# Stronger lower bounds for text searching and polynomial evaluation

Alexander Golynski
David R. Cheriton School of Computer Science
University of Waterloo
agolynsk@cs.uwaterloo.ca

**Abstract**

In this paper, we give two main technical results: (i) we show a stronger lower bound for substring search problem via compression extending results of Demaine and López-Ortiz (SODA '01); (ii) improve the results of Gal and Miltersen (ICALP '03) by showing a bound on the redundancy needed by the polynomial evaluation problem that is linear in terms of the information-theoretic minimum storage required by a polynomial.

## 1 Introduction

In this paper, we revisit recent work of Demaine and López-Ortiz [5], Gal and Miltersen [10], and Munro et al. [25]. We start with the text retrieval problem. This problem gained more importance in recent years. For example, Google index expanded by a factor of 1000 in seven years from 1998-2005 [27]. They are also rumored to increase their index up to 100 billion pages as compared to their index in 2005 that was of size approximately 8 billion pages. Consider a simple scenario where we have a text $T$ and a pattern $P$, we are to answer a query whether $P$ occurs in $T$, and, if yes, find a position of such an occurrence. There has been large body of work devoted to this problem, for example, the algorithm of Knuth, Morris, and Pratt [19] has complexity $O(|T| + |P|)$, which can be acceptable for small text files on a PC hard drive (e.g. grep utility), but not for large scale search engines like Google. This algorithm uses some precomputation on the given pattern $P$, but not on the text $T$. In real life applications, e.g. search engines, it is possible to perform some precomputation on the text while crawling the web. The common data structures that take advantage of the preprocessing stage are suffix trees [32, 22, 31, 12, 6, 17], suffix arrays [21, 18, 15, 16, 28, 29], FM-index [7, 8, 9], and wavelet trees [14]. Some recent developments include [13, 2] that can be used as building blocks in these data structures. Such data structures provide various time-space tradeoffs (so called upper bounds) between the size of the storage and time to perform text retrieval operations, such as searching for a pattern.

To facilitate the progress in this direction further, it is important to understand the intrinsic limitations (lower bounds) imposed on any type of data structure by the problem. The question can be informally stated as follows: is it possible to design a data structure that stores $s$ bits of information and performs a given set of queries in time $t$ ($s$ and $t$ can be functions of $|T|$, $|P|$, the alphabet size $|\Sigma|$, and some other parameters such as the $k$-th order entropy of the text)? Very little is known about lower bounds for text retrieval operations. The widely accepted model to study lower bounds is Yao's cell probe model [33]. However, no lower bounds for the text searching problem are known in this model unless we make an additional restriction that our data structure has to store the text in its "raw form" plus some auxiliary information $I$ (called the *index*) to facilitate efficient implementation of retrieval operations. We can informally describe the model

as follows: the text is stored in slow and inexpensive memory, and the index is stored in fast and expensive memory. The retrieval algorithm is allowed to access the text incurring the time cost one, while accessing the index is free of charge; we pay the space cost one for each bit of the fast memory used by the index, however the text is stored for us without any cost. We are not concerned with the time and space for intermediate computations. Thus, the computation can be viewed as a decision tree with the top node labeled "$I = ?$" and $2^r$ outgoing edges labeled by the possible contents of the index, where $r = |I|$ is the space cost. The other nodes of the tree are labeled "$S[i]$=?", where $S[i]$ is the $i$-th position of the text, and the outgoing edges are labeled with characters from $\Sigma$. The leaves are labeled with the outputs of the algorithm. The time cost $t$ of the algorithm is defined to be the depth of this tree. For example, such lower bounds were considered by Yao [34] for permutations and inverting functions.

Demaine and López-Ortiz [4, 5] considered the substring report problem where we are required to output the location of an occurrence of the pattern $P$, if any. Gal and Miltersen [10] considered the problem where we are to output only YES/NO. Both results are for the case of binary alphabets and are shown in the the indexing bit probe model where we can only query one bit of $T$ at a time. Let $L$ denote the length of the text.

**Theorem 1** (Demaine and López-Ortiz [5])**.** *For the substring report problem in the indexing bit probe model, if* $|P| = \lg n + o(\lg n)$*,* $t = o((\lg L)^2/\lg\lg L)$ *and* $t = \Omega(\lg L)$*, then* $(r+1)t = \Omega(L \lg L)$*.*

**Theorem 2** (Gal and Miltersen [10])**.** *For the substring search problem in the indexing bit probe model, if* $2\lg L + 5 \le |P| \le 5\lg L$ *then* $(r+1)t = \Omega(L/\lg L)$*.*

For small values of $t$, Theorem 1 is always better than Theorem 2, particularly, in the case $t = \Theta(\lg L)$, Theorem 1 gives a linear lower bound on the space cost. Clearly, in this model, the linear space cost is the best possible: if we can afford to store the whole text in the index, then no bit probes are needed to the slow memory. However, as $t$ reaches $\Omega((\lg L)^2/\lg\lg L)$, the techniques from [5] aren't good enough to yield any meaningful lower bound on the space cost, while Theorem 2 gives a non-trivial lower bound. An interesting question posed in [10] " Can the two techniques be combined to yield a better lower bound?". Is the limitation $t = o((\lg L)^2/\lg\lg L)$ essential for the substring search/report problem? We develop a new compression technique and answer this question affirmatively by showing

**Theorem 3.** *For the substring report problem in the indexing bit probe model, if* $|P| = \lg L + o(\lg L)$*,* $t = o(\sqrt{L}/\lg L)$*, and* $t \ge \lg L$*, then*

$$r \ge \frac{LD^2}{4t\lg L} - \Theta\left(\frac{LD}{t\lg L}\right)$$

*where* $D = \lg L - 2\lg\lg L - 2\lg t$*.*

This bound is stronger than [5]: it applies to a much wider range of parameters $t$; in the range $\le L^{1/2-\varepsilon}$ for arbitrarily small constant $\varepsilon > 0$, this bound is as good as Theorem 1; and also in the interesting case $t = c\lg L$, it gives a better constant for the index size in front of $\Theta(L)$.

Similar techniques were used in Cryptography by Gennaro and Trevsian [11, Lemma 1]. Although not explicitly claimed in [11], using their techniques it is possible to show a stronger lower bound for inverting permutations than in Munro et al. [25]. The permutations problem is to represent $\pi$ on $n$ elements such that $\pi(i)$ and $\pi^{-1}(i)$ queries can be supported efficiently for $1 \le i \le n$. A natural setting to consider this problem can be the indexing cell probe model, where the nodes of the decision tree are labeled "$\pi(i)$=?", and edges are labeled by the numbers from $[n]$. In other

words, the permutation is stored in "raw form", the value $S[i]$ of the $i$-th cell of storage is $\pi(i)$ (so that the $\pi(\cdot)$ queries are trivial to implement). The algorithm that implements $\pi^{-1}(i)$ is allowed to keep a small index, and perform cell probes to $S$ for a time cost of 1.

