Analysis and experimental evaluation of an approximation algorithm for the length of an optimal Lempel-Ziv parsing

Joonas Nietosvaara

Helsinki June 2019
Master’s thesis
UNIVERSITY OF HELSINKI
Department of Computer Science
We examine a previously known sublinear-time algorithm for approximating the length of a string’s optimal (i.e. shortest) Lempel-Ziv parsing (a.k.a. LZ77 factorization). This length is a measure of compressibility under the LZ77 compression algorithm, so the algorithm also estimates a string’s compressibility. The algorithm’s approximation approach is based on a connection between optimal Lempel-Ziv parsing length and the number of distinct substrings of different lengths in a string. Some aspects of the algorithm are described more explicitly than in earlier work, including the constraints on its input and how to distinguish between strings with short vs. long optimal parsings in sublinear time; several proofs (and pseudocode listings) are also more detailed than in earlier work. An implementation of the algorithm is provided.

We experimentally investigate the algorithm’s practical usefulness for estimating the compressibility of large collections of data. The algorithm is run on real-world data under a wide range of approximation parameter settings. The accuracy of the resulting estimates is evaluated. The estimates turn out to be consistently highly inaccurate, albeit always inside the stated probabilistic error bounds. We conclude that the algorithm is not promising as a practical tool for estimating compressibility. We also examine the empirical connection between optimal parsing length and the number of distinct substrings of different lengths. The latter turns out to be a surprisingly accurate predictor of the former within our test data, which suggests avenues for future work.

ACM Computing Classification System (CCS):
Information systems $\rightarrow$ Data compression
Theory of computation $\rightarrow$ Approximation algorithms analysis
Theory of computation $\rightarrow$ Streaming, sublinear and near linear time algorithms
Contents

1 Introduction 1

2 Notation 5

3 LZ77 compression and parsings 5
  3.1 Overview of LZ77 ................................. 5
  3.2 Relating optimal parsing length to LZ77-compressibility .......... 7

4 Approximating the optimal parsing length 8
  4.1 Classes of approximation algorithms .......................... 8
  4.2 Amplification of success probability ........................... 9
  4.3 Query complexity and sublinearity ............................. 10
  4.4 Relating optimal parsing length to distinct substring counts ... 11
  4.5 Estimating distinct substrings counts ........................ 12
  4.6 Estimating optimal parsing length ............................ 15
     4.6.1 An initial unoptimized algorithm ......................... 17
     4.6.2 An intermediate form .................................. 18
     4.6.3 The final algorithm ................................... 19
     4.6.4 Error bounds and complexity ............................ 23

5 Distinguishing strings with short optimal parsings from strings with long optimal parsings in sublinear time 28

6 Experimental investigation 32
  6.1 Choice of test files ........................................ 32
  6.2 Choice of test parameters ................................... 33
  6.3 Results ..................................................... 35
  6.4 Analysis of results ......................................... 38
  6.5 Sources of inaccuracy in estimating optimal parsing length ..... 38

7 Conclusions and future work 41

Appendices

1 A simple binary encoding for parsings
2 Proof of Lemma 4.3

3 Determining $A$ and $\epsilon$ from $l_0$ and $f_q$

4 Full results of test runs
1 Introduction

In this work we examine an algorithm, due to Raskhodnikova, Ron, Rubinfeld and Smith [RRRS13] which, given a string, produces an approximation of the length of an optimal Lempel-Ziv parsing of that string.

The Lempel-Ziv parsing is also known as the \textit{LZ77 parsing}, \textit{Lempel-Ziv factorization} and \textit{LZ77 factorization}. We use these terms interchangeably. The word “parsing” by itself also refers to the same thing. The formal definition of a Lempel-Ziv parsing is given in Section 3. For the purposes of this introduction, it suffices to note that a Lempel-Ziv parsing is a data structure that encodes a string as a sequence of phrases, and a parsing is \textit{optimal} if there is no shorter parsing that encodes the same string.

The LZ77 parsing serves as the core data structure of a compression algorithm also known as LZ77 [ZL77]; this is where it originates. Several mainstream compression utilities such as \textit{gzip} and \textit{7zip} are based on LZ77. LZ77 parsings are also relevant to the problem of indexing compressed collections of repetitive text [KKJ16]. They are also useful for detecting periodicities in strings [KKJ16].

The length of an optimal Lempel-Ziv parsing of a string, or its \textit{optimal parsing length}, is closely connected with, if not straightforwardly proportional to, the compressibility of that string under LZ77 and related compression algorithms. The shorter a string’s optimal LZ77 parsing is, the more compressible that string is. Thus the optimal parsing length is a measure of compressibility under these algorithms. The optimal parsing length is also a lower bound for the size of the smallest context-free grammar representing a string (while computing the actual size is an NP-hard problem) [CLD+05].

With the most efficient algorithms currently known, computing an optimal LZ77 parsing of a string requires time and memory linear in the length of the string [KKJ16]. More precisely, the fastest algorithms currently known use between $6n$ and $13n$ bytes of memory [KKJ16], where $n$ is the size of the input in bytes. The most space-efficient practical algorithm available is one with a tunable time-space tradeoff [KKP13], whose memory usage can go slightly below $2n$ at best [KKP14]. External-memory-based algorithms that use a combination of RAM and disk also exist; they can use less than $n$ bytes of RAM, compensating for this by using a large amount of working space on disk [KKP14]. Memory usage remains a major bottleneck when dealing with large input sizes, as we often are in the contexts of data compression and indexing.

\textit{Approximation algorithms}, in general, are algorithms that compute an answer to some computational problem which is inexact but has a bounded error, while requiring less resources than algorithms that compute an exact answer. Given how resource-intensive computing an optimal parsing is, the prospect of an approximation algorithm for the length of an optimal parsing is of some interest. The most obvious use case of a practical approximation algorithm for optimal parsing length would be quickly and cheaply estimating the compressibility of large files. Ideally, we could get an estimate of the compressibility of a file using much less computa-
tional resources than it would take to actually compress that file, and we could then make the decision of whether or not to compress based on that estimate.

As noted, the bulk of this text is dedicated to examining a certain algorithm due to Raskhodnikova et al. [RRRS13]. This algorithm, which we refer to by the name EstOPL throughout, is an approximation algorithm for the length of an optimal LZ77 parsing. (EstOPL is described in detail in Section 4; the explicit pseudocode is given as Algorithm 4.7 in Section 4.6.3.) In outline, the error bounds and resource usage of EstOPL are as follows: given a string $w$ of length $n$ and numbers $A > 1$ and $\epsilon > 0$, which also satisfy some additional constraints discussed later, EstOPL produces a number $X$ that, with probability at least $\frac{2}{3}$, satisfies

$$OPL(w)/A - \epsilon n \leq X \leq OPL(w) \cdot A + \epsilon n$$

where $OPL(w)$ stands for the length of an optimal parsing of $w$, a notation that is used throughout this text. The larger the error bounds $A$ and $\epsilon$ are made, the faster EstOPL runs. The time and space complexity of EstOPL is $\tilde{O}(n^{1-3x+y})$ when $A = n^x$ and $\epsilon = n^{-y}$ and $x$ and $y$ are any positive constants satisfying certain constraints\(^1\).

(1) The original article describing EstOPL [RRRS13] states that it has a time and space complexity of "$\tilde{O}(\frac{n}{\log n})$", an obviously reminiscent expression. However, we found it difficult to pin down a precise meaning for the multiple-variable asymptotic notation, so in this text we limit ourselves to the single-variable claim just outlined.)

Note that aiming for a success probability of $\geq \frac{2}{3}$ is merely an arbitrary convention. This is because, given an algorithm with any success probability $p_1 > \frac{1}{2}$, we can derive a new algorithm with a success probability of $p_2$, where $p_1 < p_2 < 1$, by repeating the original algorithm $\Theta\left(\log \frac{1}{1-p_2}\right)$ times and taking the median of the results. This is known as amplifying the success probability, and it is discussed in more detail in Section 4.2.

A sublinear time algorithm, or just sublinear algorithm, is a special kind of approximation algorithm that computes its output while looking at only a part of its input [RS11]\(^2\). For example, a sublinear time algorithm may compute an estimate of some numerical property of a graph while inspecting only a subset of the vertices of the graph [ORRR12]. A sublinear algorithm that computes an estimate of the number of unique elements in a list, by inspecting a random subset of the elements of that list, is used as a building block of EstOPL.

EstOPL itself can be used as a sublinear algorithm, that is, with the right settings for $A$ and $\epsilon$, it can produce its estimate of $OPL(w)$ while looking at only part of the string $w$. (And with the right settings for $x$ and $y$ above, the running time can

---

\(^1\) $\tilde{O}(g(n))$ means $O(g(n))$ except ignoring logarithmic factors. Formally, $f(n) \sim \tilde{O}(g(n))$ iff $f(n) \sim O(g(n) \log(g(n))^k)$ for some $k$.

\(^2\) Note that an algorithm is said to run in “linear time” if it runs in $O(n)$ time, and an obvious extension of that terminology would be to say that an algorithm runs in “sublinear time” if it runs in $o(n)$ time (for example, $O(n^\alpha)$ time for some $\alpha < 1$.)

This latter concept of sublinearity is indeed practically equivalent to the one discussed in the main text, since an algorithm that runs in $o(n)$ where $n$ is the input size will not, in general, access all of its input, under conventional models of computation.
Since all algorithms for computing the optimal parsing itself naturally look at their whole input, and also need at least $O(n)$ time and space, EstOPL can indeed produce estimates of OPL($w$) while using substantially less resources than any algorithm that computes OPL($w$) exactly.

The technique that enables us to estimate OPL($w$) in sublinear time is based on a combinatorial connection, discovered by Raskhodnikova et al. [RRRS13], between (1) the number of distinct substrings of different lengths in a string, and (2) the optimal parsing length of that string. Namely, if we know the numbers $d_1, d_2, \ldots, d_{l_0}$, where $d_l$ stands for the number of distinct substrings of length $l$ in the string and $l_0$ is some positive integer, then we can derive lower and upper bounds for the length of an optimal LZ77 parsing of the string. Proposition 4.1 states these bounds precisely. If we know imprecise estimates for $d_1, d_2, \ldots, d_{l_0}$, we can derive correspondingly wider bounds on the optimal parsing length.

Estimating the number of distinct elements in a collection is a well-studied problem, often referred to as the distinct estimation problem (DE). The connection between distinct substring counts and optimal parsing length mentioned above establishes a link between DE and the problem of estimating the optimal parsing length of a string. It enables us to use an algorithm for DE to obtain estimates of optimal parsing length, via estimating the number of distinct substrings of different lengths and then converting those estimates into bounds for the optimal parsing length.

Since sublinear algorithms for DE exist, it is then relatively straightforward to derive a sublinear algorithm for estimating the optimal parsing length. This is what Raskhodnikova et al. [RRRS13] do to derive EstOPL, though EstOPL also has some added optimizations. The sublinear algorithm for DE used is a simple one and also due to Raskhodnikova et al. [RRRS13]; it is given as Algorithm 4.2 in Section 4.5.

The question then is, what is the quality of the estimates produced by EstOPL? That is, how wide must the error bounds be made to attain satisfactorily low resource usage, and how close to the true answer are the estimates produced? Does EstOPL show promise as a practical algorithm for estimating the compressibility of large files? These questions are investigated in Section 6, where we test an implementation of EstOPL against various large files. The results obtained in Section 6 strongly suggest that EstOPL is not a practically useful algorithm. (The original article describing EstOPL [RRRS13] does not claim that practical usefulness should be expected.) The error bounds must be made very wide to reach usefully low resource usage, and the estimates produced are far off from the exact answer.

Our original contribution in this work consists primarily of the experimental results of Section 6, as well as the implementation of EstOPL with which the results were collected. As far as we know, no other implementation of the algorithm is publicly available; possibly none exists since the article describing EstOPL [RRRS13] has a theoretical focus. Our implementation is available at https://github.com/oneb/estcompr.

The secondary part of our original contribution consists of our explicit and precise
treatment of several issues that were, reasonably, left implicit or imprecise in the original article describing the EstOPL algorithm [RRRS13]. These include, in decreasing order of importance:

- The constraints that the input of EstOPL must satisfy for the algorithm to work are discussed and stated explicitly (as assert statements in pseudocode listings.) They are not discussed by Raskhodnikova et al. [RRRS13].

- The original article states that for any $\alpha > 0$, EstOPL can be used to “distinguish” between “strings compressible to $O(n^{1-\alpha})$” and “strings compressible to $\Omega(n)$”, “in time $O(n^{1-\alpha})$”. The precise meaning of this is slightly challenging to pin down, and only about two sentences are given to the notion. Meanwhile, the entirety of Section 5 in this text is dedicated to making this claim precise and proving it.

- Unlike Raskhodnikova et al. [RRRS13], we provide pseudocode of the “final form” of EstOPL (Algorithm 4.7 in Section 4.6.3). In the original article, only the pseudocode of a simplified, slower version is given while the modifications required to transform it into the proper algorithm are described in the accompanying text.

- The proofs of Lemma 4.3 (related to distinct estimation) and of Propositions 4.5 and 4.6 (which give the output bounds and complexity of EstOPL) are more explicit and detailed than the corresponding proofs in the original article describing EstOPL [RRRS13]. Also, as part of Proposition 4.6, we prove that despite the constraints on the input mentioned above, EstOPL is able to run for a nontrivial range of inputs.

- In Section 3.2 we briefly consider the exact bounds that we can put on the size, in bits, of the LZ77-compressed version of a string, given knowledge of the optimal parsing length of that string. In the original article [RRRS13], it is stated that the size in bits of the LZ77-compressed version of a string $w$ is at most $2\log n \cdot OPL(w)$, where $n$ is the length of $w$. We derive a slightly higher upper bound, with extra terms that were likely ignored in that article. The difference is practically insignificant.

The rest of this text is organized as follows:

- Section 3 briefly discusses LZ77 compression and introduces the concepts of an LZ77 parsing and an optimal parsing.

- Section 4 describes the building blocks of the algorithm EstOPL and the algorithm itself in detail. The error bounds and asymptotic resource usage of EstOPL are stated and proved.

- Section 5, as noted above, describes a way in which EstOPL can distinguish highly compressible strings from incompressible strings in sublinear time.
Section 6, as noted above, describes experimental results obtained by applying ESTOPL to real-world data. We also consider the sources of the inaccuracy in the results and observe that they will apply to any attempt to estimate optimal parsing length from the kind of data used by ESTOPL. We also briefly speculate about an alternative approach for estimating the optimal parsing length that would require very little memory (but substantial CPU time).

2 Notation

It is useful for us to distinguish between abstract mathematical functions, and the algorithms that compute them. Identifiers written in SMALLCAPSFONT stand for algorithms. Identifiers written in SansSerifFont stand for mathematical functions. (One-letter identifiers like $f$ can also stand for mathematical functions.) As noted, an algorithm may compute a mathematical function. Thus, for example, in Section 4.2, AMPCOUNT is an algorithm that computes AmpCount.

In pseudocode, we use “$a \leftarrow (...)$” to set the value of a variable named $a$, and “$a = (...)$” to define an immutable constant named $a$.

If $T = t_1t_2..t_n$ is a string of length $n$, then $T[a..b]$, where $1 \leq a \leq b \leq n$, denotes the substring $t_a, t_{a+1}..t_b$ of $T$ (which is of length $b - a + 1$). “$\epsilon$” stands for the empty string of length 0.

In all subsequent sections, “$\log x$” stands for the natural logarithm of $x$ (that is, the logarithm base $e$). (This makes a difference in some places.)

3 LZ77 compression and parsings

In this section we introduce and define the LZ77 parsing and outline its use in the LZ77 compression algorithm. In Section 3.2, we briefly examine the relation between LZ77 compressibility and optimal parsing length, since, as noted in the introduction, estimating compressibility via estimating the optimal parsing length is a potential application for an algorithm like the one we will be examining in this text.

