⋆</td>
<td>⋆</td>
<td>⋆</td>
<td>⋆</td>
<td>0</td>
<td>0</td>
<td>⋆</td>
<td>0</td>
</tr>
<tr>
<td>$r_{i_2}$</td>
<td>⋆</td>
<td>⋆</td>
<td>⋆</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_{i_3}$</td>
<td>⋆</td>
<td>0</td>
<td>⋆</td>
<td>⋆</td>
<td>0</td>
<td>0</td>
<td>⋆</td>
<td>0</td>
</tr>
<tr>
<td>$r_{i_4}$</td>
<td>0</td>
<td>⋆</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_{i_5}$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_{i_6}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>I</th>
<th>I</th>
<th>I</th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>I</th>
<th>III</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pr_{r \leftarrow \mathcal{R}}[\langle \bigoplus_{i \in T} r_i, 1_k \rangle = 1]$</td>
<td>1/2</td>
<td>1/2</td>
<td>1/2</td>
<td>1/2</td>
<td>1</td>
<td>0</td>
<td>1/2</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3: Example of index types, $T = \{i_1, i_2, i_3, i_4, i_5, i_6\}$.

Notice that $T$ fully defines the types of all indices and thus the values of $\langle \bigoplus_{i \in T} r_i, 1_k \rangle$ for $k$ of types II and III don’t depend on the choice of $r \leftarrow \mathcal{R}$. On the other hand, the values of indices of type I do depend on the choice of $r \leftarrow \mathcal{R}$. Each of them is either a parity of independent random bits or the negation of a parity of independent random bits which is fixed by $T$ too. Thus they behave as independent bits themselves and therefore the values $\langle \bigoplus_{i \in T} r_i, 1_k \rangle$ are mutually independent for all indices $k$.

Denote by $n_I, n_{II}, n_{III}$ the numbers of indices of each type. Notice that $n_I + n_{II} + n_{III} = n$ and $n_{II} \leq d$. Then in this notation

$$\Pr_{r \leftarrow \mathcal{R}} \left[ \bigoplus_{i \in T} r_i = 1^n \right] = \left( \frac{1}{2} \right)^{n_I} 1^{n_{II}} 0^{n_{III}}.$$ 

If there exists $k \in [n]$ of the third type, the probability $\Pr_{r \leftarrow \mathcal{R}}[\bigoplus_{i \in T} r_i = 1^n]$ becomes 0, so to upper bound the probability we may assume all the indices have one of the first two types. And, since $n_{II} \leq d$, to maximize the value we assume that $n_{II} = d$. Thus we have
Since \( d = \frac{n}{4\log m} - 1 \) and \( m \geq n \), we have \( n - d > \frac{n}{2} \) for sufficiently large \( n \). So for a fixed \( T \),

\[
\operatorname{Pr}_{r \leftarrow \mathcal{R}} \left[ \bigoplus_{i \in T} r_i = 1^n \right] = \left( \frac{1}{2} \right)^{n \cdot m} \leq 1^d \left( \frac{1}{2} \right)^{n - d} = 2^{-(n-d)}.
\]

There are \( \binom{m}{\leq d} \) ways to choose the set \( T \), so by a union bound over the choice of \( T \), the probability that for some set of size at most \( d \) the value \( \bigoplus_{i \in T} r_i \) is equal to the string of all ones is

\[
\operatorname{Pr}_{r \leftarrow \mathcal{R}} \left[ \bigoplus_{i \in T} r_i = 1^n \right] < 2^{-\frac{n}{2}}.
\]

3.4.2 Lower bound on the degree of \( \text{PUR}[\mathcal{R}] \)

For every good base \( \mathcal{R} \) and for every fixed \( r \in \mathcal{R} \) define the function \( \text{PUR}[\mathcal{R}]_r : D[\mathcal{R}]_r \to \{0, 1\} \) where \( D[\mathcal{R}]_r = \{ \mathcal{Y}[\mathcal{R}](r, x) \mid x \in \{0, 1\}^m \} \) is the subset of \( \{0, 1\}^m \) where each domain point corresponds to one specific \( x \in \{0, 1\}^n \) and is consistent with the fixed \( r \). This function outputs the parity of the string encoded by the input: \( \text{PUR}[\mathcal{R}]_r(\mathcal{Y}(r, x)) = \text{parity}(x) \). It is well defined since \( \text{parity}(x) = \bigoplus_{r \in \mathcal{R}} (x, r_i) = \bigoplus_{i \leq n} (\mathcal{Y}(r, x))_i \). Note that both \( D[\mathcal{R}]_r \) and \( \text{PUR}[\mathcal{R}] \) are parameterized by \( \mathcal{R} \) and, as with \( \mathcal{Y}[\mathcal{R}] \), we will omit the parameter later in places where the parameter is clear from the context.

Our goal is to show that \( \text{PUR}[\mathcal{R}] \) is hard to approximate if \( \mathcal{R} \) is a good base of small size. We do this by showing that for every good base \( \mathcal{R} \) of size \( m \) if \( r \leftarrow \mathcal{R} \) u.a.r. then every polynomial \( p \) of degree at most \( d = O(\frac{n}{\log m}) \) is completely uncorrelated with \( \text{PUR}[\mathcal{R}]_r(\mathcal{Y}(r, x)) \) with high probability over the choice of \( r \).

**Theorem 7.** For every good base \( \mathcal{R} \) of size \( m \) if \( r \leftarrow \mathcal{R} \) u.a.r. then with probability at least \( \frac{2}{3} \) over the choice of \( r \) every polynomial \( p : \{0, 1\}^m \to \mathbb{R} \) of degree at most \( d = \frac{n}{4\log m} - 1 \) doesn’t approximate \( \text{PUR}[\mathcal{R}]_r \):

\[
\operatorname{Pr}_{r \leftarrow \mathcal{R}} \left[ \forall \epsilon < \frac{1}{2}, \forall p, \deg(p) \leq d, \exists y \in D[\mathcal{R}]_r : |p(y) - \text{PUR}[\mathcal{R}]_r(y)| > \epsilon \right] \geq \frac{2}{3}
\]

Note that Theorem 7 rules out approximating polynomials that may be unbounded outside of the domain of \( \text{PUR}[\mathcal{R}]_r \). That is, it asserts that there is no low-degree approximating polynomial even when that polynomial is permitted to take values outside of \([0, 1]\) on points outside of the domain of \( \text{PUR}_r \). Note also that since the lower bound applies for all \( \epsilon < 1/2 \), it actually entails a threshold degree lower bound on computing \( \text{PUR}[\mathcal{R}] \).
**Proof.** Fix an arbitrary good base $R$ of size $m$.

For convenience in this proof, let us change notation to consider polynomials approximations over $\{-1, 1\}$ instead of over $\{0, 1\}$. Define $\mathcal{Y}': R \times \{-1, 1\}^n \rightarrow \{-1, 1\}^m$ to be $(\mathcal{Y}'(r, x'))_i = 1 - 2(\mathcal{Y}(r, (\frac{x_1}{2}, \frac{x_2}{2}, \ldots, \frac{x_n}{2})))_i = 1 - 2(x_i, r_i)$ where $r_i$ is the vector corresponding to $i$th component of $\mathcal{Y}(r, x)$ and $x \in \{0, 1\}^n$ is the vector that corresponds to $x' \in \{-1, 1\}^n$: $x_i = \frac{1-x_i}{2}$ for all $i \in [n]$. Notice that this change of notation satisfies the following: if $a \in \{0, 1\}$ and $a'$ is the corresponding value in the new notation $a' \in \{-1, 1\}$ then $a' = (-1)^a$.