**Theorem 4** (Theorem 5, Munro et al. [25])**.** *For the permutations problem in the indexing cell probe model, if $t = o(\lg n / \lg\lg n)$, then $t(r+1) = \Omega(n \lg n)$.*

This result is derived from Theorem 1 by converting cell probes into bit probes. A disadvantage of these results is that the range of applicability is quite small: $t = o(\lg n / \lg\lg n)$. It is not unreasonable to have running times higher than this, for example, an interesting data structure based on Benes networks proposed by Munro et al. [25] offers very little redundancy space in exchange for running time $t = O(\lg n / \lg\lg n)$ (this is not an indexing data structure, and we leave its discussion outside the scope of the paper). We first show how to apply the results of [11] to the permutations problem obtaining

**Theorem 5.** *For the permutations problem in the indexing cell probe model, if $t = o(\sqrt{n})$, then $t(r+1) \geq n \lg \dfrac{n}{(t+1)^2}$.*

Yao [34, Theorem 2] also shows a similar lower bound for inverting 1-cycle permutations and functions $[n] \mapsto [n]$ in the randomized case. The proof was omitted from [34] (extended abstract) and to the best of our knowledge did not appear in a journal version.

However, the results of [11] cannot be directly applied to improve the lower bound of [5] for the substring report problem. Our contribution is a new technique that allows to apply these results to the substring report problem and in conjunction with ideas from [5] obtain a stronger lower bound for the substring report problem, Theorem 3.

In the second part of the paper, we consider lower bounds in Yao's bit probe model without the indexing assumption. There are only very few non-trivial lower bounds for data structures in this model. We are aware of only one famous example of lower bounds for the predecessor problem, e.g. [1, 23, 24, 3, 30, 26]. In this problem, we are to store a set $X$ of $n$ numbers from the universe $[m]$, and for $i \in [m]$, answer a query of the form "what is the largest number from $X$ that is smaller than $i$?" However these bounds are intrinsically developed for the cell probe model with the word size $w = \Omega(\lg n)$. The only example of a non-trivial lower bound in Yao's bit probe complexity that we are aware of is the polynomial evaluation problem [10] that can be stated as follows. Given a polynomial $P \in \mathbb{F}[x]$ of degree $d$ over a field $\mathbb{F}$ of order $n$, represent it so that values $P(i)$ can be computed efficiently for $i \in \mathbb{F}$. The decision trees for queries $P(i)$ do not have the top node corresponding to the index, all other nodes are labeled "$S[l]$=?" for $1 \leq l \leq s$, where $s = |S|$ is the space cost. The depth of tree $t$ is the time cost. In this model, Gal and Miltersen [10] showed the following

**Theorem 6** (Gal and Miltersen [10])**.** *For the polynomial evaluation problem in the non-indexing model $s \geq (1 + 1/3t)\Upsilon$, where $\Upsilon = (d+1)\lg n$ is the information-theoretic lower bound for storing such a polynomial.*

For the case $t = \Theta(\lg n)$, their lower bound amounts to $s = \Upsilon + \Omega(\Upsilon / \lg n)$. Our general approach to tackle this problem is somewhat similar to [10] and ideas in [11, 5]. Our contribution is to combine these ideas with a new lemma about families of sets (Lemma 2) that might be of independent interest. We are able to obtain bounds of the form $s = \Upsilon + \Omega(\Upsilon)$ for the case of polynomials of degrees $d = n^{\Theta(1)}$. This is the first result that allows to show a bound of this form for a problem in the bit probe model without the indexing assumption. We show some partial progress on this problem

**Theorem 7.** *For the polynomial evaluation problem (in Yao's bits probe model without indexing assumption) with parameters $s$, $t = R \lg n$ and $d = n^\alpha$, we have the bound $s \geq cd \lg n$ for a choice of parameters $R$, $c$, and $\alpha$, such that $R/c < -\ln(1 - \alpha/c)$.*

For example, for polynomials of degree $d = n^{3/4}$, and running times $R = 1.25 \lg n$, we can show that a storage of size at least $1.1\Upsilon$ bits is required. Similarly to [10], our results also naturally extend to the case of any problem with an error correcting property. We belive that one might be able to prove stronger results for this problem by using an improved version of the Lemma 2 in the discrete case.

## 2 Lower Bounds for Indexing Data Structures

We start by considering the permutations problem. Gennaro and Trevsian [11, Lemma 1] showed that given a permutation $\pi$ on $n$ elements and an algorithm $A$ that computes $\pi^{-1}(i)$ with the time cost $t$ using no auxiliary data (i.e. index), it is possible to encode $\pi$ using $2 \lg \binom{n}{a} + \lg((n-a)!)$ bits, where $a = n/(t+1)$. It is not hard to generalize this lemma to the case where $A$ is allowed to store an index of size $r$. We first show a different proof of this lemma in the spirit of the results of Demaine and López-Ortiz [5]. An advantage of this proof is that we are able to extend it for the substring report problem in the indexing bit probe model.

**Lemma 1.** *Given algorithm $A$ computes $\pi^{-1}(i)$ for $i \in [n]$ with time cost $t$ and the space cost $r$, it is possible to encode any $\pi$ on $n$ elements with $\lg \binom{n}{a} + \lg(n!/a!) + r$ bits, where $a = n/(t+1)$.*

*Proof.* Let $S$ denote the "raw storage" for $\pi$, $S[j] = \pi(j)$. Initially all the cells in $S$ are marked as being not *discovered*. We simulate $A$ on queries $\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(n)$ in this order. In the process of these simulations, we mark some of the cells as discovered; at the end of the simulations, all the cells are discovered. Also, we record a list of cells $\mathcal{P}$, and a binary list $\mathcal{L}$. If a simulation probes an undiscovered cell, we mark it as discovered, append the value stored in this cell to the list $\mathcal{P}$ and append 0 to the list $L$. We make an exception from this rule: if we simulate a query $q$ that computes $\pi^{-1}(i)$ and $A$ happens to probe a cell $j$, such that $S[j] = i$, then we (i) do not record the value $i$ in $\mathcal{P}$, (ii) terminate $A$ immediately, (iii) append 1 to $\mathcal{L}$ (to mark the end of the simulation), and (iv) mark $S[j]$ as discovered. We call $S[j]$ the *target cell* of $q$. If the simulation $\pi^{-1}(i)$ successfully terminates with the answer $j$, then we mark the target cell $S[j]$ discovered, and append 1 to $\mathcal{L}$. Before running the next simulation $q$ in the list, say $\pi^{-1}(i)$, we first check whether its target cell was discovered earlier; if so, then we call the simulation $q$ *absent* and skip it without recording anything to $\mathcal{P}$ and $\mathcal{L}$ (since its result is already known), otherwise we call it *present*, and proceed with the execution of the algorithm $A$ on the query $q$. Note that the simulations that were terminated are also present.

Let $a$ be the number of present simulations. Note that $\mathcal{L}$ is $n$ bits long, exactly $a$ of which are 1-bits (marking the stops of present simulations). The list $\mathcal{P}$ consists of $n - a$ cells, since for each present simulation, we mark exactly one cell the value of which is not recorded in $\mathcal{P}$ as discovered. Since the distances between consecutive 1-bits in $\mathcal{L}$ cannot be larger than $t$ (time complexity of $A$), we have $a \leq n/(t+1)$. We can store $\mathcal{P}$ using the information-theoretic minimum of $\lceil \lg(n!/a!) \rceil$ bits, since the same value does not appear twice in $\mathcal{P}$. $\mathcal{L}$ can be stored using $\lg \binom{n}{a}$ bits.