3.1 Overview of LZ77

Several slightly different compression schemes have gone under the name of LZ77, involving slightly different definitions of a parsing. The exact type of parsing we are concerned with in this text is as follows:

Definitions. A parsing over an alphabet $\Sigma$ is a sequence $p = p_1p_2..p_m$ of phrases where each phrase $p_i$ is either a symbol from the alphabet $\Sigma$ or a pair of positive integers $(k, l)$.
A parsing $p$ is *valid* if $\text{PARSINGToString}$ (Algorithm 3.1 below) runs successfully when given $p$ as an input. It is *invalid* otherwise.

A valid parsing $p$ encodes a string $w$ if $\text{PARSINGToString}$ outputs $w$ when given $p$ as an input. When $p$ encodes $w$, we can also say that $w$ has the parsing $p$. A string can have several parsings.

A parsing that encodes a string $w$ is *optimal* if there is no shorter parsing, by number of phrases, that also encodes $w$. A string can have several optimal parsings.

When performing LZ77 compression, we are given a string $w$ over an alphabet $\Sigma$. We compute a valid parsing over $\Sigma$ that encodes $w$. Precisely how the parsing may be computed from the string is discussed later in this section.

When decompressing, we are given a parsing $p = p_1 p_2 \ldots p_m$. We compute a string $w$ from the parsing using $\text{PARSINGToString}$ (Algorithm 3.1), which runs in $O(|w|)$ time.

In a full implementation of LZ77 compression, there are naturally additional steps. When compressing, after computing the parsing, the parsing may be further processed, and then the result is encoded in binary to be stored as a file. When decompressing, the binary is decoded, any other processing needed to recover the parsing is done, and only then the parsing is converted into the original string.

For the purposes of this text, the only part of LZ77 compression we are concerned with is the transformation from string to parsing and vice versa, except for Section 3.2 where we briefly examine the relationship between optimal parsing length and LZ77-compressibility.

A naive algorithm that, given a string $w$ of length $n$, computes a valid parsing $p$ that encodes $w$ (as done when compressing) is as follows. Start with $p$ equal to an empty parsing and $j = 1$. At each step, search for the longest substring $w[k..k+l]$ that starts at an index $k < j$ and is equal to $w[j..j+l]$. If such a substring is found, add the pair $(k, l)$ to the parsing $p$ and set $j = j + l$. If no such substring exists, add the symbol $w[j]$ to $p$ and set $j = j + 1$. Continue until $j = n + 1$. Output $p$.

The above algorithm runs in $O(n^2)$ time and the parsing that it computes is optimal. As discussed in the introduction, more advanced algorithms exist that compute an optimal parsing in $O(n)$ time and space [KKJ16].

As already noted in the introduction, given a string $w$ of length $n$ we are interested in finding the *length of an optimal parsing* of $w$. We denote this length by $\text{OPL}(w)$. As noted above, $\text{OPL}(w)$ can be found in $O(n)$ time, that is, in linear time. From Section 4 onwards, we will be examining an algorithm that can find an approximation of $\text{OPL}(w)$ in sublinear time.
Algorithm 3.1 Computing the string encoded by a parsing. Done as part of LZ77 decompression.

function ParsingToString(p)
    \( m = \text{Length}(p) \)
    \( w \leftarrow "\n" \)
    for \( i \leftarrow 1 \) to \( m \) do
        if \( p_i \) is a symbol \( \alpha \) then
            Append \( \alpha \) to \( w \).
        if \( p_i \) is a pair \((k,l)\) and \( k \leq |w| \) then
            for \( j \leftarrow 0 \) to \( l - 1 \) do
                Append \( w[k+j] \) to \( w \).
        if \( p_i \) is a pair \((k,l)\) and \( k > |w| \) then
            Abort. \( \triangleright \) The parsing is invalid.
    return \( w \)

3.2 Relating optimal parsing length to LZ77-compressibility

An approximation of the size of the LZ77-compressed version of \( w \) can be derived from \( \text{OPL}(w) \) (and as noted, an approximation of \( \text{OPL}(w) \) can be produced with the algorithm EstOPL). The precise relation of \( \text{OPL}(w) \) to the size of the compressed version depends on specifics of the implementation. A naive binary encoding of the parsing \( p \) requires

\[
A_{\Sigma} + 2[^{\log_2[\log_2 n]}] - 1 + |p| ([\log_2 n] + [\log_2 \max(n, |\Sigma|)])
\]

bits, where \( n \) is the length of \( w \), and \( A_{\Sigma} \) is the number of bits required to indicate, in some standard format, that the alphabet being used is \( \Sigma \). (See Appendix 1 for a description of the encoding.)

If \( |\Sigma| \leq n \), which is usually the case, this is equal to

\[
A_{\Sigma} + 2[^{\log_2[\log_2 n]}] - 1 + |p|2[^{\log_2 n}].
\]

All reasonable implementations of LZ77 compression will yield a compressed file that is no larger than the binary encoding of the optimal parsing under the above naive encoding. So if we know the length of an optimal parsing of \( w \), we can upper-bound the size of the compressed version of \( w \): it is at most \( A_{\Sigma} + 2[^{\log_2[\log_2 n]}] - 1 + |p|([\log_2 n] + [\log_2 \max(n, |\Sigma|)]) \) bits. Obtaining a meaningfully tight lower bound would be much more complicated; we do not attempt it. This means that, from an upper bound for the size of an optimal parsing we can derive an upper bound for the size of the compressed file, whereas from a lower bound for the size of an optimal parsing, we cannot, in any trivial way, derive information about the size of the compressed file. EstOPL provides both a lower bound and an upper bound for the length of an optimal parsing of \( w \), but when it comes to estimating the size of the compressed version of \( w \), only the upper bound is informative to us.
Outside of the current section, we focus on estimating \( OPL(w) \), and ignore the question of the size of the LZ77-compressed version of \( w \).

4 Approximating the optimal parsing length

In this section we describe the algorithm EstOPL and its building blocks. Sections 4.1 to 4.3 discuss basic prerequisite concepts. Sections 4.4 and 4.5 discuss how the distinct estimation problem relates to the problem of estimating the optimal parsing length, and describe the algorithm for distinct estimation that is adapted for use as part of EstOPL. In Section 4.6 we describe EstOPL in stages; Sections 4.6.1 and 4.6.2 discuss simplified slow versions of EstOPL as a way of leading up to the actual EstOPL algorithm in Section 4.6.3. Finally, in Section 4.6.4 we state and prove EstOPL’s error bounds and its time, space and query complexity.

4.1 Classes of approximation algorithms

The following definitions will be useful to characterize the kinds of approximation performed by the algorithms that we investigate.

Definitions.

- Given \( A > 1 \), a quantity \( x \) is an \( A \)-approximation for a quantity \( X \) if \( X/A \leq x \leq XA \).

- An algorithm \( G \) is an \( A \)-approximation algorithm for a function \( f \) if, given \( A > 1 \) and an input \( w \), running \( G(A,w) \) produces an \( A \)-approximation of \( f(w) \) with probability \( \geq \frac{2}{3} \).

- Given \( A > 1 \) and \( Z > 0 \), a quantity \( x \) is an \((A,Z)\)-approximation for a quantity \( X \) if \( X/A - Z \leq x \leq XA + Z \).

- An algorithm \( G \) is an \((A,\epsilon)\)-approximation algorithm for a function \( f \) if, given \( A > 1 \) and \( \epsilon > 0 \), and an input \( w \) of size \( n \), running \( G(w,A,\epsilon) \) produces an \((A,\epsilon n)\)-approximation of \( f(w) \) with probability \( \geq \frac{2}{3} \).

- An algorithm is a partial \( A \)-approximation algorithm or a partial \((A,\epsilon)\)-approximation algorithm if it only works for some subset of inputs and aborts otherwise. If the algorithm does not abort on a given combination of inputs, we say that it runs successfully with that combination of inputs. We also say that combination of inputs is runnable.

Under this terminology, the algorithm that we are examining, namely EstOPL in Section 4.6.3, is a partial \((A,\epsilon)\)-approximation algorithm for \( OPL(w) \), the length of an optimal parsing of a string \( w \). A partial \( A \)-approximation algorithm for the
number of distinct elements in a list, namely SimpleDistEst in Section 4.5, is used as a building block of EstOPL.

Obviously, for the existence of a partial A- or (A, ϵ)-approximation algorithm to be of any significance, the space of runnable inputs must be large enough in some sense. This is the case for the algorithms examined in this text. For example, the runnable inputs of SimpleDistEst in Section 4.5 are those for which $1 \leq \frac{10m}{A^2} \leq n$, where $n$ is the length of the input list.

4.2 Amplification of success probability

In the main algorithm we will use a subroutine that produces a number in a desired range with a medium probability like $\frac{3}{4}$, and by running it multiple times and combining the results, we will obtain a number that is in that range with a high probability. The more times we repeat the subroutine, the higher this probability. This technique is called amplification of success probability. It works as follows:

We are given an algorithm that produces a value in the range $[x_1, x_2]$ with probability $p_1 > \frac{1}{2}$. For any $p_2 > p_1$, we can obtain a value that is in $[x_1, x_2]$ with probability $\geq p_2$ by running the algorithm $\text{AmpCount}(p_1, p_2)$ times and taking the median of the results, where $\text{AmpCount}$ is defined as follows:

If less than half the values are outside the range, the median of those values is necessarily inside the range. With $k$ repetitions, the probability that less than half the values are outside the range, that is, that at most $\lfloor (k-1)/2 \rfloor$ values are outside the range, is

$$\text{AmpResult}(p_1, k) = \sum_{i=0}^{\lfloor (k-1)/2 \rfloor} \binom{k}{i} p_1^k (1-p_1)^i.$$ 

There is no closed formula for this quantity, but it can be computed. $\text{AmpResult}$ in Algorithm 4.1 does this. Now the number of repetitions needed to reach probability $p_2$ from $p_1$ is:

$$\text{AmpCount}(p_1, p_2) = \min \{k \in \mathbb{N} \mid \text{AmpResult}(p_1, k) \geq p_2 \}.$$ 

With $p_1$ constant, $\text{AmpResult}(p_1, k)$ is an increasing function of $k$. So $\text{AmpCount}$ can also be computed, by simply trying increasing values of $k$ in order until one is found that produces a high enough result, as shown in Algorithm 4.1.

Despite the lack of a closed formula, we know that $\text{AmpCount}(p_1, p_2)$ grows with $p_2$ like $\Theta \left( \log \frac{1}{1-p_2} \right)$ given a constant $p_1 > \frac{1}{2}$ [RRRS13]. Table 1 gives some illustrative values of $\text{AmpCount}$.

The time complexity of $\text{AmpCount}$ is small enough that it does not contribute to the time complexity of the main algorithm. It is as follows. Computing $\binom{k}{i} = \frac{k!}{(k-i)!i!}$ takes $O(k)$ multiplication operations, for $O(k)$ time. In a call to
Algorithm 4.1 An algorithm for computing $AmpCount$.

function $AmpCount(p_1, p_2)$
  
  $k \leftarrow 1$
  $p \leftarrow p_1$
  
  while $p < p_2$ do
    $k \leftarrow k + 1$
    $p \leftarrow AmpResult(p_1, k)$
  
  return $k$

function $AmpResult(p_1, k)$

  $r \leftarrow 0$

  for $i \leftarrow 0$ to $\lfloor (k - 1)/2 \rfloor$ do
    $r \leftarrow r + \binom{k}{i} p_1^{k-i}(1-p_1)^i$

  return $r$

<table>
<thead>
<tr>
<th>$p_2$</th>
<th>$AmpCount\left(\frac{3}{7}, p_2\right)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 - 1/10$</td>
<td>10</td>
</tr>
<tr>
<td>$1 - 1/100$</td>
<td>20</td>
</tr>
<tr>
<td>$1 - 1/1000$</td>
<td>30</td>
</tr>
<tr>
<td>$1 - 1/10^6$</td>
<td>76</td>
</tr>
<tr>
<td>$1 - 1/10^9$</td>
<td>122</td>
</tr>
</tbody>
</table>

Table 1: Repetitions needed to amplify a success probability of $\frac{3}{7}$.

$AmpResult(p_1, k)$, $\binom{k}{i}$ is computed $\lfloor (k - 1)/2 \rfloor = O(k)$ times for various $i$, for $O(k^2)$ time$^3$. Given $p_1$, a call to $AmpCount(p_1, p_2)$ calls $AmpResult$ once for each number in $\{2, 3, \ldots, AmpCount(p_1, p_2)\}$; so $AmpResult$ is called $O\left(\log \frac{1}{1-p_2}\right)$ times with arguments of size $O\left(\log \frac{1}{1-p_2}\right)$, giving a running time of

$$O\left(\log \frac{1}{1-p_2}\right) \cdot O\left(\left(\log \frac{1}{1-p_2}\right)^2\right) = O\left(\left(\log \frac{1}{1-p_2}\right)^3\right).$$

4.3 Query complexity and sublinearity

Given an algorithm $G$ that takes as input a string $w$ of length $n$, we are interested in three different performance measures of $G$: time usage, space usage and number

$^3$The time complexity could be reduced further by exploiting the fact that computing $\binom{k}{i}$ takes $O(1)$ time if $\binom{k}{i-1}$ has already been computed, and also the fact that computing the $i$th power of a number requires only $O(\log i)$ multiplications. But the “naive” time complexity that we calculate in the main text, which ignores these optimizations, is already low enough that this would make no difference for our purposes.
of queries made. Time and space usage have their usual meaning and are examined in a standard way. We can also ask about the total number of times during its execution that $G$ queries (i.e., looks at, accesses) a character of $w$. For example, given a string of length 1000, $G$ could query only 500 locations in the string and compute its result from the results of those queries.

Similar to time and space complexity, we have query complexity, determined by the asymptotic growth of the number of queries an algorithm makes as a function of the input size. So, an algorithm could have a time complexity of $O(n^2)$, a space complexity of $O(n)$ and a query complexity of $O(\sqrt{n})$.

We classify an order of growth as sublinear if it is $o(n)$. So, for example, $O(\sqrt{n})$ and $O\left(n^{\frac{3}{2}} + \log n\right)$ are sublinear and $O(n)$ and $O(n^2)$ are not. For large enough inputs, any algorithm with a sublinear query complexity queries less than the entire input string.

The algorithm that we are examining is a partial $(A, \epsilon)$-approximation algorithm for $OPL(w)$. When $A = A(n)$ and $\epsilon = \epsilon(n)$ are set as functions of $n$ in the right way, the time, space and query complexity becomes sublinear with respect to $n$, as stated in Proposition 4.6.

### 4.4 Relating optimal parsing length to distinct substring counts

Given a string $w$ of length $n$, we will be interested in the number of distinct length-$l$ substrings of $w$, for various values of $l$. We denote this number by $d_l(w)$, or just $d_l$ when $w$ is clear from context. $d_l(w)$ is at least 1 and at most $n - l + 1$.

Exploiting combinatorial connections between the number of distinct length-$l$ substrings and the length of an optimal parsing, Rashkodnikova et al. [RRRS13] prove the following fact relating $d_l(w)$ to $OPL(w)$:

**Proposition 4.1.** Let $w$ be a string of length $n$. Let $l_0$ be a positive integer less than $n$. Denote $m = \max_{l=1}^{l_0} \frac{d_l(w)}{l}$. Now the following holds:

$$m \leq OPL(w) \leq 4 \left( m \log l_0 + \frac{n}{l_0} \right).$$

In ESTOPL, this fact will be used to obtain an estimate of $OPL(w)$ as follows: First choose an $l_0$. Then compute estimates for all of $d_1, d_2, \ldots, d_{l_0}$ using a version of the SIMPLEDISTEST algorithm described in the next section. From these estimates compute an estimate for $m$. From this estimate and Proposition 4.1, obtain bounds for $OPL(w)$.