Let’s also rewrite $\text{PUR}_r$ in this new notation. Let $D_r'$ represent the domain of $\text{PUR}_r'$: $D_r' = \{\mathcal{Y}'(r, x') \mid x' \in \{-1, 1\}^n\}$ and the function $\text{PUR}_r'(\mathcal{Y}_1', \mathcal{Y}_2', \ldots, \mathcal{Y}_m') = 1 - 2\text{PUR}_r((\frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2}))$. Note that every polynomial $p' : \{-1, 1\}^m \rightarrow \mathbb{R}$ that approximates $\text{PUR}_r'$ to error $\varepsilon$ can be converted by a linear transformation into a polynomial $p : \{0, 1\}^m \rightarrow \mathbb{R}$ of the same degree that approximates $\text{PUR}_r$ to error $\varepsilon/2$. So it suffices to prove that no polynomial $p'$ of degree at most $d$ approximates $\text{PUR}_r'$ to error $\varepsilon < 1$.

Assume toward a contradiction that there is a polynomial $p'$ of degree $d$ that approximates $\text{PUR}_r'$. This means that there exists $\varepsilon < 1$ such that for all $y' \in D_r'$,

$$|p'(y') - \text{PUR}_r'(y')| < \varepsilon.$$

Consider the following expression:

$$\frac{1}{2^n} \left| \sum_{y' \in D_r'} \text{PUR}_r'(y')(\text{PUR}_r'(y') - p'(y')) \right| \leq \frac{1}{2^n} \left( \max_{y' \in D_r'} |p'(y') - \text{PUR}_r'(y')| \right) \left( \sum_{y' \in D_r'} |\text{PUR}_r'(y')| \right) (1)$$

$$< \frac{1}{2^n} \varepsilon |D_r'| = \varepsilon.$$

The last equality holds because $\mathcal{Y}'(r, \cdot)$ is surjective, and hence $|D_r'| = 2^n$. On the other hand,

$$\frac{1}{2^n} \left| \sum_{y' \in D_r'} \text{PUR}_r'(y')(\text{PUR}_r'(y') - p'(y')) \right| = \frac{1}{2^n} \left| \left( \sum_{y' \in D_r'} \text{PUR}_r'(y')p'(y') \right) - \left( \sum_{y' \in D_r'} \text{PUR}_r'(y')p(y') \right) \right|$$

$$= \frac{1}{2^n} |D_r'| - \left( \sum_{y' \in D_r'} \text{PUR}_r'(y')p(y') \right).$$

$$\text{(2)}$$

We now show that with high probability the expression above is equal to $\frac{|D_r'|}{2^n}$.

**Claim 8.** With probability at least $\frac{2}{3}$ over the choice of $r \leftarrow R$, for every polynomial $p' : \{-1, 1\}^m \rightarrow \mathbb{R}$ of degree at most $d = \frac{n}{4\log m} - 1$ we have

$$\sum_{y' \in D_r'} \text{PUR}_r'(y')p(y') = 0.$$
Proof. Fix a polynomial $p'$ of degree at most $d = \frac{n}{4 \log m} - 1$. By linearity it suffices to consider the case where $p'$ is a monomial, $p'(y') = \prod_{j \in T} y'_j$ for some $T \subseteq [m], |T| \leq d$. So

$$\sum_{y' \in D'_r} \text{PUR}_r(y')p'(y') = \sum_{x \in \{-1,1\}^n} \left( \prod_{i \in [n]} (x'_i) \right) \left( \prod_{j \in T} (y'(r,x'))_j \right) = \sum_{x \in \{0,1\}^n} (-1)^{\langle x,1^n \rangle} \left( \prod_{j \in T} (-1)^{\langle x,r_j \rangle} \right)$$

$$= \sum_{x \in \{0,1\}^n} (-1)^{\langle x,1^n \rangle} (-1)^{\sum_{j \in T} \langle x,r_j \rangle} = \sum_{x \in \{0,1\}^n} (-1)^{\langle x,1^n \rangle} (-1)^{\langle x,\bigoplus_{j \in T} r_j \rangle} = \sum_{x \in \{0,1\}^n} (-1)^{\langle x,1^n \oplus (\bigoplus_{j \in T} r_j) \rangle}$$

This expression is not zero if and only if $\bigoplus_{j \in T} r_j = 1^n$. By Claim 5 the probability that such $T$ exists is at most $\frac{1}{3}$. So the probability over the choice of $r$ for some polynomial $p' : \{-1,1\}^m \rightarrow \mathbb{R}$ of degree at most $d = \frac{n}{4 \log m} - 1$ to have

$$\sum_{y' \in D'_r} \text{PUR}_r(y')p'(y') \neq 0$$

is at most $\frac{1}{3}$. $\square$

Combining expressions (1) and (2) and Claim 8 we have that with probability at least $\frac{2}{3}$,

$$\varepsilon > \frac{1}{2^n} \left| \sum_{y' \in D'_r} \text{PUR}_r(y') (\text{PUR}_r(y') - p'(y')) \right| = \frac{1}{2^n} \left| D'_r \right| - \left( \sum_{y' \in D'_r} \text{PUR}_r(y')p'(y') \right) = \frac{|D'_r|}{2^n} = 1.$$

And so $\varepsilon > 1$ which contradicts our assumption. Thus $\text{PUR}_r(Y(r,x))$ cannot be approximated by a polynomial of degree at most $\frac{n}{4 \log m} - 1$ with probability at least $\frac{2}{3}$ over the choice $r \leftarrow \mathcal{R}$ sampled uniformly at random. And therefore $\text{PUR}_r(Y(r,x))$ cannot be $\varepsilon$-approximated for every constant $\varepsilon < \frac{1}{2}$ with a polynomial of degree less than $\frac{n}{4 \log m}$ with probability at least $\frac{2}{3}$ over the choice $r \leftarrow \mathcal{R}$ sampled uniformly at random for any good base $\mathcal{R}$ of size $m$. $\square$

3.5 Lower bound for ordered search

Finally, we combine our general lower bound on the approximate degree of PUR with the upper bound on approximating GT to conclude our lower bound on the approximate degree of ordered search. We will use the statement of Claim 5 with a lower degree of polynomials approximating GT since, even though its proof is more complicated than the proof of the weaker bound, as it allows us to obtain a better lower bound on ordered search.

First, we apply Theorem 7 to obtain a lower bound on the approximate degree for $\text{PUR}[\mathcal{R}_{\text{OS}++}]$.