The decoding algorithm also simulates the queries $\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(n)$ in this order. It starts with an uninitialized storage $S$. Before simulating a query $q$ for $\pi^{-1}(i)$, we check whether the value $i$ is already present in $S$, if so, we skip the simulation ($q$ is absent in this case). Whenever $A$ makes a probe to an uninitialized location $l$, we first read the next bit from $\mathcal{L}$, if its value is 0 then

we read the next element from $\mathcal{P}$ and store it in $S[l]$, otherwise we store $i$ in $S[l]$ and terminate the simulation. When $A$ terminates with an answer $j$, we store $j$ in $S[l]$ and skip the next bit in $\mathcal{L}$. □

We present the pseudocode for the encoding/decoding algorithms in the appendix for completeness (Algorithm 1 and Algorithm 2). Using this lemma, it is not hard to show Theorem 5.

*Proof.* There are at least a constant fraction of permutations that have high Kolmogorov complexity [20] of $K = \lg n! - O(1)$, and therefore cannot be described using less than $K$ bits. Thus, for such permutations, the length of our encoding is

$$\lg\left(\frac{n!}{(n/t)!}\right) + \lg\binom{n}{n/t} + r \geq \lg n! - O(1)$$

Therefore $r \geq (n/t)\lg(n/t^2) - \Theta(n/t)$. □

Now we present the proof of Theorem 3.

*Proof.* Let $A$ be an algorithm that implements the substring search query on a string of length $L$ with time cost $t$ and space cost $r$. We set $L = n(2\lg n + 3)$. Let $\pi$ be a permutation on $n$ elements. We will encode $\pi$ as a bit vector $B$ of length $L$:

$$B = \underbrace{0\,\mathtt{binary}_{\lceil \lg \mathtt{n} \rceil}(\pi(\mathtt{1}) - \mathtt{1})\,0}_{\lceil \lg n \rceil + 2 \text{ bits}} \quad \underbrace{11\ldots1}_{\lceil \lg n \rceil + 1 \text{ bits}} \quad \ldots \quad \underbrace{0\,\mathtt{binary}_{\lceil \lg \mathtt{n} \rceil}(\pi(\mathtt{n}) - \mathtt{1})\,0}_{\lceil \lg n \rceil + 2 \text{ bits}} \quad \underbrace{11\ldots1}_{\lceil \lg n \rceil + 1 \text{ bits}} \tag{1}$$

where $\mathtt{binary}_{\lceil \lg \mathtt{n} \rceil}(\mathtt{x})$ is the binary representation for $x$ using $\lceil \lg n \rceil$ bits padded with leading zeroes as necessary. The part of $B$ that encodes $\pi(i) - 1$ is called the *i-th chunk*. The bits of $B$ that do not depend on $\pi$ are called *separators*. For each $i = 1, 2, \ldots, n$ in that order, we simulate the query that searches for the pattern "$q_i = \mathtt{binary}_{\lceil \lg \mathtt{n} \rceil}(\mathtt{i} - \mathtt{1})0$". In the process of these simulations, we mark some of the bits in $B$ as *discovered*. Initially, all the separator bits are marked as discovered. We build the bit vectors $\mathcal{P}$ and $\mathcal{L}$ similarly to the proof of Lemma 1. However, in the bit probe model we have five differences: (i) If a simulation $q_i$ probes a bit in its *target chunk*, the chunk that encodes $i - 1$ in binary, we terminate the simulation just before such probe is made and append a 1-bit to $\mathcal{L}$. (ii) We call $q_i$ *absent* if its target chunk has more than $k$ discovered bits just before running $q_i$ (we do not perform such simulations). (iii) At the end of all the simulations, it might happen that not all the bits in $B$ are discovered, we collect all such bits from left to right in the bit vector $\mathcal{R}$. (iv) We also need to store which simulations are present in a bit vector $\mathcal{S}$, since the decoding algorithm might not be able to identify whether the target chunk of $q_i$ has at least $k$ discovered bits. (v) Also, some of the locations might be marked as discovered twice: once when probed by the simulation of a query $q_i$, and the second time when it is a part of the target chunk of another present query $q_j$; we call such locations *overlaps*. By the construction, there are at most $k$ such bits per query $q_j$ (otherwise, $q_j$ is called absent and skipped during simulations). In [5], they stored these bit vectors directly and showed Theorem 1.

We now describe a new compression technique. Let $a$ be the number of present simulations. We introduce an additional bit vector $\mathcal{C}'$ of length $n$, $\mathcal{C}'[i] = 1$ if the chunk $i$ was the target chunk of a present simulation, we call such chunks *present*, and otherwise *absent*. In other words, the chunk $j$ is present if and only if it had at most $k$ discovered bits at the moment when the simulation for the query $q_{\pi(j)}$ was about to execute. We encode all the bits in all the absent chunks in the bit vector $\mathcal{R}'$ from left to right. Note that these bits are of two types: all the bits recorded in $\mathcal{R}$, and some of the bits from $\mathcal{P}$. The bit vector $\mathcal{C}'$ can be encoded using using $\lceil \lg \binom{n}{a} \rceil$ bits of space. The bit vector

$\mathcal{R}'$ can be compressed to $\lceil \lg(n!/a!) \rceil$ bits, since it encodes $n-a$ distinct numbers from $[n]$. Using $\mathcal{C}'$ and $\mathcal{R}'$ we can restore the contents and the locations of all the absent chunks. Note that storing the bit vector $\mathcal{S}$ is no longer necessary, since the $i$-th simulation is present if and only if the chunk encoding $i-1$ is not among the absent chunks. Also, we can remove all the bits that are located in the absent chunks from $\mathcal{P}$ obtaining $\mathcal{P}'$ (since those bits are already encoded in $\mathcal{R}'$); respectively, we remove the corresponding 0-bits from $\mathcal{L}$ obtaining $\mathcal{L}'$. Note that $\mathcal{P}'$ consists of overlap bits only. Denote the number of overlap bits by $v = |\mathcal{P}'|$. Since in each present chunk we have at most $k$ overlap bits, it follows that $|\mathcal{P}'| \le ak$. The bit vector $\mathcal{L}'$ consists of the concatenation of bit vectors $1^{f_i'}0$ for all present queries $q_i$ in the increasing order of $i$. The value $f_i'$ denotes the number of bits that corresponds to $q_i$ in $\mathcal{P}'$, that is, the bits that were recorded in $\mathcal{P}$ during the simulation of $q_i$, and were not removed from $\mathcal{P}$ later (i.e. the overlap bits). The length of $\mathcal{L}'$ is $|\mathcal{P}'| + a$, and $a$ of them are 1-bits. Thus, we can encode $\mathcal{L}'$ using only $\lg \binom{ak+a}{a}$ bits of space. Using the index $I$, the algorithm $A$, and the bit vectors $\mathcal{P}'$, $\mathcal{R}'$, $\mathcal{L}'$, and $\mathcal{C}'$, we can decode the permutation $\pi$: start by decoding the absent chunks, and then follow the idea in the proof of Lemma 1. Consider permutations that are hard in Kolmogorov's sense, the length of their encoding is

$$|I| + |\mathcal{R}'| + |\mathcal{C}'| + |\mathcal{L}'| + |\mathcal{P}'| \ge \lg(n!) - O(1) \tag{2}$$