(Table 5 in Section 6.5 lists the specific upper and lower bounds that Proposition 4.1 gives for the optimal parsing lengths of some example files.)
4.5 Estimating distinct substrings counts

The distinct estimation problem (DE) is the problem of estimating the number of unique (i.e. distinct) elements in a list. It is trivial to adapt an algorithm for DE to the problem of estimating the number of distinct substrings of length $l$ in a string $w$: Treat the string as a list of $n - l + 1$ elements, and whenever the algorithm for DE would read the element at index $i$ from the list, read the $l$ characters $w[i..i + l - 1]$ from $w$.

Algorithm 4.2 The algorithm SIMPLEDISTEst – a partial $A$-approximation algorithm for the number of distinct elements in $Q$.

```plaintext
function SIMPLEDISTEst(Q, A)
    assert A > 1
    n = |Q|
    s = $\frac{10n}{A}$
    assert $1 \leq s \leq n$
    h ← EMPTYSET()
    do ⌈$s$⌉ times
        i ← RANDOMBETWEEN(1, n)
        ADD(h, Q[i])
    end
    $\hat{C} = \text{SIZE}(h)$
    return $\hat{C} \cdot A$
```

Algorithm 4.2, SIMPLEDISTEst, is the algorithm for DE whose adaptation for substrings we will use in the main algorithm. As input, it takes a list $Q$ and an approximation factor $A$. In outline, it works as follows:

1. Sample $\lceil \frac{10}{A} \cdot |Q| \rceil$ symbols from the list $Q$.
2. Count the exact number of distinct values in the sample, call it $\hat{C}$.
3. Return $\hat{C} \cdot A$.

In the pseudocode, RANDOMBETWEEN just denotes a subroutine that generates random integers; RANDOMBETWEEN($m, n$) returns a random integer between $m$ and $n$ inclusive.

Proposition 4.2 below states that this algorithm yields an $A$-approximation of the number of distinct elements in $Q$ with probability $\geq \frac{3}{4}$.

The assert statements in the pseudocode are included to make explicit the range of inputs within which it makes sense to use the algorithm, and within which we are able to prove desirable properties for the algorithm. Most pseudocode listings in this text will include such statements.
We require that \( s \leq n \) to comply with the conditions of Lemma 4.3. And we require \( 1 \leq s \) for the convenience of not having to consider the uninteresting degenerate situation where \( A \) grows arbitrarily large while the number of elements sampled does not shrink below 1.

It will be useful for later to be more explicit about how the elements of the sample are processed.

To count the exact number of distinct values in the sample, we can use a “set” data structure. For our purposes, however, it does not actually matter what we use. The final form of the algorithm uses a trie, as shown later; the simpler set data structure is only used in illustrative preliminary forms of the algorithm. So we will not dwell on the properties of the set data structure, except to note that it has these operations:

- \textsc{EmptySet}(). Return a new set with no elements.
- \textsc{Add}(S, x). Add element \( x \) into the set \( S \).
- \textsc{Size}(S). Return the number of (distinct) elements in the set \( S \).

The final algorithm does use the strategy of sampling \( \lceil \frac{10n}{A^2} \rceil \) out of \( n \) elements, and then using \( A \) times the number of distinct elements in the sample as an estimate of the number of distinct elements in the entire collection. This is why we examine \textsc{SimpleDistEst} at all.

Denote the number of distinct elements in a list \( Q \) by \textsc{Distinct}(\( Q \)). We now prove that the output \textsc{SimpleDistEst}(\( Q, A \)) is indeed within a factor of \( A \) of \textsc{Distinct}(\( Q \)), with probability \( \geq \frac{3}{4} \). (The proof is given in two parts: Proposition 4.2 and Lemma 4.3. The proof of the Lemma is in an appendix due to its length.)

**Proposition 4.2.** \textsc{SimpleDistEst}(\( Q, A \)) outputs an \( A \)-approximation of \textsc{Distinct}(\( Q \)) with probability \( \geq \frac{3}{4} \).

**Proof.** We need to prove the following: Given a list \( Q = q_1, q_2, \ldots, q_n \) of length \( n \) with \( c \) distinct elements, and a random sample, with replacement, of \( \lceil \frac{10n}{A^2} \rceil \) elements from \( Q \), then, if \( \hat{C} \) is the number of distinct elements in the sample,

\[
\frac{c}{A} \leq \frac{\hat{C}}{A} \leq c \cdot A
\]

is true with probability \( \geq \frac{3}{4} \). That is, we need

\[
\frac{c}{A} \leq \frac{\hat{C}}{A} \quad \text{(2)}
\]

and

\[
\frac{\hat{C}}{A} \leq c \cdot A \quad \text{(3)}
\]

Also, we are assured that \( 1 \leq \lceil \frac{10n}{A^2} \rceil \leq n \).
(3) is trivially true: it is equivalent to \( \hat{C} \leq c \), that is, the claim that the number of distinct elements in the sample is at most the number of distinct elements in the whole list \( Q \), which is always the case.

For (2), note that it is equivalent to \( \hat{C} \geq c/A^2 \). Applying Lemma 4.3 with \( s = \lceil \frac{10n}{A^2} \rceil \) proves that with probability \( \geq \frac{3}{4} \),

\[
\hat{C} \geq \frac{1}{10} \cdot \frac{c}{n} \cdot \lceil \frac{10n}{A^2} \rceil \geq \frac{1}{10} \cdot \frac{c}{n} \cdot \frac{10n}{A^2} = \frac{10cn}{10nA^2} = \frac{c}{A^2}
\]

as desired.

\[ \square \]

**Lemma 4.3.** Given a list \( Q = q_1, q_2, \ldots, q_n \) of length \( n \) with \( c \) distinct elements \( e_1, e_2, \ldots, e_c \), sampling \( s \in \{1, n\} \) elements from \( Q \) with replacement yields at least \( \frac{1}{10} \cdot \frac{c}{n} s \) distinct elements with probability \( \geq \frac{3}{4} \).

**Proof.** In Appendix 2. \[ \square \]

**Algorithm 4.3** The algorithm \( \text{SSDistEst} \) – \( \text{SIMPLEDistEst} \) adapted for substrings of \( w \) of length \( l \).

```plaintext
function \( \text{SSDistEst}(w, l, A) \)
    assert \( A > 1 \)
    \( n = |w| \)
    assert \( 1 \leq l \leq n \)
    \( N = n - l + 1 \)
    \( s = \frac{10n}{A^2} \)
    assert \( 1 \leq s \leq N \)

    \( h \leftarrow \text{EMPTYSET}() \)
    \( \text{do } [s] \text{ times} \)
        \( i \leftarrow \text{RANDOMBETWEEN}(1, N) \)
        \( \text{ADD}(h, Q[i..i + l - 1]) \)
        \( \hat{C} = \text{SIZE}(h) \)
    \( \text{return } \hat{C} \cdot A \)
```

Algorithm 4.3, \( \text{SSDistEst} \), is the adaptation of \( \text{SIMPLEDistEst} \) for estimating the number of distinct substrings of length \( l \) of a string \( w \). The number it outputs is an \( A \)-approximation of \( d_l(w) \) with probability \( \geq \frac{3}{4} \).

**Corollary 4.4.** \( \text{SSDistEst}(w, l, A) \) outputs an \( A \)-approximation of \( d_l(w) \) with probability \( \geq \frac{3}{4} \).

We require that \( 1 \leq l \leq n \) since \( l \) is a substring length. The number of substrings of length \( l \) in a string of length \( n \) is \( n - l + 1 \), so we are sampling from a “list” of
Algorithm 4.4 The algorithm \textsc{Estimate} – \textsc{SimpleDistEst} with amplified success probability.

\begin{verbatim}
function \textsc{Estimate}(w, l, A, \delta)
    assert 0 < \delta < 1
    assert A > 1
    (n = |w|)
    (N = n - l + 1)
    (s = \frac{10N}{A^2})
    assert 1 \leq l \leq n
    assert 1 \leq s \leq N

    r \leftarrow \text{AmpCount}(\frac{3}{4}, \delta)
    \text{for } i \leftarrow 1 \text{ to } r \text{ do}
        e_i \leftarrow \textsc{SDDistEst}(w, l, A)
    \text{return the median of } e_1, e_2, \ldots, e_r
\end{verbatim}

$N = n - l + 1$ substrings. Thus we require $1 \leq s \leq N$ for the same reason we required $1 \leq s \leq n$ in \textsc{SimpleDistEst}.

The last subalgorithm we need is Algorithm 4.4, \textsc{Estimate}. It is simply a version of \textsc{SDDistEst} with the success probability amplified. \textsc{Estimate} takes as input a string $w$ of length $n$, a substring length $l \leq n$, an approximation factor $A$, and a desired success probability $\delta$. It returns a number that with probability $\geq \delta$ is an $A$-approximation of $d_l(w)$. That is, $d_l(w)/A \leq \text{\textsc{Estimate}}(w, l, A, \delta) \leq d_l(w)A$ holds with probability $\geq \delta$.

In \textsc{Estimate}, we assert $0 < \delta < 1$ because $\delta$ is a probability, and for clarity we also explicitly re-assert the requirements of \textsc{SDDistEst}. For clarity, the definitions of variables that are only used in assertions are in parentheses; this convention is also followed in the next section.

In the final form of the algorithm, instead of being cleanly separated subroutines, both \textsc{SDDistEst} and \textsc{Estimate} are split into pieces and interleaved into the rest of the algorithm in several places.

4.6 Estimating optimal parsing length

In this section we present three pseudocode listings:

- \textsc{EstOPLSlow1} (Algorithm 4.5 in Section 4.6.1)
- \textsc{EstOPLSlow2} (Algorithm 4.6 in Section 4.6.2)
- \textsc{EstOPL} (Algorithm 4.7 in Section 4.6.3.)

The final one, \textsc{EstOPL}, defines the algorithm that we are primarily interested
in. Due to Raskhodnikova et al. [RRRS13], it is a partial \((A, \epsilon)-approximation algorithm for \text{OPL}(w)\), the length of an optimal parsing of a string \(w\), as shown in Proposition 4.5.

In the original article describing EstOPL [RRRS13], it is stated that EstOPL has a time, space and query complexity of \(\tilde{O}\left(\frac{A}{\epsilon^3}\right)\). In this text, we prove a claim that is similar but narrower and easier to interpret, namely that when \(A = n^x\) and \(\epsilon = n^{-y}\), it runs in \(\tilde{O}(n^{1-3x+y})\) time, space and queries.\(^4\) This is shown in Proposition 4.6.

EstOPLSlow1 and EstOPLSlow2 are illustrative preliminary forms of the algorithm. They have unoptimized resource usage but their output satisfies the same guarantees as EstOPL (that is, they are also partial \((A, \epsilon)-approximation algorithms for \text{OPL}(w)\)). EstOPLSlow1 is roughly the same pseudocode as presented in the original article describing EstOPL [RRRS13]; it is the shortest and easiest to understand. EstOPLSlow2 is an intermediate form between EstOPLSlow1 and EstOPL. It does the same thing as EstOPLSlow1 but the code is rearranged so that the final modifications needed to turn it into EstOPL are small and as easy to understand as possible.

This section describes the algorithm in more explicit detail than is done in the original article describing EstOPL [RRRS13]. In that article, the authors present the pseudocode of EstOPLSlow1 and outline the modifications required to obtain the final algorithm EstOPL; pseudocode of EstOPL is not provided. This text, on the other hand, goes over the modifications in detail and does provide explicit pseudocode of EstOPL.

Like the algorithms examined in earlier sections, in order to work properly, these algorithms require that their input \((w, A, \epsilon)\) satisfy certain properties (beyond just \(A > 1\) and \(0 < \epsilon < 1\)). (So in the terminology of Section 4.1, they are partial \((A, \epsilon)-approximation algorithms.). The assert statements in the pseudocode state these requirements. They are all inherited from Estimate, except that for the substrings length \(l_0\) we require \(2 \leq l_0\) instead of \(1 \leq l_0\) so that the denominator of \(B\) does not become 0. It may seem that \("B > 1"\) needs to be asserted separately, but in fact, it is implied by \(s \leq N\).

Proposition 4.6 shows that for a nontrivial range of inputs, EstOPL runs successfully, that is, none of the assertions fail.

Finally, an apparent difference from Raskhodnikova et al. [RRRS13] is that we set \(B = \frac{A}{2\sqrt{\log[\frac{1}{\epsilon}]}}\), while in the article by Raskhodnikova et al. [RRRS13], it is set to \(B = \frac{A}{2\sqrt{\log \frac{1}{\epsilon}}} - \text{note the absence of the ceiling function. We believe this is likely a misprint in the article.}\(^5\)

\(^4\)As noted earlier, \(\tilde{O}(g(n))\) means \(O(g(n))\) except ignoring logarithmic factors; \(f(n) \sim \tilde{O}(f(n))\) iff \(f(n) \sim O(g(n)(\log g(n))^k)\) for some \(k\).

\(^5\)The proof of Proposition 4.5 fails if the ceiling function is absent.
4.6.1 An initial unoptimized algorithm

**Algorithm 4.5** The algorithm EstOPLSlow1, with pseudocode identical to that provided in the original article describing EstOPL [RRRS13].

```plaintext
1: function EstOPLSlow1(w, A, ϵ)
2:     assert A > 1
3:     assert 0 < ϵ < 1
4:     n = |w|
5:     \( l_0 = \lceil \frac{2}{A\epsilon} \rceil \)
6:     assert \( 2 \leq l_0 \leq n \)
7:     \( N = n - l_0 + 1 \)
8:     \( B = \frac{A}{2\sqrt{\log l_0}} \)
9:     \( s = \frac{10N}{B^2} \)
10:    assert \( 1 \leq s \leq N \)
11:   
12:   for \( l \leftarrow 1 \) to \( l_0 \) do
13:       \( \hat{d}_l \leftarrow \text{Estimate}(w, l, B, 1 - \frac{1}{3l_0}) \)
14:   
15:   \( \hat{m} = \max_{l=1}^{l_0} \frac{\hat{d}_{l}}{l} \)
16:   return \( \hat{m} \cdot \frac{A}{B} + \epsilon n \)
```

We start by discussing EstOPLSlow1. As input, the algorithm takes a string \( w \) and approximation parameters \( A \) and \( \epsilon \). It either aborts or returns a number that is an \((A, \epsilon, n)\)-approximation of OPL\((w)\) with probability \( \geq \frac{2}{3} \), where \( n \) is the length of \( w \). Internally, it works as follows.

Internal parameters \( l_0 \) and \( B \) are set as functions of \( A \) and \( \epsilon \). Estimate is called \( l_0 \) times as a subroutine to obtain, for each \( l \in \{1..l_0\} \), a number \( \hat{d}_l \) that is a \( B \)-estimate of \( d_l(w) \) with probability at least \( 1 - \frac{1}{3l_0} \) (recall that \( d_l(w) \), or \( d_l \) when \( w \) is clear from context, stands for the number of distinct substrings of length \( l \) in the string \( w \)).

A number \( \hat{m} \) is computed from the \( \hat{d}_l \), using a formula identical to the formula for \( m \) in Proposition 4.1. If all the \( \hat{d}_l \) are indeed \( B \)-estimates of \( d_l \), which turns out to be true with probability \( \geq \frac{2}{3} \), then \( \hat{m} \) is a \( B \)-estimate of \( m \) in Proposition 4.1.