Corollary 9. If $r \leftarrow \mathcal{R}_{\text{OS}++}$ u.a.r. then with probability at least $\frac{2}{3}$ over the choice of $r$ every polynomial $p : \{0,1\}^m \rightarrow \mathbb{R}$ of degree at most $d = \frac{n}{4 \log n} - 1$ fails to approximate $\text{PUR}[\mathcal{R}_{\text{OS}++}]$.

$$\Pr_{r \leftarrow \mathcal{R}_{\text{OS}++}} \left[ \forall \varepsilon < \frac{1}{2}, \forall p, \deg(p) \leq d, \exists y \in D[\mathcal{R}_{\text{OS}++}]_r : |p(y) - \text{PUR}[\mathcal{R}_{\text{OS}++}]_r(y)| > \varepsilon \right] \geq \frac{2}{3}.$$
\textbf{Proof.} The set $\mathcal{R}_{\text{OS}^{++}}$ is a good base and has size $m = O(n^3 \log n)$. By Theorem \[\text{7}\] with probability at least $\frac{2}{3}$ over the choice of \( r \) every polynomial \( p : \{0,1\}^m \rightarrow \mathbb{R} \) of degree at most $\frac{n}{16 \log n} - 1$ fails to approximate \( \text{PUR}[\mathcal{R}_{\text{OS}^{++}}]_r \). But since the size of $\mathcal{R}_{\text{OS}^{++}}$ is $m \leq n^3$ for sufficiently large $n$ then every polynomial of degree at most $d = \frac{n}{16 \log n} - 1 = \frac{n}{4 \log n^4} - 1 \leq \frac{n}{4 \log m} - 1$ fails to approximate \( \text{PUR}[\mathcal{R}_{\text{OS}^{++}}]_r \). \hfill \Box

By combining Claim \[\text{5}\] and Corollary \[\text{9}\] we obtain the following.

\textbf{Theorem 10.} The approximate degree of ordered search is

\[ \overline{\deg}(\text{OS}_{2^n}) = \Omega \left( \frac{n}{\log^2 n} \right). \]

\textbf{Proof.} Suppose $\text{OS}_{2^n}$ can be \( \varepsilon \)-approximated by a bounded polynomial of degree \( d \) for some \( \varepsilon > 0 \).

By [She12a, Theorem 1.1], for every \( \delta > 0 \), this polynomial can be converted to a polynomial \( p \) of degree \( O(d + \log \frac{1}{\varepsilon}) \) that \((\varepsilon + \delta)\)-approximates $\text{OS}_{2^n}$ and is robust to noise in its inputs. That is:

\[ |\text{OS}_N(y) - p(y + \Delta)| < \varepsilon + \delta \]

for all \( y \in \{0,1\}^N \) where \( \Delta \in [-\frac{1}{\varepsilon}, \frac{1}{\varepsilon}]^N \) and \( N = 2^n \). If we put \( \delta = \varepsilon = \frac{1}{n} \), then \( p \) is a $\frac{1}{4}$-approximating polynomial for $\text{OS}_{2^n}$ with degree $O(d)$. Thus we can assume without loss of generality that the polynomial is robust to noise.

Note that $\text{OS}_{2^n}(\text{GT}_{0^0}(x), \text{GT}_{0^{0-1}}(x), \ldots, \text{GT}_{1^n}(x)) = \text{PUR}[\mathcal{R}_{\text{OS}^{++}}]_r(\mathcal{Y}(r,x))$ for every \( r \in \mathcal{R}_{\text{OS}^{++}} \). So by Claim \[\text{5}\] there exists a constant \( c \) such that the composed polynomial \( p(q_{(r,0^0)}(\mathcal{Y}(r,x)), q_{(r,0^{0-1})}(\mathcal{Y}(r,x)), \ldots q_{(r,1^n)}(\mathcal{Y}(r,x))) \) has degree at most $\deg(p) \max_i(\deg(q_{(r,i)}) = cd \log n$ and approximates $\text{PUR}[\mathcal{R}_{\text{OS}^{++}}]_r(\mathcal{Y}(r,x))$ to error $\frac{1}{3}$ with probability at least $\frac{2}{3}$ over the choice of \( r \leftarrow \mathcal{R}_{\text{OS}^{++}} \). This holds because although the polynomials \( q_{(r,i)} \) do not compute the functions \( \text{GT}_i \) exactly, but only approximate them with small error, the outer polynomial \( p \) is robust to this small error in the inputs. Note also that while the composed polynomial is bounded on the domain of \( \text{PUR}_r \), it may be arbitrarily unbounded on points outside its domain.

On the other hand, by Claim \[\text{9}\] with probability at least $\frac{2}{3}$ over the choice of \( r \), the function \( \text{PUR}[\mathcal{R}_{\text{OS}^{++}}]_r \) cannot be approximated by a polynomial in \( \mathcal{Y} \) of degree less than $\frac{n}{16 \log n}$. By a union bound, with probability at least $1 - \frac{2}{3} - \frac{2}{3} = \frac{1}{3}$ both conditions on \( r \) hold simultaneously. Thus there exists \( r \in \mathcal{R}_{\text{OS}^{++}} \) such that \( p(q_{(r,0^0)}(\mathcal{Y}(r,x)), q_{(r,0^{0-1})}(\mathcal{Y}(r,x)), \ldots q_{(r,1^n)}(\mathcal{Y}(r,x))) \) approximates $\text{PUR}_r(\mathcal{Y}(r,x))$ and $\text{PUR}_r(\mathcal{Y}(r,x))$ cannot be approximated by a polynomial of degree less than $\frac{n}{16 \log n}$. So

\[ cd \log n \geq \frac{n}{16 \log n}. \]

And thus

\[ d = \Omega \left( \frac{n}{\log^2 n} \right). \]

\hfill \Box

Note that by standard error amplification for approximating polynomials (see, e.g., [BNRdW07, Lemma 1]), $\deg_e(\text{OS}_{2^n}) = \Omega(n/\log^2 n)$ for every constant $\frac{1}{2} > \varepsilon > 0$. 

20
4 Anchored hidden string and hidden string

Now let us switch gears and consider the (anchored) hidden string problem, in which the goal is to reconstruct a string given information about the presence of specific substrings. In the decisional anchored hidden string (AHS) problem, the information given as input consists of whether each string \( s \) is a substring of the hidden input \( x \) starting at position \( i \), for all valid \( i \) and \( s \). The goal is then to compute \( \text{parity}(x) \).

In order to prove a lower bound for AHS we will follow the same outline as for OS. That is, first we will introduce a convenient set \( \mathcal{R}_{\text{AHS}} \) of collections of \( n \)-bit strings and show that oracle \( \mathcal{V}[\mathcal{R}_{\text{AHS}}] \) providing the inner products of \( x \) with strings from the random sample from \( \mathcal{R}_{\text{AHS}} \) are useful for computing \( \phi_{i,s}(x) \) for all possible queries \((i, s)\) where \( i \in [n], s \in \{0, 1\}^{n-i+1} \). After that, we will show that it is hard to compute \( \text{PUR}[\mathcal{R}_{\text{AHS}}] \), the parity of \( x \) using the oracle \( \mathcal{V}[\mathcal{R}_{\text{AHS}}] \) with high probability. And finally, we will conclude that computing AHS is hard since composing an approximating polynomial for AHS with polynomials approximating \( \phi_{i,s}(x) \) would allow us to approximate the \( \text{PUR}[\mathcal{R}_{\text{AHS}}] \) function.

4.1 Preliminaries

We define several functions in order to formalize the problem.

Throughout this section, we use the following notation \( \{0, 1\}^{\leq n} \) to denote the set of all bit strings of size at most \( n \): \( \{0, 1\}^{\leq n} = \bigcup_{k=0}^{n} \{0, 1\}^k \).

Definition 11. For all \( s \in \{0, 1\}^{\leq n} \) define the function \( \chi_s : \{0, 1\}^n \to \{0, 1\} \) to be the indicator of whether the input string \( x \) has \( s \) as a substring: that is, there exists an integer \( i \) such that \( x_{i+k-1} = s_k \) for all \( 1 \leq k \leq |s| \) then \( \chi_s(x) = 1 \) and otherwise \( \chi_s(x) = 0 \).