We estimate all the sizes on the right side up to an $O(a)$ additive term:

$$\begin{aligned} |I| &\ge \lg(n!) - \lg(n!/a!) - \lg\binom{n}{a} - \lg\binom{ak+a}{a} - v \\ &\ge a\lg a - a\lg\frac{n}{a} - a\lg k - v - \Theta(a) = a\lg\frac{a^2}{nk} - v - \Theta(a) \\ &= aD - v - \Theta(a), \end{aligned}$$

where $D = \lg(a^2/(nk))$. The total number of bits $A$ probed during $a$ present simulations is at most $ta$; $v$ of these bits were probed in the present chunks and hence at most $ta - v$ in the absent ones. However, there are at least $(k+1)(n-a)$ bits probed in the absent chunks by definition of the absent chunks. We conclude that $ta - v > k(n-a)$ and thus $a > \dfrac{nk+v}{t+k}$. Therefore,

$$D > \lg(nk/(t+k)^2) \ge \lg n - 2(\lg t) - \lg\lg n$$

since $k \le \lceil \lg n \rceil$. To derive a meaningful lower bound, we require $D = \omega(1)$, so that $(n\lg n)/t^2 = \omega(1)$ choosing $k = \Theta(\lg n)$, in other words $t = o(\sqrt{n\lg n})$.

We will minimize the function $aD - v$ subject to the linear constraints $0 \le v \le ak$, and $a > (nk+v)/(t+k)$. These constraints define two points on the $(a,v)$-plane that correspond to the vertices of the feasible set of the corresponding linear program (the variables $n$, $t$, $k$, and $D$ are fixed) where the minimum can be reached: $(nk/(t+k), 0)$ and $(nk/t, nk^2/t)$. We maximize the resulting expression over all possible choices of parameter $k$. It follows that

$$aD - v \ge \max_k \min\left\{\frac{nkD}{t+k}, \frac{nk}{t}(D-k)\right\}$$

The first term is bigger iff $D/(t+k) > (D-k)/t$ iff $Dt > Dt - tk + kD - k^2$ iff $k > D - t$. There are two candidates for the max: $k = D/2$ (the second term is quadratic in $k$) and $k = D - t$ (the first term is an increasing function of $k$). If $D/2 < D - t$, then $k = D - t$ and $\max = n(D-t)$,

otherwise $k = D/2$, and $\max = nD^2/4t$. Hence,

$$|I| \geq aD - v - \Theta(a) \geq \begin{cases} n(D-t) - \Theta\left(\frac{n(D-t)}{t}\right) & \text{for } t < \frac{D}{2} \\ \frac{nD^2}{4t} - \Theta\left(\frac{nD}{t}\right) & \text{otherwise} \end{cases}$$

where $D = \lg n - 2\lg t - \lg\lg n$. We consider running times $t \geq \lg n$ and omit the first branch.

We can also reduce the length of the bit vector $B$ by using separators of the form $01^{\lg\lg n}0$ instead of $01^{\lceil \lg n \rceil + 1}0$ without losing the asymptotic bounds. We give the proof of this result in the appendix. □

Note that for an interesting case $t \sim c \lg L$, $c > 1$, we obtain a bound $r \geq L/4c - \Theta(L/(\lg L))$ is always better (in terms of constants) than the lower bound for the similar case shown by Demaine and López-Ortiz [5], namely $r \geq L(1 + 2c - 2\sqrt{c(1+c)}) - \Theta((L \lg\lg L)/\lg L)$. We also present the pseudocode for the encoding/decoding algorithms in the appendix for completeness (Algorithm 3 and Algorithm 4).

# 3 Lower Bounds for Non-Indexing Data Structures

In this section, we consider the polynomial evaluation problem. For convenience, we say that a polynomial $P(x) := a_0 + a_1 x + \ldots a_{d-1} x^{d-1}$ has degree $d$, the coefficients belong to some finite field $\mathbb{F}$ of order $n$. The problem is to represent this polynomial so that the queries $P(i)$ can be implemented efficiently for $i \in P$.

We use the following notation: $\Upsilon = d \lg n$ is the information-theoretic lower bound to store a polynomial of degree $d$ over $\mathbb{F}$; $s$ is the space cost of our data structure storing a polynomial; $r = s - \Upsilon$ is *redundancy*; $t$ is the running time of the algorithm implementing queries $P(i)$ for $i \in \mathbb{F}$; $R = t/\lg n$.

## 3.1 Families of Sets with Large Unions

We first discuss the problem that concerns families of sets. The problem is as follows: we are given a universe of size $s$, and a family of $n$ subsets of size $t$ each. We say that this family satisfies the large union condition if the union of every subfamily of $d$ subsets (from our given family) is of size at least $z$. The question is: under what condition of parameters $s$, $n$, $t$, $d$, and $z$ a family of sets with the large union condition exists. The values $s$, $m$, $n$, $t$ correspond to their counterparts defined earlier for the polynomial evaluation problem, so we use this overlapping notation.

We can answer this question in the following two simple cases: (i) if $z = t$ then the universe of size $s = t$ is large enough; and (ii) if $z = dt$ then all sets are required to be disjoint, and we require the of size at least $s = nt$. An interesting question is to solve this problem when $t < z < dt$. This problem can be stated in two flavors: for discrete sets (indivisible elements of measure 1) and for sets with continuous measure. In this section, we are able to provide an answer to this question in the case of continuous measure sets using linear programming. Our approach also gives non-trivial bounds for the discrete case as well, however, we believe that the integrality gap of the linear program is large, and one should be able to obtain stronger results for this case.

**Lemma 2.** *Let $S_1, S_2, \ldots, S_n$ be a family of $n$ subsets of a set $S$ with a measure $\mu$. Let $\mu(S_i) = t$ for some fixed $t > 0$. Assume that the measure of union of every subfamily of $d$ sets $\mu(S_{i_1} \cup S_{i_2} \cup$*

$\ldots \cup S_{i_d}) \geq z$ *for some* $z > 0$*, and all* $1 \leq i_1 < i_2 < \ldots i_n \leq n$*. Then* $\mu(S_1 \cup S_2 \cup \ldots \cup S_n) \geq s(z)$*, where* $s$ *is a piecewise linear function of* $z$ *with the following* $n$ *vertices on the* $(z, s)$*-plane:*

$$V_i = \left\{ \frac{nt}{i} \left( 1 - \frac{\binom{n-d}{i}}{\binom{n}{i}} \right), \frac{nt}{i} \right\}$$

*for* $1 \leq i \leq n$*. If* $s \geq s(z)$*, we can also construct* $S$*,* $\mu$*, and* $S_1, S_2, \ldots, S_n$ *that satisfy the above conditions.*

*Proof.* Define the following notation: let $\alpha$ be a subset of $[n]$, then $S^{\cup}_{\alpha} = \cup_{i \in \alpha} \cup S_i$. We can rewrite our problem as

$$\text{Find } s(z) = \min S^{\cup}_{[n]} \text{ subject to } \left\{ \begin{array}{lcll} \mu(S_i) & = & t & \text{for all } i \in [n] \\ \mu(S^{\cup}_{\alpha}) & \geq & z & \text{for all } \alpha \subset [n] \text{ such that } |\alpha| = d \end{array} \right.$$