The algorithm has now computed a number \( \hat{m} \) that, with probability \( \geq \frac{2}{3} \), is a \( B \)-estimate of another number \( m \), which the inequalities of Proposition 4.1 relate to OPL\((w)\). So now we can also derive an inequality relating \( \hat{m} \) and \( B \) to OPL\((w)\), that holds with probability \( \geq \frac{2}{3} \). This inequality turns out to imply that the number \( \hat{m} \cdot \frac{A}{B} + \epsilon n \) is necessarily in the range

\[
[\text{OPL}(w)/A - \epsilon n, \ \text{OPL}(w)A + \epsilon n],
\]

that is, that it is an \((A, \epsilon n)\)-approximation of OPL\((w)\) (this is shown in detail in the proof of Proposition 4.5.) Thus, we output \( \hat{m} \cdot \frac{A}{B} + \epsilon n \).
18

4.6.2 An intermediate form

Algorithm 4.6 The algorithm ESTOPLSLOW2. Rearranged version of
ESTOPLSLOW1 that does essentially the same thing.

1: function ESTOPLSLOW2\((w, A, \epsilon)\)
2: \textbf{assert} \(A > 1\)
3: \textbf{assert} \(0 < \epsilon < 1\)
4: \(n = |w|\)
5: \(l_0 = \lceil \frac{A}{3x} \rceil\)
6: \textbf{assert} \(2 \leq l_0 \leq n\)
7: \(N = n - l_0 + 1\)
8: \(B = \frac{A}{2\sqrt{\log n}}\)
9: \(s = \frac{10N}{B^2}\)
10: \textbf{assert} \(1 \leq s \leq N\)
11: \(r = \text{AMPCOUNT}\left(\frac{3}{4}, 1 - \frac{1}{3l_0}\right)\)
12: \begin{algorithmic}
13: \end{algorithmic}
14: \begin{algorithmic}
15: \textbf{for} \(l \leftarrow 1\) to \(l_0\) \textbf{do}
16: \begin{algorithmic}
17: \textbf{for} \(i \leftarrow 1\) to \(r\) \textbf{do}
18: \begin{algorithmic}
19: \(h \leftarrow \text{EMPTYSET()}\)
20: \textbf{do} \(s\) \textbf{times}
21: \begin{algorithmic}
22: \(j \leftarrow \text{RANDOMBETWEEN}(1, N)\)
23: \(v \leftarrow w[j..j + l_0 - 1]\)
24: \(h.\text{ADD}(v)\)
25: \(\hat{C}_{l,i} = h.\text{SIZE()}\)
26: \end{algorithmic}
27: \end{algorithmic}
28: \end{algorithmic}
29: \begin{algorithmic}
30: \end{algorithmic}
31: \begin{algorithmic}
32: \textbf{for} \(l \leftarrow 1\) to \(l_0\) \textbf{do}
33: \begin{algorithmic}
34: \textbf{for} \(i \leftarrow 1\) to \(r\) \textbf{do}
35: \begin{algorithmic}
36: \(\hat{d}_{l,i} \leftarrow \hat{C}_{l,i} \cdot B\)
37: \end{algorithmic}
38: \end{algorithmic}
39: \end{algorithmic}
40: \begin{algorithmic}
41: \textbf{for} \(l \leftarrow 1\) to \(l_0\) \textbf{do}
42: \begin{algorithmic}
43: \(\hat{d}_l \leftarrow \text{median of } \hat{d}_{l,1}, \hat{d}_{l,2},..\hat{d}_{l,r}\)
44: \end{algorithmic}
45: \end{algorithmic}
46: \begin{algorithmic}
47: \(\hat{m} = \max_{l=1}^{l_0} \frac{\hat{d}_l}{l}\)
48: \end{algorithmic}
49: \end{algorithmic}
50: \begin{algorithmic}
51: \textbf{return} \(\hat{m} \cdot \frac{4}{B} + \epsilon n\)
52: \end{algorithmic}

The final optimized form is ESTOPL. To make it easier to understand, it is useful to first look at ESTOPLSLOW2. ESTOPLSLOW2 is merely a rearranged version of ESTOPLSLOW1, obtained by “inlining” ESTIMATE and SSDISTEST and moving the parts where estimates are derived from sample distinct counts to lines 23.27, after the main loop. ESTOPLSLOW2 computes its output in the same way and with the same (in)efficiency as ESTOPLSLOW1.

The computations of ESTIMATE are done in the loop on lines 14..21 of ESTOPLSLOW2. On the \(l\)th iteration of the loop, values are computed for
$C_l, 1, \ldots, C_l, i$. These values determine the eventual values of $d_l, 1, \ldots, d_l, i$ and $d_l$. The loop is run $l_0$ times, corresponding to the fact that ESTOPL calls ESTIMATE $l_0$ times.

The computations of SSDISTEST are done in the inner loop on lines 15..21. On the $i$th iteration of the inner loop inside the $l$th iteration of the outer loop, a value is computed for $C_l, i$. This value determines the eventual value of $d_l, i$ via \( d_l, i \leftarrow C_l, i \cdot B \) on line 25. The loop is run \( r = \text{AmpCount} \left( \frac{3}{4}, 1 - \frac{1}{3l_0} \right) \) times on each iteration of the outer loop, corresponding to the fact that each call to ESTIMATE in ESTOPL calls SSDISTEST $r$ times.

As in SSDISTEST, on each iteration of the inner loop, a batch of $k = \frac{10N}{B^2}$ substrings of length $l$ is sampled. Denote the batch sampled on the $i$th iteration of the inner loop inside the $l$th iteration of the outer loop by $s_l, i$. Denote the $k$ strings of length $l$ that $s_l, i$ consists of by $s_l, 1, i, s_l, 2, i, \ldots, s_l, k, i$. A total of $l_0 \cdot r$ batches are sampled (for a total of $\sum_{l=1}^{l_0} rkl = rk\sum_{l=1}^{l_0} l = rk\frac{1}{2}(l^2 + l)$ symbols queried).

The variables in ESTOPLSLOW2 correspond to the following variables in ESTOPLSLOW1:

- $d_l$ corresponds to the variable of the same name in ESTOPLSLOW1, which is assigned the return value from the $l$th call to ESTIMATE, which has arguments \((w, l, B, 1 - \frac{1}{3l_0})\). Thus the final value of $d_l$ is a $B$-estimate of $d_l$ with probability $1 - \frac{1}{3l_0}$.

- $d_l, i$ corresponds to the value returned by the $i$th call to SSDISTEST inside the $l$th call to ESTIMATE, and stored in the variable $e_i$ inside the $l$th call to ESTIMATE.

- $d_l, 1, d_l, 2, \ldots, d_l, r$ contain $r$ different estimates for the number of substrings of length $l$ in $w$, corresponding to the variables $e_1, e_2, \ldots, e_r$ inside the $l$th call to ESTIMATE.

- $C_l, i$ corresponds to the variable $C$ inside the $i$th call to SSDISTEST inside the $l$th call to ESTIMATE.

The value that $C_l, i$ ends up with after the $i$th iteration of the inner loop inside the $l$th iteration of the outer loop, and the value that and $d_l, i$ eventually equals, are determined by the batch of substrings $s_l, i$: $C_l, i = \text{Distinct}(s_l, i)$ and $d_l, i = \text{Distinct}(s_l, i) \cdot B$.

### 4.6.3 The final algorithm

In this section, we finally present ESTOPL itself (Algorithm 4.7). We begin by discussing how it differs from ESTOPLSLOW2.

In ESTOPL, we also compute values for $l_0 \cdot r$ variables named $C_l, i$. These values then determine the values of the $d_l, i$, the $d_l$ and the output in the same way as
Algorithm 4.7 EstOPL, the main algorithm investigated in this text. Due to Raskhodnikova et al. [RRRS13], it is a partial \((A, \epsilon)\)-approximation algorithm for \(OPL(w)\).

1: function EstOPL\((w, A, \epsilon)\)
2: assert \(A > 1\)
3: assert \(0 < \epsilon < 1\)
4: \(n = |w|\)
5: \(l_0 = \lceil \frac{2}{A\epsilon} \rceil\)
6: assert \(2 \leq l_0 \leq n\)
7: \(N = n - l_0 + 1\)
8: \(B = \frac{A}{2\sqrt{\log l_0}}\)
9: \(s = \frac{4N}{B^2}\)
10: assert \(1 \leq s \leq N\)
11: \(r = \text{AMPCount}\left(\frac{\epsilon}{4}, 1 - \frac{1}{2\epsilon}a\right)\)
12: \(t \leftarrow \text{EMPTYTRIE}()\)
13: \(\text{do \ [s] times}\)
14: \(j \leftarrow \text{RANDOMBETWEEN}(1, N)\)
15: \(v \leftarrow w[j..j + l_0 - 1]\)
16: \(l' \leftarrow |\text{LONGESTPREFIXINTRIE}(t, v)|\)
17: \(\text{for } l \leftarrow (l' + 1) \text{ to } l_0 \text{ do}\)
18: \(\hat{C}_{l,i} \leftarrow 0\)
19: \(\hat{C}_{l,i} \leftarrow \hat{C}_{l,i} + 1\)
20: \(\text{INSERTINTOTRIE}(t, v)\)
21: \(\text{for } l \leftarrow 1 \text{ to } l_0 \text{ do}\)
22: \(\text{for } i \leftarrow 1 \text{ to } r \text{ do}\)
23: \(\hat{d}_{l,i} \leftarrow \hat{C}_{l,i} \cdot B\)
24: \(\hat{d}_l \leftarrow \text{median of } \hat{d}_{l,1}, \hat{d}_{l,2}, \ldots, \hat{d}_{l,r}\)
25: \(\hat{m} = \max_{l=1}^{l_0} \hat{d}_l\)
26: \(\text{return } \hat{m} \cdot \frac{A}{B} + \epsilon n\)
in \text{EstOPLSlow} 2. The important difference is in how the values for the \( \hat{C}_{t,i} \) are obtained.

Given \( l, \hat{d}_{t,1}, \hat{d}_{t,2}, \ldots, \hat{d}_{t,r} \) still stand for \( r \) different estimates of the number of distinct substrings of length \( l \) in \( w \), and their median \( \hat{d}_l \) stands for the combined estimate. However, these estimates are obtained via a different strategy of sampling substrings than in \text{EstOPLSlow}1 and \text{EstOPLSlow}2.

Instead of reading \( l_0 \cdot r \) batches of \( k \) substrings of varying lengths, we read only \( r \) batches of \( k \) substrings of length \( l_0 \). Denote these batches \( S_1, S_2, \ldots, S_r \). The \( i \)th batch \( S_i \) is read on the \( i \)th iteration of the main loop on lines 18..26. Denote the strings in \( S_i \) by \( S_{i,1}, S_{i,2} \ldots S_{i,k} \).

The \( \hat{C}_{t,i} \) are determined by the \( S_i \) as follows. Given any sequence of strings \( T = t_1 t_2 \ldots t_k \) and any positive integer \( l \), let \( \text{Prefixes}(l, T) \) stand for the sequence of length-\( l \) prefixes of those strings. That is, \( \text{Prefixes}(l, T) = t_1[1..l], t_2[1..l], \ldots, t_m[1..l] \). Now for all \( l \in \{1..l_p\} \) and \( i \in \{1..r\} \), \( \hat{C}_{t,i} \) gets the value \( \text{Distinct}(\text{Prefixes}(l, S_i)) \).

So for a given \( l, \hat{C}_{1,1}, \hat{C}_{1,2}, \ldots, \hat{C}_{1,r} \), and \( \hat{d}_{1,1}, \hat{d}_{1,2}, \ldots, \hat{d}_{1,r} \), the \( r \) different estimates for the number of substrings of length \( l \) in \( w \), are determined as:

\[
\hat{d}_{t,1} = \hat{C}_{t,1} \cdot B = \text{Distinct}(\text{Prefixes}(l, S_1)) \cdot B \\
\hat{d}_{t,2} = \hat{C}_{t,2} \cdot B = \text{Distinct}(\text{Prefixes}(l, S_2)) \cdot B \\
\vdots \\
\hat{d}_{t,r-1} = \hat{C}_{t,r-1} \cdot B = \text{Distinct}(\text{Prefixes}(l, S_{r-1})) \cdot B \\
\hat{d}_{t,r} = \hat{C}_{t,r} \cdot B = \text{Distinct}(\text{Prefixes}(l, S_r)) \cdot B.
\]

And for a given \( i, \hat{C}_{1,i}, \hat{C}_{2,i}, \ldots, \hat{C}_{l_0,i} \), and \( \hat{d}_{1,i}, \hat{d}_{2,i}, \ldots, \hat{d}_{l_0,i} \), the \( l_0 \) estimates computed on the \( i \)th iteration of the main loop of \text{EstOPL}, are determined as:

\[
\hat{d}_{1,i} = \hat{C}_{1,i} \cdot B = \text{Distinct}(\text{Prefixes}(1, S_i)) \cdot B \\
\hat{d}_{2,i} = \hat{C}_{2,i} \cdot B = \text{Distinct}(\text{Prefixes}(2, S_i)) \cdot B \\
\vdots \\
\hat{d}_{l_0-1,i} = \hat{C}_{l_0-1,i} \cdot B = \text{Distinct}(\text{Prefixes}(l_0-1, S_i)) \cdot B \\
\hat{d}_{l_0,i} = \hat{C}_{l_0,i} \cdot B = \text{Distinct}(\text{Prefixes}(l_0, S_i)) \cdot B = \text{Distinct}(S_i) \cdot B.
\]

In \text{EstOPL}, these estimates, \( \hat{d}_{1,i}, \hat{d}_{2,i}, \ldots, \hat{d}_{l_0,i} \), are \emph{not} independent, while in \text{EstOPLSlow2}, the corresponding estimates are independent. This is because in \text{EstOPLSlow2}, each of the estimates \( \hat{d}_{t,i} \) is determined by a separate sample of substrings \( s_{t,i} \) that is used only for that estimate. In \text{EstOPL} in contrast, for every \( i \in \{1..r\} \), the estimates \( \hat{d}_{1,i}, \hat{d}_{2,i}, \ldots, \hat{d}_{l_0,i} \) are all determined by a single sample \( S_i \) as described above.
This means ESTOPL’s estimates have a higher probability of error, even as it has a better query complexity than ESTOPLSLOW2. But its heightened error probability remains within the bounds required, as proved later.

(Note that even in ESTOPL, the estimates $\hat{d}_{l1}, \hat{d}_{l2}, \ldots, \hat{d}_{li}$ are independent, and thus $\hat{d}_l$ is still a $B$-estimate for $d_l$ with probability $\geq 1 - \frac{1}{3l_0}$.)

We have now described exactly how the output of ESTOPL is determined as a function of the samples $S_1, S_2, \ldots, S_r$. But we have not yet described how ESTOPL computes its output from the samples. We do so below.

All the $\hat{C}_{l,i}$ are initialized as 0. The main loop on lines 18..26 runs $r = \text{AMP_COUNT} \left(\frac{3}{4}, 1 - \frac{1}{3l_0}\right)$ times. On the $i$th iteration, values are computed for $\hat{C}_{1,i}, \hat{C}_{2,i}, \ldots, \hat{C}_{l_0,i}$. After the final ($r$th) iteration we have values for all $r \cdot l_0$ of the $\hat{C}_{l,i}$.

On the $i$th iteration of the main loop, the inner loop on lines 20..26 iterates over the substrings in $S_i$, with $s$ containing each substring in turn. On each iteration, we increment $\hat{C}_{l,i}$ for every $l \in 1..l_0$ for which the prefix $s[1..l]$ has not yet been seen as a prefix of an earlier substring of $S_i$. On the first iteration, for example, every $\hat{C}_{1,i}, \hat{C}_{2,i}, \ldots, \hat{C}_{l_0,i}$ gets incremented from 0 to 1. After the last iteration, $\hat{C}_{l,i}$ has been incremented exactly $\text{Distinct(Prefixes}(l, S_i))$ times, so its value is as claimed earlier.