Definition 12. Define the “hidden string” function \( \text{HS}_N : \{0, 1\}^N \to \{0, 1\} \) be the partial function that takes \( N = |\{0, 1\}^{\leq n}| = 2^{n+1} - 1 \) inputs, each corresponding to a substring \( s \in \{0, 1\}^{\leq n} \), and, given a collection of \( \chi_s(x) \) for some fixed \( x \in \{0, 1\}^n \) as an input, outputs \( \text{parity}(x) \).

We will also consider a variation of this problem where the additional information is not only whether a specific substring is present in the hidden string, but if this specific substring is present at a specific location of the hidden string.

Definition 13. For all \( i \in [n] \) and \( s \in \{0, 1\}^{\leq n-i+1} \) define the function \( \phi_{i,s} : \{0, 1\}^n \to \{0, 1\} \) to be the indicator of whether the input string \( x \) has \( s \) as a substring starting from position \( i \): that is, if \( x_{i+k-1} = s_k \) for all \( 1 \leq k \leq |s| \) then \( \phi_{i,s}(x) = 1 \) and otherwise \( \phi_{i,s}(x) = 0 \).

Definition 14. Let the “anchored hidden string” function \( \text{AHS}_N : \{0, 1\}^N \to \{0, 1\} \) be the partial function that takes \( N = |\{(i, s) \mid i \in [n], s \in \{0, 1\}^{\leq n-i+1}\}| = 2^{n+2} - n - 4 \) inputs, each corresponding to a pair of \( i \in [n] \) and \( s \in \{0, 1\}^{\leq n-i+1} \), and, given a collection of \( \phi_{i,s}(x) \) for some fixed \( x \in \{0, 1\}^n \) as an input, outputs \( \text{parity}(x) \).

4.2 Approximating polynomials for \( \phi_{i,s} \)

We start our proof by showing that for some good base \( \mathcal{R}_{\text{AHS}} \) the oracle \( \mathcal{V}[\mathcal{R}_{\text{AHS}}] \) could be used to make the computation of the functions \( \phi_{i,s} \) more efficient.
Claim 15. There exists a good base $\mathcal{R}_{\text{AHS}}$ of size $m = O(n^3)$ such that if $r \leftarrow \mathcal{R}_{\text{AHS}}$ is sampled uniformly at random, then with probability at least $\frac{2}{3}$ over the choice of $r$ there exists a family of polynomials $\{q_{r,i,s}: \{0,1\}^m \rightarrow \{0,1\} \mid i \in [n], s \in \{0,1\}^{\leq n-i+1}\}$ of such that given $Y[\mathcal{R}_{\text{AHS}}](r, x)$ as the input, each polynomial $q_{r,i,s}(Y[\mathcal{R}_{\text{AHS}}](r, x))$ approximates the corresponding $\phi_{i,s}(x)$ with error at most $\frac{1}{6}$. That is,

$$\Pr_{r \leftarrow \mathcal{R}_{\text{AHS}}} \left[ \exists i \in [n], x \in \{0,1\}^n, s \in \{0,1\}^{\leq n-i+1} : |q_{r,i,s}(Y[\mathcal{R}_{\text{AHS}}](r, x)) - \phi_{i,s}(x)| > \frac{1}{6} \right] \leq \frac{1}{3}.$$  

Proof. The proof of this statement follows the same outline as the proof of Claim 4. First, we will construct a good base $\mathcal{R}_{\text{AHS}}$, and then we will show (in two stages) how to compute $\phi_{i,s}$ given $Y[\mathcal{R}_{\text{AHS}}]$.

**Constructing the good base $\mathcal{R}_{\text{AHS}}$.** Our base for the oracle should contain all the strings needed to check the equality with every substring of $x$, so let $\hat{\mathcal{R}} = \mathcal{R}^n \times (\mathcal{R}^{1n-1} \times \mathcal{R}^{01n-1}) \times (\mathcal{R}^{1n-2} \times \mathcal{R}^{01n-1} \times \mathcal{R}^{02n-2}) \times \ldots \times (\mathcal{X}_{i=0}^k \mathcal{R}^{01i+k0n-i-k}) \times \ldots \times (\mathcal{X}_{i=0}^{n-1} \mathcal{R}^{01i+10n-i(i+1)}).$ See Figure 4 for an illustration.

<table>
<thead>
<tr>
<th>$\tau$</th>
<th>$\mathcal{R}^\tau$</th>
<th>Structure of $\mathcal{R}^\tau$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>${0,1}^4$</td>
<td>$\star \star \star \star$</td>
</tr>
<tr>
<td>1110</td>
<td>${0,1}{0}$</td>
<td>$\star \star 0 \ 0$</td>
</tr>
<tr>
<td>0111</td>
<td>${0}{0,1}^3$</td>
<td>$0 \ 0 \star \star$</td>
</tr>
<tr>
<td>1100</td>
<td>${0,1}{0}^2$</td>
<td>$\star \star 0 \ 0$</td>
</tr>
<tr>
<td>0110</td>
<td>${0}{0,1}{0}$</td>
<td>$0 \ 0 \ 0 \ 0$</td>
</tr>
<tr>
<td>0011</td>
<td>${0}^2{0,1}$</td>
<td>$0 \ 0 \ 0 \ 0$</td>
</tr>
<tr>
<td>1000</td>
<td>${0,1}{0}^3$</td>
<td>$\star \ 0 \ 0 \ 0$</td>
</tr>
<tr>
<td>0100</td>
<td>${0}{0,1}{0}^2$</td>
<td>$\star \ 0 \ 0 \ 0$</td>
</tr>
<tr>
<td>0010</td>
<td>${0}^2{0,1}{0}$</td>
<td>$0 \ 0 \ 0 \ 0$</td>
</tr>
<tr>
<td>0001</td>
<td>${0}^3{0,1}$</td>
<td>$0 \ 0 \ 0 \ 0$</td>
</tr>
</tbody>
</table>

Figure 4: Structure of $\hat{\mathcal{R}}$ for $n = 4$. Blue cells with $\star$ represent places where either 0 or 1 values could be.

On top of this, we are going to add two other steps to the structure. First, we are going to have $t$ individual “prepackaged” copies. Let $\mathcal{R}_1 = \ldots = \mathcal{R}_t = \times_{\alpha} \hat{\mathcal{R}}$, and $\mathcal{R}' = \mathcal{X}_{\in [t]} \mathcal{R}_j$. Secondly, we add a set of “basis” strings to the structure: $\mathcal{B} = \{1_1\} \times \{1_2\} \times \ldots \times \{1_{n-1}\} \times \{1_n\} = \{10\ldots0\} \times \{010\ldots0\} \times \ldots \times \{00\ldots010\} \times \{00\ldots01\}$. The final underlying structure of the oracle will be a Cartesian product of all $t$ copies and $\mathcal{B}$: $\mathcal{A} = \mathcal{B} \times \mathcal{R}' = \mathcal{B} \times (\mathcal{X}_{\in [t]} \mathcal{R}_j)$. See Figure 5 for an illustration. We also set the parameters to be $\alpha = 4, t = 1000n \ln 2$. Notice that this set $\mathcal{R}_{\text{AHS}}$ is a good base by construction, and has size $m = n + \alpha t \frac{n(n+1)}{2} = O(n^3)$. 

22
Constructing the family of approximating polynomials. For all \( i \in [n], s \in \{0, 1\}^{n-i+1}, j \in [t], r \in \mathcal{R}_{\text{AHS}} \) let \( B_{(r,i,s,j)}(Y[\mathcal{R}_{\text{AHS}}](r, x)) \) be the following deterministic algorithm.