We can formulate this problem as a linear program. Let $\beta$ be a vertex of the hypercube $\{+1, -1\}^n$, define

$$Q_\beta = \mu(S^{\beta_1} \cap S^{\beta_2} \cap \ldots \cap S^{\beta_n})$$

where $X^{+1}$ is defined as $X$, and $X^{-1}$ is defined as $S^{\cup}_{[n]} \setminus X$. Note that $Q_{(-1,-1,\ldots,-1)}$ is always 0. Given a set of variables $Q'_\beta$ for all $\beta \in H$, where $H = \{+1, -1\}^n \setminus \{(-1, -1, \ldots, -1)\}$, we claim that the there exists a family of $n$ sets, so that $Q_\beta = Q'_\beta$ for all $\beta \in H$ if and only if $Q'_\beta \geq 0$ for all $\beta \in H$. Informally, the values $Q_\beta$ form "elementary blocks" of our family of sets $\{S_i\}$: given $Q_\beta$'s, we can express any possible combination of unions, intersections, or set differences of $S_i$'s as a sum of $Q_\beta$'s. Thus, we can also express $\mu(S^{\cup}_{\alpha})$ for every $\alpha$ as follows

$$\mu(S^{\cup}_{\alpha}) = \sum \{Q_\beta | \text{ there exist an index } i \in \alpha, \text{ such that } \beta_i = +1\} \tag{3}$$

(the union of a subfamily of sets $\alpha$ consists of all the blocks $Q_\beta$ that have at least one of the sets $i \in \alpha$ with $+1$ sign in $\beta$). Thus, we can rewrite our problem as a linear program with variables $Q_\beta$.

To solve it, we first reduce the number of variables by observing that this linear program is symmetric upon permutations of the indices of the sets. Namely let $\pi$ be a permutation on $[n]$. Define $\pi Q_\beta = Q_{\pi\beta}$, where $\pi\beta_i = \beta_{\pi(i)}$ is permutation of the indexes. We claim that if $Q_\beta$ form a feasible point of the linear program, then $\pi Q_\beta$ is a feasible point as well. If $\pi$ acts on $\pi Q_\beta$, then the conditions (3) on $\mu(S^{\cup}_{\alpha})$ are also going to permute according to $\pi$. Namely the condition $\mu(S^{\cup}_{\alpha}) \geq z$ in variables $Q_\beta$ is equivalent to the condition $S^{\cup}_{\pi\alpha} \geq z$ in permuted variables $\pi Q_\beta$. The conditions (3) on $\mu(S_i)$ are going to permute similarly. Note that the objective function is invariant with respect to permutations of indices. Therefore, if $Q_\beta$ is a minimum of our linear program, then $\pi Q_\beta$ is also a minimum for any permutation $\pi$. We consider all possible permutation of indices of $Q_\beta$ and take their convex combination

$$Q^s_\beta = \frac{1}{n!} \sum_{\pi} Q_{\pi\beta}$$

Note that $Q^s$ is also an optimum of the linear program, and it is also invariant with respect to permutation of indices, namely $Q^s_\beta$ depends only on the number of $+1$'s in $\beta$. We can define the new set of variables $Q_i = Q^s_\beta$ where $\beta$ has exactly $i$ ones in it, and rewrite the linear program in terms of $Q_i$'s.

$$\text{Find } s(z) = \min S^{\cup}_{n} \text{ subject to } \left\{ \begin{array}{lcll} Q_i & \geq & 0 & \text{for all } i \in [n] \\ \mu(S^{\cup}_{1}) & = & t & \\ \mu(S^{\cup}_{d}) & \geq & z & \end{array} \right.$$

where $\mu(S_j^{\cup})$ denotes the measure of the union of any $j$ sets ($\mu(S_j^{\cup})$ is also invariant under permutations of indices in the new variables $Q_\beta^s$). Using (3), we can express $\mu(S_j^{\cup})$ as a sum of $Q_i$ as follows

$$\mu(S_j^{\cup}) = \sum_i \left( \binom{n}{i} - \binom{n-j}{i} \right) Q_i$$

where $\binom{n}{i} - \binom{n-j}{i}$ is the number of ways to choose a set of size $i$ so that it intersects with a given set of size $j$ from a universe of size $n$.

The feasible set of our linear program can be viewed as the $n-1$ dimensional simplex that is bounded by the "fixed" constraints $Q_i \geq 0$ and $\mu(S_1^{\cup}) = t$ intersected with the "sliding" hyperspace $\mu(S_d^{\cup}) \geq z$. If $z \leq t$, then our hyperspace contains the whole simplex, and the solution is clearly $s(z) = \min \mu(S_n^{\cup}) = t$ at the vertex $V_n'$ with the coordinates $Q_n = t$ and $Q_i = 0$ for $i < n$ (all the sets coincide with each other). The biggest value of $z$ where the hyperspace still intersects with the simplex is $z = dt$, and the solution is $s(z) = nt$ at the vertex $V_1'$ with the coordinates $Q_1 = t$, and $Q_i = 0$ for $i > 1$ (all the sets are disjoint). We can show that $s(z)$ is a piecewise linear function of the parameter $z$, and the $i$-th vertex of it corresponds to the vertex of the simplex where all coordinates are zero except for the coordinate $Q_i$. The linear function between the vertices $V_i$ and $V_{i+1}$ on $(z, s)$-plane correspond to edge of the simplex between the corresponding vertices $V_i'$ and $V_{i+1}'$ of the simplex. At $i$-th vertex $V_i'$, we have

$$\begin{aligned} Q_i &= \frac{t}{\binom{n}{i} - \binom{n-1}{i}} = \frac{nt}{i\binom{n}{i}} \\ z &= \mu(S_d^{\cup}) = \frac{nt}{i} \left( 1 - \frac{\binom{n-d}{i}}{\binom{n}{i}} \right) \\ s(z) &= \mu(S_n^{\cup}) = \frac{nt}{i} \end{aligned}$$

The statement of the lemma follows. □

An example of such function $s(z)$ is shown in the appendix, Section 5.2.

### 3.2 Implications to the Lower Bounds

We start by describing the techniques from [10]. Let $S$ be a data structure for the polynomial evaluation problem. They pick $E$, a set of cardinality $r+1$, randomly and uniformly chosen from $[s]$; and erase the locations indexed by $E$ in $S$. For given polynomial, each query $P(i)$ uses some fixed set $Y_i$ of locations in $S$, $|Y_i| \leq t$. If the probability of the set $E$ to intersect the fixed set $Y_i$ is small enough, then with probability $2/3$ there are at least $d$ sets $Y_i$ that do not intersect $E$. Using the probabilistic method, there exists a choice of $E$ such that for at least $2/3$ fraction of polynomials, we are still able to perform $d$ queries after erasing the locations of $E$. However, a polynomial $P$ of degree at most $d$ is uniquely identified by its values at any given set of $d$ points of $\mathbb{F}$. We arrive at the conclusion that are are able to decode $2/3$ fraction of polynomials using just $\Upsilon - 1$ bits which contradicts the simple counting argument.