We use a “trie” data structure to keep track of prefixes seen so far. We have the following operations and resource usage:

- **EMPTY_TRIE().** Return a new empty trie. $O(1)$ time and space.
- **INSERT_INTO_TRIE(t, s).** Insert a string $s$ of length $l$ into a trie $t$. $O(l)$ time.
- **LONGEST_PREFIX_IN_TRIE(t, s).** Given a string $s$ of length $l$, return the longest prefix of $s$ that is also a prefix of some string that was previously inserted into $t$. $O(l)$ time.
- **A trie that has had $k$ strings of length $l$ inserted into it takes up $O(kl)$ space.**

Obviously, more operations could be supported, but these are the only ones needed.

The trie $t$ is used in the inner loop to keep track of seen prefixes as follows (one trie handles all prefix lengths simultaneously). Before the first iteration of the inner loop, $t$ is an empty trie. On each iteration, we find the length $l'$ of the longest prefix of $s$ that was also a prefix of some substring seen earlier in $S_i$ (line 23). On the first iteration, $l'$ is naturally 0. We know now that all prefixes of $s$ of length $l'$ and less have been seen, and none of the prefixes of $s$ of length $l' + 1$ and greater have yet been seen. So we increment $\hat{C}_{l'+1,i}, \hat{C}_{l'+2,i}, \ldots, \hat{C}_{l_0,i}$. Naturally, if $l'$ is $l_0$, meaning that the substring $s$ has been seen before, we increment nothing; and if $l'$ is 0, meaning that the first symbol of $s$ has not been the first symbol of any earlier string in $S_i$, we increment all $\hat{C}_{1,i}, \hat{C}_{2,i}, \ldots, \hat{C}_{l_0,i}$. 
After the main loop is finished, we compute values for all \( \hat{d}_{i,l} \) and \( \hat{d}_l \) from \( \hat{C}_{i,j} \), and finally the output from the \( \hat{d}_l \), in the same way as in EstOPLSlow2.

4.6.4 Error bounds and complexity

We will now demonstrate nontrivial guarantees for the quality of the approximations produced by EstOPL (Proposition 4.5) and for EstOPL’s asymptotic resource usage (Proposition 4.6). Proposition 4.6 also shows that EstOPL runs successfully for a nontrivial range of inputs.

**Proposition 4.5.** EstOPL is a partial \((A, \epsilon)\)-approximation algorithm for \(OPL(w)\). (Also EstOPLSlow1 and EstOPLSlow2).

**Proof.** For all \( l \in \{1..l_0\} \), \( \hat{d}_l \) is a \( B \)-estimate of \( d_l \) with probability at least \( 1 - \frac{1}{3l_0} \). It is not a \( B \)-estimate with probability at most \( \frac{1}{3l_0} \). So the probability that one or more of the \( \hat{d}_l \) fails to be a \( B \)-estimate of \( d_l \) is at most \( l_0 \frac{1}{3l_0} = \frac{1}{3} \) (and this does not require independence of the \( \hat{d}_l \)). So the probability that all the \( \hat{d}_l \) are \( B \)-estimates is at least \( \frac{2}{3} \).

(This holds in each of EstOPL, EstOPLSlow1 and EstOPLSlow2.) If all the \( \hat{d}_l \) are \( B \)-estimates for \( d_l \), then \( \hat{m} = \max_{l=1}^{l_0} \frac{\hat{d}_l}{T} \) is a \( B \)-estimate for \( m = \max_{l=1}^{l_0} \frac{d_l}{T} \). That is,

\[
m/B \leq \hat{m} \leq MB
\]

and also \( \hat{m}/B \leq m \leq \hat{m}B \).

Then by Proposition 4.1:

\[
\hat{m}/B \leq m \leq OPL(w) \leq 4 \left( m \log l_0 + \frac{n}{l_0} \right) \leq 4 \left( \hat{m}B \log l_0 + \frac{n}{l_0} \right).
\]

So to simplify:

\[
\hat{m}/B \leq OPL(w) \leq 4 \left( \hat{m}B \log l_0 + \frac{n}{l_0} \right).
\]

Meanwhile, the bounds that we need to prove are

\[
OPL(w)/A - \epsilon n \leq \hat{m} \frac{A}{B} + \epsilon n \leq OPL(w) \cdot A + \epsilon n.
\]

That is,

\[
\hat{m} \frac{A}{B} + \epsilon n \geq OPL(w)/A - \epsilon n \tag{5}
\]

and \( \hat{m} \frac{A}{B} + \epsilon n \leq OPL(w) \cdot A + \epsilon n. \tag{6} \)
In the remainder of this proof we derive (5) and (6) from (4). (6) is trivial:

\[ \hat{m} / B \leq \text{OPL}(w) \iff \hat{m} \leq \text{OPL}(w) \cdot B \]
\[ \iff \hat{m} A / B + \epsilon n \leq \text{OPL}(w) \cdot A + \epsilon n. \]

For (5), first observe that

\[ 4 \left( \hat{m} B \log l_0 + \frac{n}{l_0} \right) \geq \text{OPL}(w) \]
\[ \iff 4 \hat{m} B \log l_0 \geq \text{OPL}(w) - 4 \frac{n}{l_0} \]
\[ \iff \hat{m} \geq \frac{\text{OPL}(w) - 4 \frac{n}{l_0}}{4B \log l_0}. \]

Then multiply by \( \frac{A}{B} \) and add \( \epsilon n \) to get

\[ \hat{m} A / B + \epsilon n \geq A \frac{\text{OPL}(w) - 4 \frac{n}{l_0}}{4B^2 \log l_0} + \epsilon n. \] (7)

In the expression on the right-hand side, replace \( B \) and then \( l_0 \) with their definitions in terms of \( A \) and \( \epsilon \) to get

\[ A \frac{\text{OPL}(w) - 4 \frac{n}{l_0}}{4 \left( \frac{A}{2 \sqrt{\log \left[ \frac{A}{\epsilon} \right]} \right) \log l_0} + \epsilon n = A \frac{\text{OPL}(w) - 4 \frac{n}{l_0}}{4A \log l_0} + \epsilon n \]
\[ = \log \left[ \frac{2}{\epsilon A} \right] (\text{OPL}(w) - 4 \frac{n}{l_0}) + \epsilon n = \frac{\log \left[ \frac{2}{\epsilon A} \right] (\text{OPL}(w) - 4 \frac{n}{l_0})}{A \log l_0} + \epsilon n \]
\[ = \frac{\text{OPL}(w) - 4 \frac{n}{l_0}}{A} + \epsilon n. \]

Then note that

\[ \frac{\text{OPL}(w) - 4 \frac{n}{l_0}}{A} + \epsilon n \geq \frac{\text{OPL}(w) - 4 \frac{n}{l_0}}{A} + \epsilon n \]
\[ = \frac{\text{OPL}(w) - 2nA \epsilon + \epsilon n}{A} = \frac{\text{OPL}(w)}{A} - 2n \epsilon + \epsilon n \]
\[ = \frac{\text{OPL}(w)}{A} - \epsilon n. \]
So in summary, we have
\[ \frac{\hat{m}A}{B} + \epsilon n \geq A \frac{\text{OPL}(w) - 4\frac{n}{l_0}}{4B^2 \log l_0} + \epsilon n = \frac{\text{OPL}(w) - 4\frac{n}{(\frac{2}{x})}}{A} + \epsilon n \]
proving (5), as desired.

**Proposition 4.6.** Let \( x \) and \( y \) be numbers for which
\[ 0 < x < \frac{1}{2} \text{ and } x < y < x + 1. \]
For all sufficiently large \( n \), if \( w \) is a string of length \( n \) and we set
\[ A(n) = n^x \text{ and } \epsilon(n) = n^{-y}, \]
then \( \text{ESTOPL}(w, A(n), \epsilon(n)) \) runs successfully, and has a time, space and query complexity of
\[ \tilde{O}(n^{1-3x+y}). \]

**Proof.** Let \( x \) and \( y \) be as given in the statement of the proposition. We will first show that for all sufficiently large \( n \), if \( w \) is a string of length \( n \), then \( \text{ESTOPL}(w, n^x, n^{-y}) \) runs successfully, that is, that all the assert statements succeed. The assert statements behave as follows:

- “\( A > 1 \)” becomes \( n^x > 1 \). Since \( x > 0 \) and \( n \geq 1 \), this clearly holds.
- “\( 0 < \epsilon < 1 \)” becomes \( 0 < n^{-y} < 1 \). Since \( y > 0 \), this holds.
- \( l_0 = \left\lceil \frac{n^x}{n^{y-x}} \right\rceil = \left\lceil 2n^{y-x} \right\rceil \). So “\( 1 \leq l_0 \leq n \)” becomes \( 1 \leq \left\lceil 2n^{y-x} \right\rceil \leq n \). \( 1 \leq \left\lceil 2n^{y-x} \right\rceil \) holds because \( y - x > 0 \). \( \left\lceil 2n^{y-x} \right\rceil \leq n \) holds for all sufficiently large \( n \) because \( y - x < 1 \).

Finally, we have
\[ B = \frac{n^x}{2\sqrt{\log l_0}} = \frac{n^x}{2\sqrt{\log \left\lceil 2n^{y-x} \right\rceil}} \text{ and } N = n - \left\lceil 2n^{y-x} \right\rceil + 1, \]
and so
\[ s = \frac{10N}{B^2} = \frac{10N}{\left( \frac{n^x}{2\sqrt{\log \left\lceil 2n^{y-x} \right\rceil}} \right)^2} = \frac{10N}{n^x} \left( \frac{2}{4\log \left\lceil 2n^{y-x} \right\rceil} \right)^2 = 40N \log \left\lceil 2n^{y-x} \right\rceil n^{-2x}. \]
Then “$s \leq N$” becomes

$$40N \log \left[ 2n^{y-x} \right] n^{-2x} \leq N$$

$$\iff \log \left[ 2n^{y-x} \right] n^{-2x} \leq \frac{1}{40},$$

which is true for all sufficiently large $n$ because the exponent $-2x$ is negative.

“$1 \leq s$” becomes

$$1 \leq 40N \log \left[ 2n^{y-x} \right] n^{-2x}$$

$$\iff \frac{1}{40} \leq (n - \left[ 2n^{y-x} \right] + 1) \log \left[ 2n^{y-x} \right] n^{-2x}$$

$$\iff \frac{1}{40} \leq (n \log \left[ 2n^{y-x} \right] n^{-2x}) - \left( \left[ 2n^{y-x} \right] \log \left[ 2n^{y-x} \right] n^{-2x} \right) \left( \log \left[ 2n^{y-x} \right] n^{-2x} \right)$$

For this to hold for all sufficiently large $n$, it suffices that (1) $1 - 2x > 0$ and (2) $1 - 2x > y - 3x$. (1) ensures that the leftmost term grows without bound as $n \to \infty$. (2) ensures that the growth of the leftmost term dominates the growth of the absolute value of the middle term.

(1) $\iff x < \frac{1}{2}$, so (1) holds. (2) $\iff y < x + 1$, so (2) also holds. So $1 \leq s$ is also true for all sufficiently large $n$.

Thus, all the assertions hold for sufficiently large $n$. So EstOPL($w, n^x, n^{-y}$) runs successfully for sufficiently large $n$.

We will now prove the claim about EstOPL’s time, space and query complexity. The asymptotic resource usage of the main loop on lines 18..26 dominates over everything else, so it suffices to examine that.

We will begin with the query complexity. The number of characters queried in the main loop is

$$r \cdot \left[ s \right] \cdot l_0 = r \cdot \left[ \frac{10N}{B^2} \right] \cdot l_0.$$

We will look at each of the factors separately. First,

$$l_0 = \left[ 2n^{y-x} \right] = O(n^{y-x}).$$

Then, since $\text{AmpCount}\left(\frac{3}{4}, p\right)$ grows with $p$ like $O\left(\log \frac{1}{1-p}\right)$,

$$r = \text{AmpCount}\left(\frac{3}{4}, 1 - \frac{1}{3l_0}\right) = \text{AmpCount}\left(\frac{3}{4}, 1 - \frac{1}{3\left[ 2n^{y-x} \right]}\right)$$

$$= O\left(\log \frac{1}{1 - \frac{1}{3\left[ 2n^{y-x} \right]}\right}) = O\left(\log \left[ 2n^{y-x} \right]\right) = O(\log n^{y-x}).$$
Then,
\[ N = n - l_0 + 1 = O(n) - O(n^{y-x}) + O(1) = O(n), \]
since \( y - x < 1 \).

Finally,
\[ \frac{1}{B^2} = \frac{1}{\left( \frac{n^y}{2\sqrt{\log l_0}} \right)^2} = n^{-2x}4 \log l_0 = O(n^{-2x} \log n^{y-x}). \]

So we have
\[ \lceil s \rceil = \left\lceil \frac{10N}{B^2} \right\rceil = \left\lceil O(n)O(n^{-2x} \log n^{y-x}) \right\rceil = O(n^{-2x} \log n^{y-x}). \]

So for the query complexity we have
\[ r \cdot \lceil s \rceil \cdot l_0 = O(\log n^{y-x}) \cdot O(n^{1-2x} \log n^{y-x}) \cdot O(n^{y-x})
\[ = O(n^{1-3x+y}(\log n^{y-x})^2)
\[ = \tilde{O}(n^{1-3x+y}) \]
as desired.

For the time complexity, note that the inner loop on lines 20..26 is run \( r \cdot \lceil s \rceil = O(n^{1-2x}(\log n^{y-x})^2) \) times. Inside this loop:

- \( l_0 = \lceil 2n^{y-x} \rceil = O(n^{y-x}) \) characters are read (line 22). \( O(n^{y-x}) \) time.

- \textsc{LongestPrefixInTrie} is called with a string of length \( l_0 \) (line 23). \( O(n^{y-x}) \) time.

- Up to \( l_0 \) variables are incremented (lines 24-25). \( O(n^{y-x}) \) time.

- A string of length \( l_0 \) is inserted into a trie (line 26). \( O(n^{y-x}) \) time.

So each iteration of the loop takes \( O(n^{y-x}) \) time. So the time complexity is
\[ O(n^{1-2x}(\log n^{y-x})^2)O(n^{y-x}) = O(n^{1-3x+y}(\log n^{y-x})^2) = \tilde{O}(n^{1-3x+y}) \]
as desired.

For the space complexity, note that at its fullest, the trie had \( \lceil s \rceil \) strings of length \( l_0 \) inserted into it. This means the trie takes up space
\[ O(\lceil s \rceil \cdot l_0) = O(O(n^{1-2x} \log n^{y-x})O(n^{y-x}))
\[ = O(n^{1-3x+y} \log n^{y-x}) = \tilde{O}(n^{1-3x+y}) \]
as desired.
5 Distinguishing strings with short optimal parsings from strings with long optimal parsings in sublinear time

In this section, we will demonstrate the theoretically interesting fact that EstOPL enables us to distinguish strings with short optimal parsings from strings with long optimal parsings in sublinear time, in a sense that will be described below.

In the original article describing EstOPL [RRRS13], the following statement is made:

[Using EstOPL,] for any $\alpha > 0$, we can distinguish, in sublinear time $\tilde{O}(n^{1-\alpha})$, strings compressible to $O(n^{1-\alpha})$ symbols from strings only compressible to $\Omega(n)$ symbols.

(And a footnote adds: “To see this, set $A = o(n^{\alpha/2})$ and $\epsilon = o(n^{-\alpha/2}).”)

The idea is not discussed further in the article. Its precise meaning is not immediately obvious: what does it mean for a string or strings to be compressible to $O(n^x)$ or to $\Omega(n)$? What does it mean to distinguish between such strings? This section describes a way to make this notion fully precise. Thus, this section is essentially an extensively unpacked version of the sentence quoted above.

However, instead of strings compressible to $O(n^x)$ or to $\Omega(n)$, we consider strings with optimal parsings of length $O(n^x)$ or $\tilde{\Omega}(n)$; this notion is made precise below.