1. Set \( \tau = 0^i1^{|s|}0^{n-|s|-i} \)
2. Set \( s' = 0^t0^0\ldots0^{n-|s|-i} \) so \( s' \) is an \( n \)-bit string
3. For all \( v \in [m] \) such that \( v \) corresponds to \( n \)-bit strings drawn from \( \mathcal{R}^{\tau} \) within the \( j \)th copy \( \mathcal{R}_j \):
4. Compute \( \langle s', r_v \rangle \) and compare it to \( (Y(r, x))_v = \langle x, r_v \rangle \).
5. If for some \( v \) the inner products don’t have the same value, \( \langle i, r_v \rangle \neq (Y(r, x))_v \), then reject.
6. Otherwise, accept.

This algorithm determines if \( s \) equal to the substring of \( x \) of length \( |s| \) that starts at the \( i \)th position with probability at least \( 1 - 2^{-\alpha} = 1 - 2^{-4} \geq \frac{11}{12} \) independently of the choice of \( j \in [t] \).

That is, for all \( j \in [t] \) and for all \( i \in [n], s \in \{0, 1\}^{n-i}, x \in \{0, 1\}^n \)

\[
\Pr_{r \leftarrow \mathcal{R}_{\text{AHS}}}[B_{(r,i,s,j)}(Y(r, x)) = \phi_{i,s}(x)] \geq \frac{11}{12}
\]

This algorithm makes at most \( \alpha = 4 \) queries to the oracle \( Y(r, x) \).

We have shown that for every fixed \( i \in [n], s \in \{0, 1\}^{n-i+1}, \) and \( x \in \{0, 1\}^n \) there are many \( r \in \mathcal{R}_{\text{AHS}} \) that if used as the first input for the oracle \( Y \) allow \( B_{(r,i,s,j)} \) to compute \( \phi_{i,s}(x) \).

Let \( W(i, s, x, r, j) \) be the indicator that the \( j \)th “package” of random strings in \( r \) defines a set of “bad” random strings for \( i, s, x \): \( W(i, s, x, r, j) = 1 \) if and only if \( B_{(r,i,s,j)}(Y(r, x)) \neq \phi_{i,s}(x) \). We established that \( B_{(r,i,s,j)}(Y(r, x)) \) works well if given a random \( r \leftarrow \mathcal{R}_{\text{AHS}} \) for every \( j \in [t] \) and the probability of this algorithm outputting an incorrect answer is at most \( \frac{1}{12} \).

So

\[
\Pr_{r \leftarrow \mathcal{R}_{\text{AHS}}} [W(i, s, x, r, j) = 1] = \mathbb{E}_{r \leftarrow \mathcal{R}_{\text{AHS}}} [W(i, s, x, r, j)] \leq \frac{1}{12}.
\]
Using the same transformation from before (described in detail in the proof of Claim 4), we design a new family of algorithms that succeeds on all $i, s, x$ simultaneously. For all $i \in \{0, 1\}^{\leq n-i+1}, r \in \mathcal{R}_{OS^+}$, let $A_{(r,i,s)}(\mathcal{Y}(r, x))$ be the following randomized algorithm:

- Choose $j \leftarrow [t]$ uniformly at random.
- Run $B_{(r,i,s,j)}(\mathcal{Y}(r, x))$.

The number of queries that $A_{(r,i,s)}$ makes to the oracle is the same as $B_{(r,i,s,j)}$ which is $\alpha = 4$. We fix arbitrary $i, s, x$ and evaluate the following probability.

$$\Pr_{r \leftarrow \mathcal{R}_{AHS}} \left[ \Pr_{j \leftarrow [t]} [B_{(r,i,s,j)} \neq \phi_{i,s}(x)] > \frac{1}{6} \right] = \Pr_{r \leftarrow \mathcal{R}_{AHS}} \left[ \frac{1}{t} \sum_{j \in [t]} W(i, s, x, r, j) > \frac{1}{6} \right].$$

We established that $\mathbb{E}_{r \leftarrow \mathcal{R}_{AHS}}[W(i, s, x, r, j)] \leq \frac{1}{12}$ and so by Hoeffding’s inequality,

$$\Pr_{r \leftarrow \mathcal{R}_{AHS}} \left[ \frac{1}{t} \sum_{j \in [t]} W(i, s, x, r, j) > \frac{1}{12} + \frac{1}{12} \right] \leq e^{-2\frac{1}{144}} \leq 2^{-\frac{2000n}{144}}.$$

And by a union bound,

$$\Pr_{r \leftarrow \mathcal{R}_{AHS}} \left[ \exists i \in [n], s \in \{0, 1\}^{\leq n-i+1}, x \in \{0, 1\}^n : \frac{1}{t} \sum_{j \in [t]} W(i, s, x, r, j) > \frac{1}{6} \right] \leq 2^{2n+2}2^{-\frac{2000n}{144}} < \frac{1}{3},$$

since the number of possible pairs of $i \in [n]$ and $s \in \{0, 1\}^{\leq n-i+1}$ is $\sum_{i=1}^{n} \sum_{|s|=1}^{n-i+1} 2|s| \leq 2^{n+2}$. So, we proved that

$$\Pr_{r \leftarrow \mathcal{R}_{AHS}} \left[ \exists i \in [n], s \in \{0, 1\}^{\leq n-i+1}, x \in \{0, 1\}^n : \Pr_{j \leftarrow [t]} [A_{(r,i,s)}(\mathcal{Y}(r, x))] \neq \phi_{i,s}(x)] > \frac{1}{6} \right] < \frac{1}{3}.$$

The last step is to convert this family of query algorithms into a family of approximating polynomials. Let $q_{(r,i,s)}$ denote the acceptance probability of $A_{(r,i,s)}$ which is a polynomial of degree at most $\alpha = 4$ such that

$$\Pr_{r \leftarrow \mathcal{R}_{AHS}} \left[ \exists i \in [n], s \in \{0, 1\}^{\leq n-i+1}, x \in \{0, 1\}^n : |q_{(r,i,s)}(\mathcal{Y}(r, x)) - \phi_{i,s}(x)| > \frac{1}{6} \right] < \frac{1}{3},$$

which is exactly what we were looking for. 

\[ \square \]
4.3 Lower bounds for anchored hidden string and hidden string

Following the same framework, we can combine the general statement about the degree of PUR proven earlier with the upper bound for approximating $\phi_{i,s}$ to conclude our lower bound on the approximate degree of anchored hidden string.

First, we apply Theorem 7 to obtain a lower bound on approximate degree for $\text{PUR}[\mathcal{R}_{\text{AHS}}]$.

**Corollary 16.** If $r \leftarrow \mathcal{R}_{\text{AHS}}$ u.a.r. then with probability at least $\frac{2}{3}$ over the choice of $r$ every polynomial $p : \{0,1\}^m \to \mathbb{R}$ of degree at most $d = \frac{n}{\log n} - 1$ fails to approximate $\text{PUR}[\mathcal{R}_{\text{AHS}}]_r$:

$$\Pr_{r \leftarrow \mathcal{R}_{\text{AHS}}} \left[ \forall \varepsilon < \frac{1}{2}, \forall p, \deg(p) \leq d, \exists y \in D[\mathcal{R}_{\text{AHS}}]_r : |p(y) - \text{PUR}[\mathcal{R}_{\text{AHS}}]_r(y)| > \varepsilon \right] \geq \frac{2}{3}.$$

**Proof.** The set $\mathcal{R}_{\text{AHS}}$ is a good base and has size $m = O(n^3)$. By Theorem 7 with probability at least $\frac{2}{3}$ over the choice of $r$ every polynomial $p : \{0,1\}^m \to \mathbb{R}$ of degree at most $\frac{n}{\log n} - 1$ fails to approximate $\text{PUR}[\mathcal{R}_{\text{AHS}}]_r$. But since the size of $\mathcal{R}_{\text{AHS}}$ is $m \leq n^4$ for sufficiently large $n$ then every polynomial of degree at most $d = \frac{n}{\log n} - 1 = \frac{n}{4\log n} - 1 \leq \frac{n}{4\log n} - 1$ fails to approximate $\text{PUR}[\mathcal{R}_{\text{AHS}}]_r$.