We can allow the set $E_P$ of the positions of erased bits depend on the polynomial $P$ in question. The seeming disadvantage of this approach is that we need to encode the set $E_P$ together with the values of all the bits that we did not erase in order to decode $P$. An alternative approach could be to consider a set of $d$ queries $P(i_1), P(i_2), \ldots, P(i_d)$ and encode indices $i_1, i_2, \ldots, i_d$ together with the values of the bits of $S$ that these queries need to probe. This approach is somewhat similar to

the proof of Lemma 1: for each query $P(i_j)$ in the increasing order of $i_j$, we encode the sequence of probed bits that this query made to $S$ in a list of bits $\mathcal{P}$. Once a bit is appended to $\mathcal{P}$, the corresponding location in $S$ is marked as discovered. If a probe made to a discovered location, we do not duplicate its value in $\mathcal{P}$ and proceed to the next probe. The decoding algorithm is also similar to the proof of Lemma 1, and we will omit it. The number of bits recorded in $\mathcal{P}$ is $|Y_{i_1} \cup Y_{i_2} \cup \ldots \cup Y_{i_d}|$, and the indices $i_1, i_2, \ldots, i_d$ can be encoded using $\lceil \lg \binom{n}{d} \rceil$ bits. To obtain a contradiction, we need to be able to find a set of indices such that $|\mathcal{P}| < d \lg n - \lg \binom{n}{d} - 1$. We denote $z = d \lg n - \lg \binom{n}{d} - 1 = d \lg d - \Theta(d)$. On the other hand, Lemma 2 claims that if we are not able to find such indices for every polynomial, then the size of our storage $s = |S|$ is at least $s(z)$. More precisely, if $|S| < s(z)$, then an abstract family of sets that satisfies the large union condition does not exists, and hence there is no family of discrete sets $Y_i$ that satisfies the large union condition. Therefore, we are able to find a set of indices $i_1, i_2, \ldots, i_d$, such that $|\mathcal{P}| < z$; and encode every polynomial of degree at most $d$ using less than $d \lg n$ bits as described above, obtaining a contradiction.

Let us choose $d = n^\alpha$ for $0 < \alpha < 1$. Denote $p_0 = \binom{n-d}{i} / \binom{n}{i}$, and $p_1 = 1 - p_0$ ($p_1$ can be interpreted as the probability of two uniformly chosen random sets of sizes $i$ and $d$ intersecting in a universe of size $n$) for $i$, $1 \le i \le n$. We substitute $z = d \lg d - \Theta(d) = \alpha d \lg n - \Theta(d)$ into Lemma 2. To obtain bound $s \ge s(z)$, we need to find the smallest index $i$ such that

$$\frac{ntp_1}{i} \le z, \tag{4}$$

so that $s(z) = nt/i$. We can estimate

$$\begin{aligned} p_0 &= \frac{(n-d)(n-d-1)\ldots(n-d-i+1)}{n(n-1)\ldots(n-i+1)} = \left(1 - \frac{d}{n}\right)\left(1 - \frac{d}{n-1}\right)\ldots\left(1 - \frac{d}{n-i+1}\right) \\ &= \left(1 - \frac{d}{n-\gamma}\right)^i \sim \exp\left(-\frac{id}{n-\gamma}\right) \sim \exp\left(-\frac{id}{n}\right) \end{aligned}$$

for $i = o(n)$, where $\gamma$ is some number, $0 < \gamma < i$. Let us try to choose the parameters so that $s(z) = nt/i = z/p_1 - o(z/p_1) \ge c\Upsilon$ for some constant $c > 1$. Equivalently, we need $p_1 \le z/(c\Upsilon) \sim \alpha/c$, this constraint reduces to

$$i \ge \frac{n}{d} \ln\left(\frac{1}{1 - \alpha/c}\right), \tag{5}$$

where "ln" denotes the natural logarithm. Now we need to check that $i$ chosen according to this constraint also satisfies (4) condition. Let us substitute (5) into $ntp_1/i$ and use the bound for $p_1$

$$\frac{ntp_1}{i} \sim \frac{dtp_1}{\ln\left(\frac{1}{1-\alpha/c}\right)} \le \frac{dtz}{c\Upsilon \ln\left(\frac{1}{1-\alpha/c}\right)} = \frac{Rz}{c \ln\left(\frac{1}{1-\alpha/c}\right)},$$

recall that $R = t/\lg n$, $R > 1$. The right part is less than $z$ when

$$\frac{R}{\alpha} < \frac{c}{\alpha} \ln\left(\frac{1}{1 - \alpha/c}\right) = f\left(\frac{c}{\alpha}\right),$$

where $f(x) = -x \ln(1 - 1/x)$. For fixed values of $\alpha$ and $R$, we can find $c/\alpha$ as a solution to the equation $f(c/\alpha) = \frac{R}{c}$ (for $x > 1$, $f$ is a monotonically decreasing function of $x$). This method will yield a linear lower bound $s \ge c\Upsilon$ for some $c > 1$ if $R < -\ln(1 - \alpha)$, and so $\alpha > 1 - 1/e > 0.64$. We conclude with the statement of Theorem 7.

## 4 Acknowledgments

We thank Alejandro López-Ortiz for fruitful discussions and pointing out an error in an early draft of the paper. We thank Prabhakar Ragde and Ian Munro for their help. We also thank anonymous FOCS 2007 referees for pointing out the work of Yao [34], and Gennaro and Trevsian [11], and suggestions to improve the paper.

## References

[1] Miklós Ajtai. A lower bound for finding predecessors in yao's cell probe model. *Combinatorica*, 8(3):235–247, 1988.

[2] Jérémy Barbay, Meng He, J. Ian Munro, and S. Srinivasa Rao. Succinct indexes for strings, binary relations and multi-labeled trees. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 680–689, 2007.

[3] Paul Beame and Faith Ellen. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002.

[4] Erik D. Demaine and Alejandro López-Ortiz. A linear lower bound on index size for text retrieval. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 289–294, 2001.

[5] Erik D. Demaine and Alejandro López-Ortiz. A linear lower bound on index size for text retrieval. *Journal of Algorithms*, 48(1):2–15, 2003.

[6] Martin Farach-Colton. Optimal suffix tree construction with large alphabets. In *IEEE Symposium on Foundations of Computer Science*, pages 137–143, 1997.

[7] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings of 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 390–398, 2000.

[8] Paolo Ferragina and Giovanni Manzini. An experimental study of an opportunistic index. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 269–278, 2001.

[9] Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. An alphabet-friendly FM-index. In *Proceedings of the 11th Symposium on String Processing and Information Retrieval*, pages 150–160, 2004.

[10] Anna Gál and Peter Bro Miltersen. The cell probe complexity of succinct data structures. In *International Colloquium on Automata, Languages and Programming*, pages 332–344, 2003.

[11] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *IEEE Symposium on Foundations of Computer Science*, pages 305–313, 2000.

[12] Robert Giegerich and Stefan Kurtz. From ukkonen to mccreight and weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19(3):331–353, 1997.

[13] Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: a tool for text indexing. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 368–373, 2006.

[14] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 841–850, 2003.

[15] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *ACM Symposium on Theory of Computing*, pages 397–406, 2000.