We do this so as to avoid having to consider the exact relation between LZ77 compressibility and optimal parsing length, which is not completely straightforward (this relation was briefly examined in Section 3.2). As with “$\tilde{O}$”, the tilde in “$\tilde{\Omega}$” indicates that we ignore logarithmic factors. This is necessary because the optimal parsing length of a string of length $n$ over a constant alphabet is at most $O(n/\log n)$ [LZ76]. (Stated formally using the definitions given below: there is no family of strings whose optimal parsing length grows like $\Omega(n)$ but there are families of strings whose optimal parsing length grows like $\tilde{\Omega}(n)$.)

The main result of this section is Theorem 5.2 which, still speaking informally, states that for any $\alpha \in (0, 1)$ and $\epsilon > 0$, strings with optimal parsing length $O(n^\alpha)$ can be distinguished from strings with optimal parsing length $\tilde{\Omega}(n)$ in (sublinear) time $\tilde{O}(n^{\alpha+\epsilon})$. The resemblance to the sentence quoted above is obvious. Due to the presence of the “$\epsilon$”, our claim would seem to be very slightly weaker than that in original article [RRRS13]; we are unsure of the reason for this.

We will now begin to precisely define the concepts involved. The following definitions let us talk about the asymptotic growth rates of the optimal parsing lengths of strings:

**Definition.** A *family* of strings is any infinite set $S$ of strings. $S_n$ denotes the set of strings of length $n$ in $S$. 
Definitions. Given a family of strings $S$:

- If the function $F(n) = \max_{w \in S_n}[\text{OPL}(w)]$ grows like $F(n) \sim O(g(n))$, then we say that the optimal parsing length of strings in $S$ grows like $O(g(n))$.\(^6\) We can also say that $S$ is a family of strings whose optimal parsing length grows like $O(g(n))$. (And exactly similarly for “$O(g(n))$”.)
- If the function $f(n) = \min_{w \in S_n}[\text{OPL}(w)]$ grows like $f(n) \sim \Omega(g(n))$, then we say that the optimal parsing length of strings in $S$ grows like $\Omega(g(n))$. We can also say that $S$ is a family of strings whose optimal parsing length grows like $\Omega(g(n))$. (And exactly similarly for “$\Omega(g(n))$”.)

In what follows, we will be interested in the situation where have an $\alpha \in (0, 1)$, and two families of strings $S^- \text{ and } S^+$, such that the optimal parsing length of strings in $S^-$ grows like $O(n^\alpha)$ and the optimal parsing length of strings in $S^+$ grows like $\tilde{\Omega}(n)$. In this situation, we will call $S^-$ the “strings with (asymptotically) short parsings”, and $S^+$ the “strings with (asymptotically) long parsings”.

We will be concerned with “distinguishing” $S^-$ from $S^+$, in a sense which can be made precise with the following definition:

**Definitions.** Given two families of strings $S$ and $T$ and an algorithm $\mathcal{A}$,

- **$\mathcal{A}$ simply distinguishes $S$ from $T$** if for all $s \in S$ and $t \in T$, $\mathcal{A}(s) = 1$ and $\mathcal{A}(t) = 2$.
- **$\mathcal{A}$ eventually distinguishes $S$ from $T$** if for some $n_0 \in \mathbb{N}$, and for all $s \in S$ such that $|s| \geq n_0$ and $t \in T$ such that $|t| \geq n_0$, $\mathcal{A}(s) = 1$ and $\mathcal{A}(t) = 2$.
- **$\mathcal{A}$ eventually probabilistically distinguishes $S$ from $T$** if for some $n_0 \in \mathbb{N}$, and for all $s \in S$ such that $|s| \geq n_0$ and $t \in T$ such that $|t| \geq n_0$, $\mathcal{A}(s) = 1$ with probability $\geq \frac{2}{3}$ and $\mathcal{A}(t) = 2$ with probability $\geq \frac{2}{3}$.

We will end up demonstrating (in Theorem 5.2) that for any $\alpha \in (0, 1)$, there is an algorithm $\mathcal{A}$ such that given any family $S^-$ of strings whose optimal parsing length grows like $O(n^\alpha)$ and $S^+$ of strings whose optimal parsing length grows like $\tilde{\Omega}(n)$, $\mathcal{A}$ eventually probabilistically distinguishes $S^+$ from $S^-$, and runs in sublinear time.$^7$

We are now ready discuss how ESTOPL can be used to accomplish this.

Recall that the output of ESTOPL($w, A, \epsilon$) is an approximation of $\text{OPL}(w)$ that is in the range $[\text{OPL}(w)/A - \epsilon n, \text{OPL}(w)/A + \epsilon n]$ with probability $\geq \frac{2}{3}$, where $n = |w|$.

We can examine the asymptotic behavior of these bounds on the output as follows. Let $\alpha \in (0, 1)$, $S^-$ be a family of strings whose optimal parsing length grows like

---

$^6$For the purposes of this definition, let the maximum of an empty set be 0 and the minimum of an empty set be $\infty$.

$^7$The notions of simply distinguishing and eventually distinguishing are not used in the rest of this text; they are included only to make the notion of eventually probabilistically distinguishing easy to understand by contrast.
O(n^\alpha) and S^+ be a family of strings whose optimal parsing length grows like \( \tilde{\Omega}(n) \). Set \( A = n^x \) and \( \epsilon = n^{-y} \) for some \( x, y \) as in Proposition 4.6.

Now the upper bound of the output for strings with asymptotically short parsings grows with \( n \) like

\[
\max_{w \in S^-_n} \left[ \text{OPL}(w)n^x + n^{-y}n \right]
= \max_{w \in S^-_n} \left[ \text{OPL}(w) \right]n^x + n^{-y}n
= O(n^\alpha)n^x + n^{-y}n = O(n^{\alpha + x} + n^{1-y}).
\]

And the lower bound of the output for strings with asymptotically long parsings grows with \( n \) like

\[
\min_{w \in S^+_n} \left[ \text{OPL}(w)/n^x - n^{-y}n \right]
= \tilde{\Omega}(n)/n^x - n^{-y}n = \tilde{\Omega}(n^{1-x} - n^{1-y}).
\]

Now if \( x, y \) and \( \alpha \) were chosen such that \( O(n^{\alpha + x} + n^{1-y}) \) becomes \( O(n^X) \) and \( \tilde{\Omega}(n^{1-x} - n^{1-y}) \) becomes \( \tilde{\Omega}(n^Y) \) for some \( X < Y \), and \( x \) and \( y \) satisfy the constraints in Proposition 4.6, then for all sufficiently large \( n \) it would be the case that

\[
\max_{w \in S^-_n} \left[ \text{OPL}(w)n^x + n^{-y}n \right] < n^{1/2(X+Y)} < \min_{w \in S^+_n} \left[ \text{OPL}(w)/n^x - n^{-y}n \right].
\]

That is, for all sufficiently large \( n \), the upper bound of the output for a string of length \( n \) in \( S^- \) would be less than the lower bound of the output for a string of length \( n \) in \( S^+ \). Moreover, the number \( n^{1/2(X+Y)} \) would be between these bounds.

Now constructing an algorithm that eventually probabilistically distinguishes \( S^- \) from \( S^+ \) would be easy. Given a string \( w \in S^- \cup S^+ \) of length \( n \), the algorithm would do the following:

1. Compute \( \hat{P} = \text{ESTOPL}(w, n^x, n^{-y}) \).

2. If \( \hat{P} < n^{1/2(X+Y)} \), return 1 (i.e. guess that \( w \in S^- \)). Otherwise, return 2 (i.e. guess that \( w \in S^+ \)).

For all sufficiently large \( n \), the guess would be correct with probability \( \geq \frac{2}{3} \). This algorithm would run in time \( \tilde{O}(n^{1-3x+y}) \).

Choosing \( x, y \) and \( \alpha \) this way is indeed possible; Lemma 5.1 below states the constraints that \( x, y \) and \( \alpha \) need to satisfy.

**Lemma 5.1.** Let \( \alpha \in (0, 1) \). If the following holds for \( x, y \) and \( \alpha \)

\[
0 < x < (1/2)(1 - \alpha) \quad (8)
\]
\[
x < y < x + 1, \quad (9)
\]
then there is an algorithm that, for any family $S^-$ of strings whose optimal parsing length grows like $O(n^\alpha)$ and any family $S^+$ of strings whose optimal parsing length grows like $\tilde{\Omega}(n)$, eventually probabilistically distinguishes $S^-$ from $S^+$ in $\tilde{O}(n^{1-3x+y})$ time, space and queries.

Proof. For $O(n^{\alpha+x} + n^{1-y}) \leq \tilde{\Omega}(n^{1-x} - n^{1-y})$ to hold, it suffices that:

\[
\begin{cases}
1 - x > 1 - y \iff x < y \\
\max(\alpha + x, 1 - y) < 1 - x \iff \alpha + x < 1 - x \iff x < \frac{1}{2}(1 - \alpha).
\end{cases}
\]

Now combine these with $0 < x < \frac{1}{2}$ and $x < y < x + 1$ from Proposition 4.6 and simplify. (10) is already implied by $x < y < x + 1$. And $x < \frac{1}{2}$ is implied by (11) because $\alpha \in (0, 1)$.

So if $x$, $y$, and $\alpha$ satisfy the above, then $O(n^{\alpha+x} + n^{1-y})$ becomes $O(n^X)$ with $X = \max(\alpha + x, 1 - y)$ and $\tilde{\Omega}(n^{1-x} - n^{1-y})$ becomes $\tilde{\Omega}(n^Y)$ with $Y = 1 - x$, and we are guaranteed that $X < Y$. We can then construct the desired algorithm as described above the statement of this Lemma.

\[\]

Now the natural next question is the following: given $\alpha \in (0, 1)$, which $x$ and $y$ satisfying (8) and (9) yield the best running time, i.e. minimize $1 - 3x + y$, and what is the resulting running time? Answering this question gives us the main result of this section, whose significance was discussed at the beginning of the section:

**Theorem 5.2.** Let $\alpha \in (0, 1)$ and $\varepsilon > 0$. There is an algorithm that, for any family $S^-$ of strings whose optimal parsing length grows like $O(n^\alpha)$ and any family $S^+$ of strings whose optimal parsing length grows like $\tilde{\Omega}(n)$, eventually probabilistically distinguishes $S^-$ from $S^+$ and runs in $\tilde{O}(n^{\alpha+\varepsilon})$ time, space and queries.

Proof. Clearly the settings of $x$ and $y$ that minimize $1 - 3x + y$ are the ones where $x$ is as large as possible and $y$ is as small as possible. If the inequalities (8) and (9) in Lemma 5.1 were not strict, we would set $x = \frac{1}{2}(1 - \alpha)$ and $y = x = \frac{1}{2}(1 - \alpha)$. Then we would have

\[
1 - 3x + y = 1 - 3 \left( \frac{1}{2} (1 - \alpha) \right) + \frac{1}{2} (1 - \alpha) = 1 - (1 - \alpha) = \alpha.
\]

Since they are strict, we will instead set $x$ to a value less than but close to $\frac{1}{2}(1 - \alpha)$ and $y$ to a value greater than but close to $x$. So, let $e_x > 0$ and $e_y > 0$ and set
\[
x = \frac{1}{2}(1 - \alpha) - e_x \quad \text{and} \quad y = x + e_y = \frac{1}{2}(1 - \alpha) - e_x + e_y.
\]

Then we have
\[
1 - 3x + y = 1 - 3 \left( \frac{1}{2}(1 - \alpha) - e_x \right) + \left( \frac{1}{2}(1 - \alpha) - e_x + e_y \right)
= 1 - \frac{3}{2}(1 - \alpha) + 3e_x + \frac{1}{2}(1 - \alpha) - e_x + e_y
= \alpha + 2e_x + e_y.
\]

So now, given \( \varepsilon > 0 \), choose \( e_x > 0 \) and \( e_y > 0 \) so that \( 2e_x + e_y < \varepsilon \) and set \( x = \frac{1}{2}(1 - \alpha) + e_x \) and \( y = x + e_y \). Now \( x \) and \( y \) satisfy (8) and (9) so by Lemma 5.1, an algorithm of the desired kind exists that runs in time, space and queries \( \tilde{O}(n^{1-3x+y}) = \tilde{O}(n^{\alpha+2e_x+e_y}) = \tilde{O}(n^{\alpha+\varepsilon}) \).

\[\square\]

6 Experimental investigation

In this section, we investigate the accuracy of the estimates produced by EstOPL and practical usefulness of the algorithm. For this purpose, we implemented EstOPL in Python. The implementation is available at https://github.com/oneb/estcompr. We use it to produce estimates of the optimal parsing lengths of several large files (100MB+) under several settings of the approximation parameters \( A \) and \( \epsilon \).

The results are described in Section 6.3; the full tables of results are in Appendix 4. Section 6.1 describes the files used and Section 6.2 describes the choice of \((A, \epsilon)\)-values. Note that we test the algorithm in both “sublinear” and “non-sublinear” modes, that is, with \((A, \epsilon)\)-settings where the number of characters queried is less than the number of characters in the input, and also settings where it is greater. Finally in Sections 6.4 and 6.5 we consider the meaning of the results.

Throughout this section, if “\( w \)” in expressions like \( \text{OPL}(w) \) is not otherwise defined, it stands for the contents of some file that is clear from context.

6.1 Choice of test files

We will be testing EstOPL with five different files, listed in Table 2.

The contents of the files are as follows:

- **enwik8**\(^8\) contains the first 100 million bytes of a dump of all text from the English-language Wikipedia. The dump was taken on March 3, 2006. So the file consist of non-repetitive, real-world English text.

- **einstein.en.txt**\(^9\) contains all the versions of the English-language Wikipedia article about Albert Einstein up to November 10, 2006. So it consists of highly

---

\(^8\)Obtained from http://mattmahoney.net/dc/textdata.html.
\(^9\)Obtained from http://pizzachili.dcc.uchile.cl/repcorpus/.
Table 2: Files used for testing ESTOPL, along with their exact optimal parsing lengths.

<table>
<thead>
<tr>
<th>name</th>
<th>size (bytes)</th>
<th>optimal parsing length</th>
<th>as %</th>
</tr>
</thead>
<tbody>
<tr>
<td>enwik8</td>
<td>100,000,000</td>
<td>8,220,688</td>
<td>8.22%</td>
</tr>
<tr>
<td>einstein.en.txt</td>
<td>467,626,544</td>
<td>89,467</td>
<td>0.02%</td>
</tr>
<tr>
<td>kernel</td>
<td>257,961,616</td>
<td>793,915</td>
<td>0.31%</td>
</tr>
<tr>
<td>random100</td>
<td>100,000,000</td>
<td>35,484,830</td>
<td>35.48%</td>
</tr>
<tr>
<td>almostuniform</td>
<td>100,000,000</td>
<td>959,042</td>
<td>0.96%</td>
</tr>
</tbody>
</table>

repetitive English text, which is highly compressible. Indeed, as Table 2 shows, it has a very short optimal parsing.

- **kernel**\(^{10}\) contains the source code of 36 versions of the Linux kernel. Its contents are also highly repetitive.

- **random100** is an artificial file that consists of 100 million randomly generated (8-bit) bytes. So it is a prototypical incompressible file.

- **almostuniform** is another artificial file. It consists of 100 million bytes, each of which is a constant with probability 0.99 and random with probability 0.01. So ~99% of its bytes are identical and ~1% are random.

The “alphabet” implicitly used throughout this section consists of the 256 different 8-bit bytes.