By combining Claim 15 and Corollary 16 we obtain the following.

**Theorem 17.** The approximate degree of $\text{AHS}_N$ is

$$\widetilde{\deg}(\text{AHS}_N) = \Omega \left( \frac{n}{\log n} \right)$$

where $N = 2^{n+2} - n - 4$.

**Proof.** Suppose $\text{AHS}_N$ can be $\frac{1}{4}$-approximated by a bounded polynomial of degree $d$.

By the same argument as used in Theorem 10 we can assume without loss of generality that the polynomial is robust to noise. That is,

$$|\text{AHS}_N(y) - p(y + \Delta)| < \frac{1}{3}$$

for all $y \in \{0,1\}^N$ where $\Delta \in [-\frac{1}{6}, \frac{1}{6}]^N$.

Note that $\text{AHS}_N((\phi_{i,s}(x)))_{i \in [n], s \in \{0,1\}^{\leq n-1}} = \text{PUR}[\mathcal{R}_{\text{AHS}}]_{\mathcal{Y}}(\mathcal{Y}(r, x))$ for every $r \in \mathcal{R}_{\text{AHS}}$. So by Claim 15 the polynomial $p(q_{(r,i,s)}(\mathcal{Y}(r, x)))$ of degree at most $\deg(p) \max_{i,s}(\deg(q_{(r,i,s)})) = 4d$ approximates $\text{PUR}[\mathcal{R}_{\text{AHS}}]_{\mathcal{Y}}(\mathcal{Y}(r, x))$ to error $\frac{1}{3}$ with probability at least $\frac{2}{3}$ over the choice of $r \leftarrow \mathcal{R}_{\text{AHS}}$. This holds because although the polynomials $q_{(r,i,s)}$ do not compute the functions $\phi_{i,s}$ exactly, but only approximate them with small error, the outer polynomial $p$ is robust to this small error in the inputs. Note also that while the composed polynomial is bounded on the domain of $\text{PUR}_r$, it may be arbitrarily unbounded on points outside its domain.

On the other hand, by Claim 16 with probability at least $\frac{2}{3}$ over the choice of $r$, the function $\text{PUR}[\mathcal{R}_{\text{AHS}}]_{\mathcal{Y}}$ cannot be approximated by a polynomial in $\mathcal{Y}$ of degree less than $\frac{n}{16\log n}$. By a union bound, with probability at least $(1 - (1 - \frac{2}{3}) - (1 - \frac{2}{3})) = \frac{1}{3}$ both conditions on $r$ hold simultaneously. Thus there exists $r \in \mathcal{R}_{\text{AHS}}$ such that $p(q_{(r,i,s)}(\mathcal{Y}(r, x)))$ approximates $\text{PUR}_r(\mathcal{Y}(r, x))$ and $\text{PUR}_r(\mathcal{Y}(r, x))$ cannot be approximated by a polynomial of degree less than $\frac{n}{16\log n}$. So
4d \geq \frac{n}{16 \log n}.

And thus

\[ d = \Omega \left( \frac{n}{\log n} \right). \]

This lower bound on the approximate degree of $AHS_N$ entails a lower bound on the approximate degree of $HS_N$. In [CIG+12] the authors gave a reduction between these two problems, showing that if there exists a quantum query algorithm for $HS$ then there exists a quantum query algorithm for $AHS$ with a small blow-up in the number of queries. Specifically, they showed how to compute $AHS$ by applying a query algorithm for $HS$ with a slightly bigger input, where each bit of the bigger input can be computed using a constant number of queries to the original $AHS$ input. This argument works just as well to relate the approximate degrees of $HS$ and $AHS$, giving the following statement.

**Claim 18.** If for every $n'$, there is a polynomial of degree $d(n')$ approximating $HS_{2^{n'+1}-1}$ to some constant error, then for every $n$ there is a polynomial of degree $2d(10n \log n)$ approximating $AHS_{2^{n+2}-n-4}$ to the same error.

This allows us to prove a lower bound for $HS_N$ as well.

**Corollary 19.** The approximate degree of $HS_N$ is

\[ \widetilde{\deg}(HS_N) = \Omega \left( \frac{n}{\log^2 n} \right), \]

where $N = 2^{n+1} - 1$.

**Proof.** By Claim 18 if there exists a polynomial of degree $d(n')$ approximating $HS_{2^{n'+1}-1}$ then there exists a polynomial of degree $2d(10n \log n)$ approximating $AHS_{2^{n+2}-n-2}$. On the other hand, by Corollary 16 no polynomial of degree less than $\frac{cn}{\log n}$ can approximate $AHS_{2^{n+2}-n-2}$ for some constant $c$. Therefore,

\[ 2d(10n \log n) \geq \frac{cn}{\log n}. \]

Set $n' = 10n \log n$. Then

\[ d(n') \geq \frac{cn}{2 \log n} = \frac{cn'}{20 \log^2 n} \geq \frac{cn'}{20 \log^2 n'}. \]

And thus

\[ \widetilde{\deg}(HS_{2^{n'+1}-1}) = \Omega \left( \frac{n'}{\log^2 n'} \right). \]

Note that by standard error amplification for approximating polynomials, $\widetilde{\deg}_\varepsilon(HS_{2^{n+1}-1}) = \Omega(n/\log^2 n)$ and $\widetilde{\deg}_\varepsilon(AHS_{2^{n+2}-n-4}) = \Omega(n/\log n)$ for every constant $\frac{1}{2} > \varepsilon > 0$. 

26
Acknowledgments. We thank Arkadev Chattopadhyay for suggesting the problem of determining the approximate degree of ordered search, and Arkadev and Justin Thaler for many helpful conversations about it.

References


A Proof of Claim 5

The communication protocol that we use to construct our polynomials itself is based on a result by [FRPU94] on algorithms in a noisy comparison model. To understand the protocol and to convert it to the family of polynomials later we need to open up the protocol and state their result.

The following is implicit in [FRPU94]:

**Claim 20.** Consider the following problem. There is an unknown “key” in $[n]$. Given the ability to ask questions of the type “is the unknown key greater than $a$?” for every $a \in [n]$ and get the correct answer with probability at least $\frac{3}{4}$ independently for each question, the algorithm’s goal is to find the correct location in $(0, n]$ for the key while minimizing the number of questions it asks.

Then there exists an algorithm that finds the correct location in $(0, n]$ for the key by asking at most $c \log n$ questions for some constant $c$ with probability at least $\frac{11}{12}$.