[16] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal of Computation*, 35(2):378–407, 2005.

[17] Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. USA: Cambridge University Press, 1997.

[18] Juha Kärkkäinen and Peter Sanders. Simple linear work suffix array construction. In *International Colloquium on Automata, Languages and Programming*, pages 943–955, 2003.

[19] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[20] P. Vitányi M. Li. *An Introduction to Kolmogorov Complexity and its Applications, 2nd edition*. Springer-Verlag, New York, 1997.

[21] Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[22] Edward M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[23] Peter Bro Miltersen. Lower bounds for union-split-find related problems on random access machines. In *Symposium on Theory of Computing*, pages 625–634, 1994.

[24] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.

[25] J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct representations of permutations. In *International Colloquium on Automata, Languages and Programming*, pages 345–356, 2003.

[26] Mihai Patrascu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Symposium on Theory of Computing*, pages 232–240, 2006.

[27] Anna Patterson. We wanted something special for our birthday. `http://googleblog.blogspot.com/2005/09/we-wanted-something-special-for-our.html`.

[28] Kunihiko Sadakane. Compressed text databases with efficient query algorithms based on the compressed suffix array. In *International Conference on Algorithms and Computation*, pages 410–421, 2000.

[29] Kunihiko Sadakane. Succinct representations of lcp information and improvements in the compressed suffix arrays. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 225–232, 2002.

[30] Pranab Sen. Lower bounds for predecessor searching in the cell probe model. In *IEEE Conference on Computational Complexity*, pages 73–83, 2003.

[31] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[32] P. Weiner. Linear pattern matching algorithms. In *IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[33] Andrew Chi-Chih Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981.

[34] Andrew Chi-Chih Yao. Coherent functions and program checkers (extended abstract). In *ACM Symposium on Theory of Computing*, pages 84–94, 1990.

# 5 Appendix

## 5.1 The function $s(z)$

In the figure 5.1, we illustrate the function $s(z)$ by specifying vertices $V_i$, the vertex $V_1$ has coordinates $(4, 10)$, and $V_{10}$ has coordinates $(1, 1)$, $V_2, V_3, \ldots, V_9$ are located from right to left. The specified family of abstract sets with parameters $n = 10$, $d = 4$, $t = 1$ exists if and only $s \leq s(z)$.

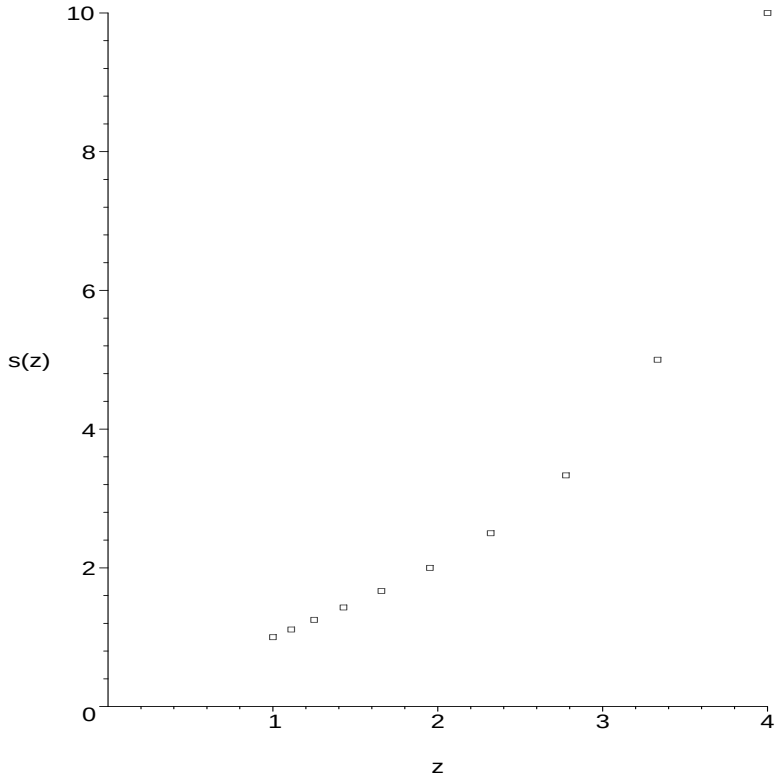


Figure 1: Example of $s(z)$ for $n = 10$, $d = 4$, $t = 1$

### 5.2 Proof of Theorem 3. Small separators.

*Proof.* We can reduce the length of the bit vector $B$ by using separators of the form $01^z0$ instead of $01^{\lceil \lg n \rceil + 1}0$ without losing the asymptotic bounds. In the following, for simplicity of the presentation, we assume that $n$ is a power of 2. We call a number $i \in [n]$ *valid* if its binary encoding does not contain $1^z$ anywhere in it. Let $M_n^z$ be the set of all valid numbers. We choose $z = 2 \lg \lg n$, and bound

$$n' = |M_n^z| \geq n - (\lceil \lg n \rceil - z + 1) 2^{\lceil \lg n \rceil - z} \geq n - \frac{2n \lg n}{(\lg n)^2} = n - o(n)$$

by excluding all the numbers that have a substring $1^z$ starting at position $j$ for all the positions $j = 1, 2, \ldots, \lceil \lg n \rceil - z + 1$. We will encode a permutation on $[n']$ elements instead of $n$, and simulate queries for valid numbers only; note the set $M_n^{2 \lg \lg n}$ depends on $n$ only, so we do not need to encode it explicitly. Inequality (2) becomes

$$|I| \geq \lg(n'!) - \lg(n'!/a'!) - \lg \binom{n'}{a'} - \lg \binom{a'k' + a'}{a'} - v',$$

We use the same analysis as before and obtain

$$|I| \geq a'D' - v' - \Theta(a') \geq \begin{cases} n'(D' - t) - \Theta\left(\frac{n'D'}{t}\right) & \text{for } t < \frac{D'}{2} \\ \frac{n'(D')^2}{4t} - \Theta\left(\frac{n'D'}{t}\right) & \text{otherwise} \end{cases} \tag{6}$$

where $D' = \lg n' - 2 \lg t - \lg \lg n'$. The length of the bit vector $B'$ with short separators is

$$L' = n' \lg n + 2n' \lg \lg n + 2n' < n' \lg n' + 2n' \lg \lg n' + 3n'$$

and thus $n' > L' / \lg L'$, and $\lg n' > \lg L' - \lg \lg L'$. Substituting this into (6), we obtain

$$|I| \geq \left( \begin{cases} \frac{L'}{\lg L'}(D' - t) & \text{for } t < \frac{D'}{2} \\ \frac{L'(D')^2}{4t \lg L'} & \text{otherwise} \end{cases} \right) - \Theta\left(\frac{L'D'}{t \lg L'}\right), \tag{7}$$

where $D' = \lg L' - 2 \lg \lg L' - 2 \lg t$. The reasonable value for the running time $t$ is at least $\lg n'$ (to be able to check whether the occurrence we output is correct and to be able to read the input), so that $t \geq \lg n' > \lg L' - \lg \lg L' > D'/2$. Thus, it is natural to consider the second case only. The statement of the theorem follows. □