### 6.2 Choice of test parameters

To begin, note that the number of characters queried by ESTOPL is

\[
rvs = \text{AmpCount} \left( \frac{3}{4} \cdot 1 - \frac{1}{3l_0} \right) \cdot \left\lfloor \frac{10N}{B^2} \right\rfloor \cdot l_0
\]

\[
= \text{AmpCount} \left( \frac{3}{4} \cdot 1 - \frac{1}{3l_0} \right) \cdot \left\lfloor \frac{10(n - l_0 + 1)}{\left( \frac{A}{2\sqrt{\log l_0}} \right)^2} \right\rfloor \cdot l_0
\]

\[
= \text{AmpCount} \left( \frac{3}{4} \cdot 1 - \frac{1}{3l_0} \right) \cdot \left\lfloor \frac{40(n - l_0 + 1) \log l_0}{A^2} \right\rfloor \cdot l_0.
\]

So the fraction of the input string queried is

\[
F_q = \left( \text{AmpCount} \left( \frac{3}{4} \cdot 1 - \frac{1}{3l_0} \right) \cdot \left\lfloor \frac{40(n - l_0 + 1) \log l_0}{A^2} \right\rfloor \cdot l_0 \right) / n.
\]

Fix a number \( f_q \) and consider the task of finding parameters \( A \) and \( \epsilon \) for which \( F_q \) is approximately equal to \( f_q \). If we also fix \( l_0 \) and \( n \), the only remaining variable is

\(^{10}\)Also obtained from http://pizzachili.dcc.uchile.cl/repcorpus/.
<table>
<thead>
<tr>
<th>$f_q$</th>
<th>$l_0$</th>
<th>$A$</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>6.45</td>
<td>0.2946</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>42.78</td>
<td>0.006596</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>145.16</td>
<td>0.0004431</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>409.5</td>
<td>0.00003843</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>1087.1</td>
<td>0.0000036</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>9.12</td>
<td>0.2083</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>60.5</td>
<td>0.004664</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>205.29</td>
<td>0.0003133</td>
</tr>
<tr>
<td>2</td>
<td>128</td>
<td>579.11</td>
<td>0.00002717</td>
</tr>
<tr>
<td>2</td>
<td>512</td>
<td>1537.39</td>
<td>0.000002545</td>
</tr>
<tr>
<td>0.75</td>
<td>2</td>
<td>14.89</td>
<td>0.1276</td>
</tr>
<tr>
<td>0.75</td>
<td>8</td>
<td>98.79</td>
<td>0.002856</td>
</tr>
<tr>
<td>0.75</td>
<td>32</td>
<td>335.24</td>
<td>0.0001918</td>
</tr>
<tr>
<td>0.75</td>
<td>128</td>
<td>945.69</td>
<td>0.00001664</td>
</tr>
<tr>
<td>0.75</td>
<td>512</td>
<td>2510.55</td>
<td>0.000001559</td>
</tr>
<tr>
<td>0.125</td>
<td>2</td>
<td>36.48</td>
<td>0.05208</td>
</tr>
<tr>
<td>0.125</td>
<td>8</td>
<td>241.99</td>
<td>0.001166</td>
</tr>
<tr>
<td>0.125</td>
<td>32</td>
<td>821.15</td>
<td>0.00007832</td>
</tr>
<tr>
<td>0.125</td>
<td>128</td>
<td>2316.45</td>
<td>0.000006793</td>
</tr>
<tr>
<td>0.125</td>
<td>512</td>
<td>6149.57</td>
<td>0.0000006363</td>
</tr>
<tr>
<td>0.03125</td>
<td>2</td>
<td>72.96</td>
<td>0.02604</td>
</tr>
<tr>
<td>0.03125</td>
<td>8</td>
<td>483.97</td>
<td>0.000583</td>
</tr>
<tr>
<td>0.03125</td>
<td>32</td>
<td>1642.31</td>
<td>0.00003916</td>
</tr>
<tr>
<td>0.03125</td>
<td>128</td>
<td>4632.91</td>
<td>0.000003397</td>
</tr>
<tr>
<td>0.03125</td>
<td>512</td>
<td>12299.1</td>
<td>0.0000003182</td>
</tr>
</tbody>
</table>

Table 3: Parameters used for test runs of EstOPL. (All of the parameters are runnable.)

$A$, and there is then a practically-unique value of $A$ that brings $F_q$ as close to $f_q$ as possible (actually a small range of values, because of the ceiling function.) This $A$ is also practically independent of $n$. From $l_0$ and $A$, $\epsilon$ is also determined. So in short, when $f_q$ and $l_0$ are fixed, $A$ and $\epsilon$ are also pinned down. Appendix 3 describes the details of how we calculate $A$ and $\epsilon$ as a function of $l_0$ and $f_q$.

We will choose $A$ and $\epsilon$ for our test runs by choosing various combinations of $f_q$ and $l_0$, and then using the corresponding $A$ and $\epsilon$. Table 3 shows the resulting values of $A$ and $\epsilon$. For all these values, $F_q$ (the actual fraction of characters queried) is very close to $f_q$ (the desired fraction), regardless of $n$; this is illustrated in Appendix 3.

The rationale for choosing $A$ and $\epsilon$ like this is as follows. $F_q \approx f_q$ is a straightforward measure of the resource usage of EstOPL. If $f_q$ and $n$ are fixed, then the behavior of the algorithm can only vary along one remaining dimension, that of small $l_0$ vs. large $l_0$, since fixing $l_0$ fixes the rest of the parameters. So if we set $f_q = \frac{3}{4}$ and let $l_0$ range over some reasonable selection of positive integers, and run the algorithm
with the resulting list of parameters, we end up trying roughly all the ways the algorithm can behave while querying $\sim 75\%$ of its input. This should let us draw somewhat reliable conclusions about what the algorithm can and cannot accomplish for a given level of resource usage.

We choose $\{2, 8, 32, 128, 512\}$ for the range of values for $l_0$ to vary over for the following reasons. 2 is included because it is the smallest allowed value. 512 is the highest value because for higher values of $l_0$ the behavior of the algorithm becomes somewhat degenerate: there are a large number iterations, on each of which only a tiny fraction of the substrings of the input are sampled. This seems unlikely to yield valuable results. The numbers in the middle are skewed towards the smaller end because of the aforementioned degenerate behavior with larger $l_0$ and because, as evident from Table 3, larger $l_0$ corresponds to a larger multiplicative error bound $A$ and a smaller additive error bound $\epsilon$, and with numbers of the size seen in Table 3, limiting the multiplicative error $A$ makes more of a difference in the bounds “$OPL(w)/A - \epsilon n$” and “$OPL(w)A + \epsilon n$” than making the additive error $\epsilon$ even tinier.

For $f_q$ we choose the values $\{4, 2, \frac{3}{4}, \frac{1}{8}, \frac{1}{32}\}$. We try several values less than 1 because “sublinearity” is a unique feature of EstOPL – no other algorithm can provide any information about optimal parsing length while accessing only a part of the string. We try the larger values to see how the algorithm does when its resource usage is nearer that of existing algorithms that compute the exact length of an optimal parsing.

It is evident from Table 3 that the attainable approximation guarantees are quite weak, that is, the error bounds are wide. Nevertheless, it is conceivable that in practice, the algorithm returns usefully accurate results. The results of our tests will reveal whether or not this is the case.

### 6.3 Results

Figure 1 to Figure 5 illustrate the results of all runs of EstOPL. (The caption of Figure 1 explains the meaning of the graph shown in it; the graphs in Figures 2 to 5 have the same structure.) See Appendix 4 for tables containing the complete results of all runs and some extra details about the test setup.

A first observation to make about the results is that they are all inside the range $[OPL(w)/A - \epsilon n, OPL(w)A + \epsilon n]$. (In Proposition 4.5, we proved that the output of EstOPL is in this range with probability $\geq \frac{2}{3}$.)

Also, the results of repeated runs with the same inputs vary very little; there is very little spread. For $f_q \in \{\frac{3}{4}, \frac{1}{8}, \frac{1}{32}\}$, EstOPL was run three times$^{11,12}$, and for example, for enwik8 with $f_q = \frac{3}{4}$ and $l_0 = 32$, the estimates produced were 1,194,250,

$^{11}$Every such location in Figures 1-5 technically contain three dots instead of one, but they are so close to each other that they are indistinguishable.

$^{12}$For $f_q \in \{2, 4\}$, the algorithm was run only once due to time constraints, and because extra runs would probably not provide valuable new information since the spread would likely be as
Figure 1: The results of running EstOPL on enwik8. The vertical axis is the estimate of $OPL(w)$ produced by EstOPL, as a fraction of $n$, the size of the input file in bytes. The dashed red line is the exact optimal parsing length of ENWI8K, i.e. the number those estimates are approximating. Each location on the horizontal axis corresponds to a combination of $f_q$ and $l_0$. The numbers below the horizontal axis indicate values of $l_0$; and the texts along the top of the graph, together with the vertical dashed lines, indicate values of $f_q$. The leftmost three results with $l_0 = 2$ are so disproportionately high that they are left out of the graph; the same convention is followed for Figures 2-5. (See Appendix 4 for the full tables of results.)

Figure 2: Results for einstein.en.txt.

1,192,630 and 1,193,401.

Clearly, the estimates are wildly inaccurate, despite falling within the provided error bounds and having little random variance. The estimates are not totally unconnected from the correct number; files with shorter optimal parsings generally yield smaller estimates.

The algorithm does not consistently over- or underestimate. The estimates are mostly overestimates for the two files with the smallest parsing (einstein.en.txt and kernel) and mostly underestimates for the other three files.

small as with $f_q \in \left\{\frac{3}{4}, \frac{1}{5}, \frac{1}{32}\right\}$. 
Figure 3: Results for kernel.

Figure 4: Results for random100.

Figure 5: Results for almostuniform.

An obvious pattern is that the estimates decrease as $l_0$ increases (this only fails with einstein.en.txt with the larger $f_q$). The estimates also decrease as $f_q$ decreases. Whether this makes the estimate better or worse depends on whether it was an overestimate to begin with. There is no consistently best value of $l_0$ and, more surprisingly, no best value of $f_q$. So allowing the algorithm extra resources did not
consistently result in better estimates.

6.4 Analysis of results

Given the inaccuracy of the estimates it produces, EstOPL seems unpromising for practical use.

The reason that the results have so little random variation is that the estimates of $d_1,d_2..d_{l_0}$, where $d_l$ stands for the number of distinct substrings of length $l$ in the file, turn out to have very little variation themselves. (Recall that EstOPL computes estimates of $d_l$ for each $l \in \{1..l_0\}$ and derives its guess for $OPL(w)$ from these $d_l$-estimates.)

For example, with $f_q = \frac{3}{4}$ and $l_0 = 32$, we have $r = 19$ and $B \approx 90.0$, so a total of 19 separate 90-estimates of $d_l$ are computed for each $l \in \{1..32\}$; this is done by sampling $\lceil 10N/B^2 \rceil$ length-32 substrings 19 times. With the file enwik8, 19 samples of $\lceil 10N/B^2 \rceil = 123,355$ length-32 substrings are taken, and in one run, the number of distinct length-30 prefixes in a sample ranged from 121,623 at the lowest to 121,807 at the highest, meaning that the 90-estimates for $d_{30}$ ranged from 10,950,585 to 10,967,152 – a very narrow range\(^\text{13}\). (In reality, $d_{30} = 93,077,979$, meaning that the estimates were consistently biased downwards.)

This suggests that we could get results of equal quality by just setting $r$ to 1 instead of to $\text{AMPCount} \left( \frac{3}{4}, 1 - \frac{1}{32} \right)$, cutting the runtime and the number of characters queried to a fraction (the peak memory usage would remain the same).

6.5 Sources of inaccuracy in estimating optimal parsing length

In general, when trying to estimate $OPL(w)$ the way EstOPL does, there are two sources of inaccuracy (i.e., uncertainty):

1. The error of the $B$-estimates of $d_l$.

2. The distance between the lower and upper bound for $OPL(w)$ that can be derived from (estimates of) $d_1,d_2..d_{l_0}$ via Proposition 4.1.

To conclude this section, we will briefly examine both factors in isolation.

Regarding (1), Table 4 gives the multiplicative errors obtained when estimating the number of distinct substrings of length 30 in our test files. To obtain a $B$-estimate of $d_{30}$, we sample $\lceil (n - 29)/B^2 \rceil$ substrings of length 30 and return $B$ times the number of unique elements in the sample as our estimate – as in SSDISTEst in Section 4.5. By Proposition 4.2, the ratio of this estimate and the correct answer is

\(^{13}\)For most runs, the intermediate $d_l$ were not recorded.
Table 4: The multiplicative errors obtained when computing $B$-estimates of $d_{30}$ for our test files using the strategy of SSDISTEST in Section 4.5. This is done as part of EstOPL. Three values of $B$ were tested: 10, 50 and 100. The number $5.14$ in a cell means that the estimate produced was $5.14$ times greater than the true number; $\frac{1}{474}$ means that the estimate was $\frac{1}{474}$ times the true number. In each case, the multiplicative error is guaranteed to be between $\frac{1}{B}$ and $B$ with probability $\geq \frac{3}{4}$.

The exact true number is given in the second column.

<table>
<thead>
<tr>
<th>file</th>
<th>$d_{30}$</th>
<th>$B=10$</th>
<th>$B=50$</th>
<th>$B=100$</th>
</tr>
</thead>
<tbody>
<tr>
<td>enwik8</td>
<td>93,077,979</td>
<td>1.01</td>
<td>1.74</td>
<td>9.43</td>
</tr>
<tr>
<td>einstein.en.txt</td>
<td>796,947</td>
<td>5.14</td>
<td>14.9</td>
<td>20.0</td>
</tr>
<tr>
<td>kernel</td>
<td>8,650,393</td>
<td>7.97</td>
<td>5.25</td>
<td>2.85</td>
</tr>
<tr>
<td>random100</td>
<td>99,999,971</td>
<td>1.05</td>
<td>1.01</td>
<td>10.0</td>
</tr>
<tr>
<td>almostuniform</td>
<td>3,395,822</td>
<td>1.02</td>
<td>1.09</td>
<td>1.17</td>
</tr>
</tbody>
</table>

Table 5: Approximate bounds from Proposition 4.1 for the optimal parsing lengths of the test files, with $l_0 = 128$. The second column gives the actual parsing length; the third and fourth columns give the lower and upper bounds from Proposition 4.1. The “±” error ranges are calculated assuming a maximum 2% error for the $5 \cdot 127$ different $d_l$-estimates from which the bounds are derived. (The $d_l$-estimates were computed using the HyperLogLog algorithm, as described in the main text.) The rightmost column gives the value of $l$ at which $m = \max_{l=1}^{l_0} d_l/l$ attains its maximum value (according to the estimates).

<table>
<thead>
<tr>
<th>file</th>
<th>true OPL</th>
<th>OPL lower</th>
<th>OPL upper</th>
<th>at $l =$</th>
</tr>
</thead>
<tbody>
<tr>
<td>enwik8</td>
<td>8,220,688</td>
<td>4,585,566 ± 2.0%</td>
<td>92,122,212 ± 1.9%</td>
<td>16</td>
</tr>
<tr>
<td>einstein.en.txt</td>
<td>89,467</td>
<td>42,778 ± 2.0%</td>
<td>15,443,574 ± 0.1%</td>
<td>10</td>
</tr>
<tr>
<td>kernel</td>
<td>793,915</td>
<td>406,587 ± 2.0%</td>
<td>15,952,390 ± 1.0%</td>
<td>14</td>
</tr>
<tr>
<td>random100</td>
<td>35,484,830</td>
<td>25,034,280 ± 2.0%</td>
<td>488,993,342 ± 2.0%</td>
<td>4</td>
</tr>
<tr>
<td>almostuniform</td>
<td>959,042</td>
<td>282,550 ± 2.0%</td>
<td>8,608,772 ± 1.3%</td>
<td>128</td>
</tr>
</tbody>
</table>

at least $1/B$ and at most $B$ with probability $\geq \frac{3}{4}$. Table 4 suggests that in practice the ratio varies in a narrower range, and whether it is an over- or underestimate depends on the file.