The algorithm basically performs a binary search for the correct location of the key with slight modifications. We are not going to present the algorithm in detail, but we are going to describe what questions the algorithm asks along the way. For each interval $(a, b]$ where $b - a > 1$, the algorithm seeks answers to the following questions: “is the position of the most significant bit where the inputs differ greater than $a$?”, “is the key greater than $a$?”, and “is the key greater than $\frac{a+b}{2}$?”, each correct with probability $\frac{3}{4}$ independently from all other questions, and the algorithm needs to be able to ask each potential question of each type $c \log n$ times for some constant $c$. For each interval $(a, a + 1]$ the algorithm seeks answers to the following questions: “is the key greater than $a$?” and “is the key greater than $a + 1$?”, each correct with probability at least $\frac{3}{4}$ independently from all other questions, and the algorithm needs to be able to ask each potential question of each type $2c^2 \log^2 n$ times. Note that if the range for the key position is $(0, n]$ then questions “is the key $> 0$?” and “is the key $> n$?” can be omitted by the algorithm since it already knows the correct answer to them.

The efficient communication protocol for the GT heavily relies on the algorithm above. In the communication protocol, both Alice and Bob run the algorithm to find the most significant bit where their inputs differ. Each time the algorithm asks “is the position of the most significant bit where the inputs differ greater than $a$?” (or “is the key greater than $a$?”), Alice and Bob compute
the equality of the first $a$ bits of their inputs to error $\frac{1}{4}$ by computing $\alpha = 2$ inner products of their inputs with random strings from $\mathcal{R}^\tau$ where $\tau = 1^a 0^{n-a}$. See Figure 6 for an illustration. If their inner products are the same then the answer to the algorithm is “no, the key is not greater than $a$” and otherwise, it’s “yes, the key is greater than $a$”. And at the end of the protocol, Alice and Bob check who has a greater value in the most significant bit discovered during the procedure. Now we are ready to prove the claim.  

**Constructing the good base $\mathcal{R}_{OS^{++}}$.** The construction is similar to the construction of $\mathcal{R}_{OS}$. Let $\hat{\mathcal{R}}$ have the following structure.

<table>
<thead>
<tr>
<th>Interval</th>
<th>Questions</th>
<th>$\tau$</th>
<th>Structure of $\mathcal{R}^\tau$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(0, n]$</td>
<td>“is the key $\geq \frac{n}{2}$?”</td>
<td>$1^n 2^n 0^n/2$</td>
<td>$\bullet \bullet \bullet \bullet \bullet \bullet \bullet 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$</td>
</tr>
<tr>
<td>$(0, \frac{n}{2}]$</td>
<td>“is the key $\geq \frac{n}{2}$?”</td>
<td>$1^n 2^n 0^n/2$</td>
<td>$\bullet \bullet \bullet \bullet \bullet \bullet \bullet 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$</td>
</tr>
<tr>
<td>$(\frac{n}{2}, n]$</td>
<td>“is the key $\geq \frac{n}{2}$?”</td>
<td>$1^n 2^n 0^n/2$</td>
<td>$\bullet \bullet \bullet \bullet \bullet \bullet \bullet 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$</td>
</tr>
<tr>
<td>$(\frac{n}{4}, \frac{n}{2}]$</td>
<td>“is the key $\geq \frac{n}{4}$?”</td>
<td>$1^n 2^n 0^n/2$</td>
<td>$\bullet \bullet \bullet \bullet \bullet \bullet \bullet 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$</td>
</tr>
<tr>
<td>$(\frac{n}{2}, \frac{3n}{4}]$</td>
<td>“is the key $\geq \frac{n}{4}$?”</td>
<td>$1^n 2^n 0^n/2$</td>
<td>$\bullet \bullet \bullet \bullet \bullet \bullet \bullet 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$</td>
</tr>
<tr>
<td>$(\frac{3n}{4}, n]$</td>
<td>“is the key $\geq \frac{3n}{4}$?”</td>
<td>$1^n 2^n 0^n/2$</td>
<td>$\bullet \bullet \bullet \bullet \bullet \bullet \bullet 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$</td>
</tr>
</tbody>
</table>

Figure 6: The structure of random strings used in the communication protocol.
\[ \hat{R} = R^{1^n/2} \times (R^{1^n/2} \times R^{1^n/4}) \times \ldots \times \left( R^{1^n/2} \times R^{1^n/4} \times R^{1^n/8} \times R^{1^n/16} \right) \times \left( R^{1^n/2} \times R^{1^n/4} \times R^{1^n/8} \times R^{1^n/16} \right) \times \left( R^{1^n/2} \times R^{1^n/4} \times R^{1^n/8} \times R^{1^n/16} \right) \]

See Figure 7 for an illustration. This \( \hat{R} \) describes all the strings as the source of randomness needed for the \( O(\log n) \) communication protocol for GT, but each of the strings appears in the structure only once instead of \( \alpha c \log n \) times. So, we need to duplicate this structure \( \alpha c \log n \) times to properly simulate the protocol.

As in the proof of Claim 4, we are going to add two other steps to the structure. First, we are going to have \( t \) individual “prepackaged” copies for the GT protocol. Let \( \mathcal{R}_1 = \ldots = \mathcal{R}_t = \times \alpha c \log n \hat{R} \). Each of the copies has enough randomness and the right structure of that randomness to simulate one full run of the GT protocol. Let \( \mathcal{R}' = \times_{j \in [l]} \mathcal{R}_j \), which allows us to handle \( t \) runs. Secondly, we add a set of “basis” strings to the structure: \( \mathcal{B} = \{1\} \times \{1\} \times \ldots \{1_{n-1}\} \times \{1\} = \{10 \ldots 0\} \times \{010 \ldots 0\} \times \ldots \times \{00 \ldots 010\} \times \{00 \ldots 01\} \times \{00 \ldots 01\} \).

The final underlying structure of the oracle will be a Cartesian product of \( \mathcal{R}' \) and \( \mathcal{B} \): \( \mathcal{R}_{OS++} = \mathcal{B} \times \mathcal{R}' = \mathcal{B} \times (\times_{j \in [l]} \mathcal{R}_j) \). See Figure 8 for an illustration.

We also set the parameters to be \( \alpha = 2, t = 250 n \ln 2 \). Notice that this set \( \mathcal{R}_{OS++} \) is a good base by construction and has size \( m \leq n + 3 \alpha t (cn \log n + 2cn \log^2 n) = O(n^3 \log^2 n) \).

**Constructing the family of approximating polynomials.** The construction will follow the same outline as the construction of polynomials for \( \mathcal{R}_{OS++} \). We construct a family of deterministic algorithms that work well for every fixed pair of inputs \( i, x \in \{0, 1\}^n \), and then construct a family of algorithms that work for all inputs with high probability at the same time, and then finally explain how to convert it to a family of approximating polynomials.

For all \( i \in \{0, 1\}^n, j \in [t], r \in \mathcal{R}_{OS++} \) let \( B_{(r,i,j)}(\mathcal{Y}(r,x)) \) be the following deterministic algorithm.

1. Simulate the algorithm from Claim 20.
2. Each time the algorithm asks “is the key > \( a \)?”:
3. \( \tau = 1^a 0^{n-a} \).
4. For the \( \alpha \) indices \( v \in [m] \) corresponding to \( n \)-bit strings drawn from \( \mathcal{R}' \) within the \( j \)th copy \( \mathcal{R}_j \) that were not already used prior to this step:
5. Compute \( \langle i, r_v \rangle \) and compare it to \( (\mathcal{Y}(r,x))_v = \langle x, r_v \rangle \).
6. If for all \( \alpha \) indices \( v \) the inner products are the same then reply “yes, the key > \( a \)”. Otherwise, reply “no, the key \( \leq a \)”. 