### 5.3 Related Results. Tightness of Lemma 2. Discussion

Using the probabilistic argument, we can show that lemma 2 even "somewhat tight" for discrete sets. , we show that for particular choices of $c$ and $R$, we can find a set system $S_i$ such that union of every subfamily of $d$ sets has the union of size at least $\Upsilon$. We estimate the probability $B$ that a random (uniformly chosen from universe of size $s = c\Upsilon$) family of $d$ sets has union at most $\Upsilon$, $B \leq \binom{s}{\Upsilon}\binom{\Upsilon}{t}^d / \binom{s}{t}^d$. The probability of an event that at least one subfamily of $d$ sets has union smaller than $\Upsilon$ is then at most $\binom{n}{d} B$. If this value is strictly smaller than 1 then there exist a choice of the required family of $n$ sets.

$$\log \frac{\binom{n}{d}\binom{S}{\Upsilon}\binom{\Upsilon}{t}^d}{\binom{s}{t}} d \lg \frac{ne}{d} + SH_2(1/c) + dt \lg \frac{\Upsilon e}{t} - dt \lg \frac{se}{t} \leq d \lg n (1 + cH_2(1/c) - R \lg c) - d \lg \frac{d}{e}$$

where $H_2(1/c)$ is binary entropy of $1/c$. We can choose constant $R > (1 + H_2(c))/\lg c$ large enough so that the expression in the bracket is negative. We believe that it is possible to turn such system of sets into an an artificial problem with an error correcting property. If so, then the linear lower bound is essentially optimal for this class of problems. Thus, to make further progress on the polynomial evaluation problem we either need to use some other (non-error correcting) properties or improve upper bounds.

Also we can easily show that Gal and Miltersen's [10] techniques for the polynomial evaluation problem can yield a slightly better lower bound than claimed in [10]:

$$r \geq \frac{s}{t} \lg nd \geq \frac{d \lg n}{t} \lg \frac{n}{d}$$

Using similar probabilistic method ideas and inspired by the proof of the set lemma, we can show that this lower bound is somewhat tight for $t = \Omega(\lg n \lg \lg n)$. The proof is not complicated, but slightly technical, so we leave it out for this version of the paper.

### 5.4 of the encoding/decoding algorithms

**Algorithm 1** Encode Permutation

$\mathcal{P}$ is a sequence of cells, initially empty
$\mathcal{L}$ is a sequence of bits, initially empty
**for all** $i = 1, 2, \ldots, n$ **do**
  **if** location $\pi^{-1}(i)$ is not marked as probed **then**
    Call simulation $i$ *present*
    Execute $A$ on the query $\pi^{-1}(i)$
    Let $l_1, l_2, \ldots, l_z$ denote the locations of the cells that $A$ inspected in order of its execution ($z \leq t$)
    Let $j$ be the target cell ($j = \pi^{-1}(i)$)
    **for all** $f = 1, 2, \ldots, z$ **do**
      **if** location $l_f$ is *not* marked as discovered **then**
        **if** $l_f = j$ **then**
          { Probing our target cell. Do not store $S[l_f]$ in $\mathcal{P}$ }
          Terminate the loop on $f$
        Append $S[l_f]$ to the end of $\mathcal{P}$
        Append 1 to the end of $\mathcal{L}$
        Mark $l_f$ as discovered { Same location is never recorded twice}
    Append 0 to the bit vector $\mathcal{L}$
    Mark $j$ as discovered

**Algorithm 2** Decode Permutation

Start with an uninitialized storage $S$
**for all** $i = 1, 2, \ldots, n$ **do**
  **if** the value $i$ does not occur in $S$ **then**
    { The simulation $i$ is present }
    Start simulating $A$ on the input $i$
    When $A$ needs to probe the cell at a location $l$
    **if** $S[l]$ is initialized **then**
      Provide the value $S[l]$ to the algorithm $A$
    **else**
      Read the next bit $g$ from $\mathcal{L}$
      **if** $g = 0$ **then**
        { This probe is made to the target cell}
        Initialize $S[l]$ with $i$
        Terminate the simulation and put control back to the beginning of the loop on $i$
      Initialize $S[l]$ with the next value in the sequence $\mathcal{P}$
      Provide the value $S[l]$ to the algorithm $A$
    Keep running the simulation until it stops naturally with an answer $j$
    Initialize $S[j]$ with $i$

**Algorithm 3** Encode Bit String

$\mathcal{P}$, $\mathcal{L}$, and $\mathcal{R}$ are sequences of bits, initially empty
$\mathcal{A}$ is a binary vector of length $n$
**for all** $i = 1, 2, \ldots, n$ **do**
  **if** the target chunk of $q_i$ has at most $k$ probed bits **then**
    { Simulation $i$ is called *present* }
    Set $\mathcal{A}[i]$ to 1
    Simulate $A$ on the query $q_i$
    Let $l_1, l_2, \ldots, l_z$ denote the locations of the bits that $A$ probed in order of its execution ($q \leq t$)
    Let $(2\lceil \lg n \rceil + 3)j + 1$ be the output of $A$ { the first position of the target chunk of $q_i$ }
    **for all** $f = 1, 2, \ldots, z$ **do**
      **if** location $l_f$ is not marked as discovered **then**
        **if** $l_f$ belongs to the target chunk **then**
          Terminate the loop on $f$
        Append $B[l_f]$ to the end of $\mathcal{P}$
        Append 1 to the end of $\mathcal{L}$
        Mark $l_f$ as probed {Same location is never recorded twice in $P$}
    Mark all the bits in the target chunk as discovered { Some of the bits in the target chunk might have already been marked as discovered, recall that in this case we call those bits *overlaps*, and there are at most $k$ such bits }
  **else**
    { Simulation $i$ is called *absent* }
    Set $\mathcal{A}[i]$ to 0

**Algorithm 4** Decode Bit String

Let $B$ be a bit vector of length $L = n(2\lg n + 3)$.
Initialize all the separator positions with corresponding separators
Using $\mathcal{R}'$ and $\mathcal{C}'$, initialize all the absent chunks in $B$
**for all** $i = 1, 2, \ldots, n$ **do**
  **if** $(i-1)$ does not occur as the content of an absent chunk **then**
    { $q_i$ is present }
    Simulate $A$ on the query $q_i$
    When $A$ needs to probe the bit at a location $l$ in $B$
    **if** $B[l]$ is initialized **then**
      Provide $B[l]$ to the algorithm $A$
    **else**
      Read the next bit $g$ from $\mathcal{L}'$
      **if** $g = 0$ **then**
        {This probe is made to an undiscovered location in the target chunk}
        Let $j$ be the chunk where $l$ is located
        Initialize the bits of the $j$-th chunk of $B$ with $\texttt{binary}_{\lceil \lg \mathtt{n} \rceil}(\mathtt{i} - \mathtt{1})$
        Terminate the simulation and put control back to the beginning of the loop on $i$
      Initialize $B[l]$ with the next bit in the sequence $\mathcal{P}'$
      Provide the value $B[l]$ to the algorithm $A$
    Keep running the simulation until it stops naturally with an answer $(2\lceil \lg n \rceil + 3)j + 1$
    Initialize the $j+1$-st chunk of $B$ with $\texttt{binary}_{\lceil \lg \mathtt{n} \rceil}(\mathtt{i} - \mathtt{1})$