32
<table>
<thead>
<tr>
<th>Interval</th>
<th>$\tau$</th>
<th>Structure of $R^\tau$</th>
</tr>
</thead>
</table>
| $(0, 8]$ | 11110000 | \[ \begin{array}{cccccc}
\star & \star & \star & 0 & 0 & 0 \\
\star & \star & \star & 0 & 0 & 0 \\
\end{array} \] |
| $(0, 4]$ | 11110000 | \[ \begin{array}{cccccc}
\star & \star & \star & 0 & 0 & 0 \\
\star & 0 & 0 & 0 & 0 & 0 \\
\end{array} \] |
| $(4, 8]$ | 11110000 | \[ \begin{array}{cccccc}
\star & \star & \star & 0 & 0 & 0 \\
\star & \star & \star & 0 & 0 & 0 \\
\end{array} \] |
| $(0, 2]$ | 11000000 | \[ \begin{array}{cccccc}
\star & 0 & 0 & 0 & 0 & 0 \\
\star & 0 & 0 & 0 & 0 & 0 \\
\end{array} \] |
| $(2, 4]$ | 11110000 | \[ \begin{array}{cccccc}
\star & \star & \star & 0 & 0 & 0 \\
\star & \star & \star & 0 & 0 & 0 \\
\end{array} \] |
| $(4, 6]$ | 11110000 | \[ \begin{array}{cccccc}
\star & \star & \star & 0 & 0 & 0 \\
\star & \star & \star & 0 & 0 & 0 \\
\end{array} \] |
| $(6, 8]$ | 11110000 | \[ \begin{array}{cccccc}
\star & \star & \star & \star & 0 & 0 \\
\star & \star & \star & \star & 0 & 0 \\
\end{array} \] |
| $(0, 1]$ | 10000000 | \[ \begin{array}{cccccc}
\star & 0 & 0 & 0 & 0 & 0 \\
\star & 0 & 0 & 0 & 0 & 0 \\
\end{array} \] \[ \text{repeat 2c log } n \text{ times} \] |
| $(1, 2]$ | 10000000 | \[ \begin{array}{cccccc}
\star & 0 & 0 & 0 & 0 & 0 \\
\star & 0 & 0 & 0 & 0 & 0 \\
\end{array} \] \[ \text{repeat 2c log } n \text{ times} \] |
| $(7, 8]$ | 11110000 | \[ \begin{array}{cccccc}
\star & \star & 0 & 0 & 0 & 0 \\
\star & \star & 0 & 0 & 0 & 0 \\
\end{array} \] \[ \text{repeat 2c log } n \text{ times} \] |

Figure 7: Structure of $\hat{R}$. Blue cells with $\star$ represent indices where either a 0 or 1 could appear.
Figure 8: Structure of $\mathcal{R}_{\text{OS}^+}$. Each $\mathcal{R}_j$ consists of $c\alpha \log n$ copies of $\hat{\mathcal{R}}$.

7. Let $k$ be the output of the algorithm.

8. Compare $i_k = \langle i, 1_k \rangle$ and $(\mathcal{Y}(r, x))_k = \langle x, 1_k \rangle = x_k$. If $x_k \leq i_k$ then accept. Otherwise, reject.

Notice that this algorithm emulates the $O(\log n)$ randomized communication protocol for GT communication problem.

This algorithm computes $\text{GT}_i(x)$ with probability at least $\frac{11}{12}$ for all $j \in [t]$. That is, for all $i, x \in \{0, 1\}^n$

$$\Pr_{r \leftarrow \mathcal{R}_{\text{OS}^+}} \left[ \mathcal{B}_{(r,i,j)}(\mathcal{Y}(r, x)) = \text{GT}_i(x) \right] \geq \frac{11}{12}.$$ 

This algorithm makes at most $3\alpha c \log n = 6c \log n$ queries to the oracle $\mathcal{Y}(r, x)$. Let $W(i, x, r, j)$ be the indicator that the $j$-th package of random strings in $r$ defines a set of “bad” random strings for $(i, x)$: $W(i, x, r, j) = 1$ if and only if $B_{(r,i,j)}(\mathcal{Y}(r, x)) \neq \text{GT}_i(x)$. We established that $B_{(r,i,j)}(\mathcal{Y}(r, x))$ works well if given a random $r \leftarrow \mathcal{R}_{\text{OS}^+}$ for every $j \in [t]$ and the probability of this algorithm outputting an incorrect answer is at most $\frac{1}{12}$. So for all $i, x \in \{0, 1\}^n, j \in [t]$

$$\Pr_{r \leftarrow \mathcal{R}_{\text{OS}^+}} \left[ W(i, x, r, j) = 1 \right] = \mathbb{E}_{r \leftarrow \mathcal{R}_{\text{OS}^+}} [W(i, x, r, j)] \leq \frac{1}{12}.$$ 

For all $i \in \{0, 1\}^n, r \in \mathcal{R}_{\text{OS}^+}$ let $A_{(r,i)}(\mathcal{Y}(r, x))$ be the following randomized algorithm:

- Choose $j \leftarrow [t]$ uniformly at random.
- Run $B_{(r,i,j)}(\mathcal{Y}(r, x))$.

The number of queries that $A_{(r,i)}$ makes to the oracle is the same as $B_{(r,i,j)}$, which is $3\alpha c \log n = 6c \log n$. We fix a pair $(i, x)$ and evaluate the following probability.

$$\Pr_{r \leftarrow \mathcal{R}_{\text{OS}^+}} \left[ \Pr_{j \leftarrow [t]} \left[ B_{(r,i,j)} \neq \text{GT}_i(x) \right] > \frac{1}{6} \right] = \Pr_{r \leftarrow \mathcal{R}_{\text{OS}^+}} \left[ \frac{1}{t} \sum_{j \in [t]} W(i, x, r, j) > \frac{1}{6} \right].$$
We established that $\mathbb{E}_{r \leftarrow R_{OS++}} [W(i, x, r, j)] \leq \frac{1}{12}$ and so by Hoeffding's inequality,

$$
\Pr_{r \leftarrow R_{OS++}} \left[ \frac{1}{t} \sum_{j \in [t]} W(i, x, r, j) > \frac{1}{12} + \frac{1}{12} \right] \leq e^{-2 \frac{1}{144}} \leq 2^{-\frac{500n}{144}}.
$$

By a union bound over all possible $i, x \in \{0, 1\}^n$,

$$
\Pr_{r \leftarrow R_{OS++}} \left[ \exists i, x \in \{0, 1\}^n : \frac{1}{t} \sum_{j \in [t]} W(i, x, r, j) > \frac{1}{6} \right] \leq 2^{2n}2^{-\frac{500n}{144}} \leq 2^{-n} < \frac{1}{3}.
$$

Therefore, we have proven that

$$
\Pr_{r \leftarrow R_{OS++}} \left[ \exists i, x \in \{0, 1\}^n : \Pr_{j \leftarrow [t]} [A_{(r,i)}(Y(r, x))] \neq GT_i(x)] > \frac{1}{6} \right] < \frac{1}{3}.
$$

The last step is to convert this family of query algorithms into a family of approximating polynomials. Let $q_{(r,i)}$ denote the acceptance probability of $A_{(r,i)}$ which is a polynomial of degree at most $6c \log n$ such that

$$
\Pr_{r \leftarrow R_{OS++}} \left[ \exists i, x \in \{0, 1\}^n : |q_{(r,i)}(Y(r, x)) - GT_i(x)| > \frac{1}{6} \right] < \frac{1}{3}
$$

which is exactly what we were looking for.