Regarding (2), recall that Proposition 4.1 allows us to derive lower and upper bounds for $OPL(w)$ from $d_1, d_2 \ldots d_{l_0}$. Namely, denoting $m = \max_{l=1}^{l_0} d_l/l$, we have $m \leq OPL(w) \leq 4 \left( m \log l_0 + \frac{n}{l_0} \right)$.

Table 5 gives (close approximations of) the lower and upper bounds for $OPL(w)$ implied by Proposition 4.1 with $l_0 = 128$, for each of the test files. Calculating these bounds exactly would have involved computing the exact values of each of $d_1, d_2 \ldots d_{128}$. Due to resource constraints, we instead computed close approximations of the $d_l$, and calculated the bounds from these approximations.

To compute the close approximations of the $d_l$, we used the HyperLogLog algorithm [FFGea07]. HyperLogLog is an algorithm for estimating the number of distinct
informative about
It is notable that the lower bound
are shown in Table 5.

HyperLogLog uses very little memory while producing accurate estimates of can apparently be expected to be near the lower bound from Proposition 4.1, and
1
correct answer, with a standard deviation of approximately OPL
Overall, the results of Tables 4 and 5 together suggest that any attempt to estimate the implicit lower bound.

We may recall that the way EstOPL uses these bounds is as follows: At the end of the execution of EstOPL, we have B-estimates \( \hat{d}_1, \hat{d}_2, \ldots, \hat{d}_0 \) and we know that \( \hat{m} = \max_{i=1}^{l_0} \frac{d_i}{l_0} \) is a B-estimate of \( m = \max_{i=1}^{l_0} \frac{d_i}{l_0} \) with probability \( \geq 2/3 \). Then we know that \( \hat{m}/B \leq OPL(w) \leq 4 \left( \hat{m}B \log l_0 + \frac{n}{l_0} \right) \) with probability \( \geq 2/3 \). We may call \( \hat{m}/B \) the implicit lower bound and \( 4 \left( \hat{m}B \log l_0 + \frac{n}{l_0} \right) \) the implicit upper bound for \( OPL(w) \) computed by EstOPL. The estimate we return is \( \hat{m}A_5 + \epsilon n \) (which is indeed always between the implicit lower and upper bounds). The tables in Appendix 4 list the numerical values of the implicit bounds for all of the test runs. As an example, in one run of enwik8 with \( f_q = \frac{3}{4} \) and \( l_0 = 32 \), we have the implicit bounds \( \hat{m}/B = 3505 \) and \( 4 \left( \hat{m}B \log l_0 + \frac{n}{l_0} \right) = 406,423,049 \). Meanwhile, the estimate returned is \( \hat{m}A_5 + \epsilon n = 1,194,250 \) and the actual optimal parsing length is 8,220,688. In general, the implicit bounds are very wide and the estimate is nearer the implicit lower bound.

Overall, the results of Tables 4 and 5 together suggest that any attempt to estimate \( OPL(w) \) using a similar approach to EstOPL can be expected to be substantially inaccurate.

Table 5 does suggest that a decent rough estimate of the optimal parsing length can be obtained using very little memory (as noted in the introduction, memory is the major bottleneck in computing an LZ77 parsing). This is because (1) \( OPL(w) \) can apparently be expected to be near the lower bound from Proposition 4.1, and (2) HyperLogLog uses very little memory while producing accurate estimates of \( d_l \).

The estimates produced by HyperLogLog are approximately normally distributed around the correct answer, with a standard deviation of approximately \( n \sqrt{1/2} \), where \( n \) is the number of distinct elements and \( m \) is “the number of registers”, \( 2^{16} \) in our case [FFGea07]. (The number of registers \( m \) is a tunable parameter of HyperLogLog. The higher \( m \) is, the more accurate the estimates are and the more memory HLL consumes. \( 2^{16} \) was the maximum value for \( m \) selectable with the implementation of HLL we used.)

Assuming a perfect normal distribution, \( \sim 99.9999426696856 \% \) of all estimates are within five standard deviations of the true value, that is, within \( 5n\sqrt{1/2} = 5n\sqrt{1/256} = 0.020n \) of the true value – that is, within 2.0%. And with probability \( \sim 0.999999426696856^{5 \cdot 2^{127}} \geq 99.97 \% \), all of the \( d_l \)-estimates are within 2.0% of the true value.
from which the lower bound can be calculated. For example when computing the $d_{l}$-estimates on which Table 5 is based, HyperLogLog required only $\sim 5.2$ megabytes of memory. (Note, however, that computing the aforementioned lower bound this way is not particularly fast. This is because HyperLogLog involves feeding every input element through a hash function once. Thus, to calculate the bounds in Table 5, a hash function was called $\sim 127$ times for every byte in the input).

7Conclusions and future work

We examined an algorithm, due to Raskhodnikova et al. [RRRS13], that estimates the optimal parsing length of a string in sublinear time.

In Section 5, we described in detail a theoretical setup where the algorithm can be used to obtain nontrivial information about the parsing lengths of strings while running in sublinear time, something that no previous algorithm is capable of. Namely, we showed that for any $\varepsilon > 0$ and $\alpha$ such that $0 < \alpha < 1$, the algorithm can be used to distinguish strings with optimal parsing length $O(n^\alpha)$ from strings with optimal parsing length $\Omega(n)$ in sublinear time $O(n^{\alpha + \varepsilon})$, in a sense that is made precise in Section 5. A compressed outline of the argument in Section 5 was given already by Raskhodnikova et al. [RRRS13].

We ran experiments, described in Section 6, to evaluate the quality of the algorithm’s estimates against both files containing real-world data and ones containing artificial data. We ran experiments under settings where the algorithm queried only a fraction of the characters in the input file, and also under settings where the number of characters queried was greater than the number of characters in the input. We compared the estimates produced by the algorithm to the actual optimal parsing lengths.

The results of the experiments indicate that the algorithm’s estimates are highly inaccurate. It appears that the algorithm as described is of mainly theoretical significance; it is not a promising practical tool for estimating the optimal parsing lengths of files, or, indirectly, their compressibility.

The algorithm has identical space, time and query complexity; thus its memory usage is proportional to its runtime. Also, the algorithm is based on estimating the number of distinct substrings of different lengths in the input. In Section 6.5, we observed that by applying a well-known algorithm named HyperLogLog, it is possible to closely estimate these distinct substring counts using very little memory, on the order of a few megabytes (though the CPU time required is high, albeit linear in the input size.) Using the connection between distinct substring counts and optimal

\[ \text{HyperLogLog uses approximately } 5m \text{ bits of memory, where } m \text{ is the “number of registers” (described in the preceding footnote). So with } m = 2^{16}, \text{ and estimating all substrings lengths } 1..128 \text{ in parallel in one pass, the amount of memory needed is approximately } 5 \cdot 2^{16} \cdot 127 = 41,615,360 \text{ bits, } \approx 5.2 \text{ megabytes.} \]

Note that with HyperLogLog, the amount of memory required grows very slowly with the input size, and is practically independent of it.
parsing length, which was discovered by Raskhodnikova et al. [RRRS13], and on which the main algorithm is based, it is possible to derive lower and upper bounds on the optimal parsing length. Of these bounds, the lower bound turned out to be close to the actual parsing length for all of our test files. If this empirical fact turns out to also be true for a wider range of data, it appears to be possible to obtain informative estimates of optimal parsing size using very little memory.

Future work might further examine the empirical connection between optimal parsing length and the numbers of distinct substrings of different lengths. In particular, it would be straightforward to check whether the optimal parsing length is close to the aforementioned lower bound for a wider variety of real-world data. In general, since distinct substring counts can be estimated closely using very little memory, even for extremely large inputs, the possibility of extracting useful information from them is of some interest.

References


Appendix 1. A simple binary encoding for parsings

This appendix describes one way to encode a parsing (as defined in Section 3) as a sequence of bits. The encoding described here is used in Section 3.2 to assert an upper bound for the size of an LZ77-compressed file.

We are given a parsing \( p \) for a string \( w \) of length \( n \) over the alphabet \( \Sigma \). We construct the binary encoding of \( p \) as follows:

1. Insert the \( A_{\Sigma} \) bits required to indicate that the alphabet being used is \( \Sigma \).

2. Insert a positive integer equal to the number of bits required to encode any number in the range \( 0 \) to \( n - 1 \), that is \( \lceil \log_2 n \rceil \). Encode this positive integer using some self-delimiting binary encoding, so that we can tell where it ends and the main body of the parsing begins. Using Elias gamma coding [Eli75], this takes
\[
2\lceil\log_2\lceil\log_2 n\rceil\rceil - 1 \text{ bits.}
\]

3. Insert the encoding of each of the \( p_1p_2..p_m \) in order. Each \( p_i \) is encoded as a pair of non-negative integers \( K_i, L_i \), where \( K_i \) is \( \lceil \log_2 \max(n, |\Sigma|) \rceil \) bits and \( L_i \) is \( \lceil \log_2 n \rceil \) bits.

   If \( p \) is a pair \( k,l \), just set \( K_i = k \) and \( L_i = l \). Note that \( k \) is always in \( \{0,1..n-1\} \) and \( l \) in \( \{1,2..n-1\} \), so both fit into \( \lceil \log_2 n \rceil \) bits; and if \( p_i \) is a pair, \( L_i \) is not 0.

   If \( p_i \) is a symbol \( s \), set \( L_i \) to 0 and \( K_i \) to some binary encoding of \( s \). Since \( K_i \) is at most \( \lceil \log_2 |\Sigma| \rceil \) bits, the encoding will fit.

This will result in a sequence of bits whose length is exactly
\[
A_{\Sigma} + 2\lceil\log_2\lceil\log_2 n\rceil\rceil - 1 + |p|(\lceil \log_2 n \rceil + \lceil \log_2 \max(n, |\Sigma|) \rceil).
\]
Appendix 2. Proof of Lemma 4.3

In this appendix, we provide a proof of the following Lemma, which is used in Section 4.5.

**Lemma.** Given a list $Q = q_1, q_2, \ldots, q_n$ of length $n$ with $c$ distinct elements $e_1, e_2, \ldots, e_c$, sampling $s \in \{1..n\}$ elements from $Q$ with replacement yields at least $\frac{1}{10} \cdot \frac{c}{n} s$ distinct elements with probability $\geq \frac{3}{4}$.

**Proof.** For $i \in \{1..c\}$, let $\text{Count}(e_i)$ be the number of occurrences of $e_i$ in $Q$. A randomly selected element from $Q$ is $e_i$ with probability $\frac{\text{Count}(e_i)}{n}$. The probability that $e_i$ is included in a sample of $s$ elements is $1 - (1 - \frac{\text{Count}(e_i)}{n})^s$.

Let $X_i$ be a random variable that is 1 if $e_i$ is included in the sample of $s$ elements and 0 otherwise. Now

$$E[X_i] = P[X_i = 1] \geq 1 - (1 - \frac{\text{Count}(e_i)}{n})^s$$

$$\geq 1 - (1 - 1/n)^s \geq 1 - e^{-s/n}.$$

Now, for all $x \in [0, 1]$, it is the case that $1 - e^{-x} \leq (1 - e^{-1})x$. So

$$1 - e^{-s/n} \geq (1 - e^{-1})(s/n),$$

and thus finally

$$E[X_i] \geq (1 - e^{-1})(s/n).$$

Now $X = \sum_{i=1}^c X_i$ is a random variable for the number of distinct elements in a sample of $s$ elements. It now suffices to prove that $P[X > \frac{1}{10}(c/n)s] \geq \frac{3}{4}$. To begin, note that

$$E[X] = \sum_{i=1}^c E[X_i] \geq c(1 - e^{-1})(s/n) = (1 - e^{-1})(c/n)s$$

$$\approx 0.63(c/n)s.$$

In what follows, we require this fact: $\text{Var}(X) < E[X]$. To see that it is true, first observe the following:

- For all $i, j \in \{1..c\}$ with $i \neq j$, the covariance $\text{Cov}(X_i, X_j) = E[X_i X_j] - E[X_i]E[X_j]$ is negative. This is because $e_i$ being included in the sample makes it less likely that $e_j$ was also included.

- For all $i \in \{1..c\}$, $\text{Var}(X_i) \leq E[X_i]$. This is because $X_i$ takes only the values 0 and 1, so $X_i^2 = X_i$, so $\text{Var}(X_i) = E[X_i^2] - E[X_i]^2 = E[X_i] - E[X_i]^2 \leq E[X_i]$. 
And now, as desired:

\[
\text{Var}(X) = \sum_{i=1}^{c} \sum_{j=1}^{c} \text{Cov}(X_i, X_j) = \sum_{i \in \{1, \ldots, c\}} \sum_{j \in \{1, \ldots, c\}, j \neq i} \text{Cov}(X_i, X_j) \\
< \sum_{i \in \{1, \ldots, c\}} \text{Var}(X_i) \\
\leq \sum_{i=1}^{c} E[X_i] = E[X].
\]

Chebyshev's inequality states that for all \( k > 0 \),

\[
P[|X - E[X]| > k] \leq \frac{\text{Var}(X)}{k^2}.
\]

Use it to find an upper bound for the probability that the number of distinct elements in the sample is less than a fraction \( \delta \) of the expectation \( E[X] \):

\[
P[X < \delta E[X]] \leq P[|E[X] - X| > (1 - \delta)E[X]]
\]

\[
\leq \frac{\text{Var}(X)}{( (1 - \delta)E[X] )^2} = \frac{\text{Var}(X)}{E[X]} \cdot \frac{1}{(1 - \delta)^2 E[X]}
\]

\[
\leq \frac{1}{(1 - \delta)^2 E[X]}.
\]

Now consider the numbers

\[
\delta_0 = 3 - \sqrt{8} \approx 0.17
\]

and \( \frac{4}{(1 - \delta_0)^2} \approx 5.8. \)

Either \( E[X] \geq \frac{4}{(1 - \delta_0)^2} \) or \( E[X] < \frac{4}{(1 - \delta_0)^2} \). We will consider the cases separately.

For the first case, \( E[X] \geq \frac{4}{(1 - \delta_0)^2} \), note that

\[
\delta_0 E[X] \geq \delta_0 (1 - e^{-1}) (c/n)s
\]

\[
= (3 - \sqrt{8}) (1 - e^{-1}) (c/n)s
\]

\[
\approx 0.11 (c/n)s
\]

\[
> \frac{1}{10} (c/n)s.
\]

And also,

\[
P[X < \delta_0 E[X]] \leq \frac{1}{(1 - \delta)^2 E[X]}
\]

\[
\leq \frac{1}{(1 - \delta_0)^2} \frac{4}{(1 - \delta_0)^2} = \frac{1}{4}.
\]
Since $\frac{1}{10}(c/n)s \leq \delta_0 E[X]$, also $P[X < \frac{1}{10}(c/n)s] \leq P[X < \delta_0 E[X]]$. And so in the first case,

$$P \left[ X < \frac{1}{10}(c/n)s \right] \leq \frac{1}{4}$$

as desired.

For the second case, $E[X] < \frac{4}{(1-\delta_0)^2}$, first observe that

$$\frac{1}{10}(c/n)s < \delta_0 (1 - e^{-1})(c/n)s \leq \delta_0 E[X] < \delta_0 \frac{4}{(1-\delta_0)^2}.$$ 

The number $\delta_0 \frac{4}{(1-\delta_0)^2} = \frac{4(3-\sqrt{8})}{(1-(3-\sqrt{8}))^2} \approx 1.0$ is slightly smaller than 1. Thus in the second case,

$$\frac{1}{10}(c/n)s < 1.$$

Since at least one distinct element is always included in the sample, $X$ is always at least 1. So, trivially, $P[X > \frac{1}{10}(c/n)s] \geq \frac{\delta}{4}$ in the second case as well.

$\square$
Appendix 3. Determining $A$ and $\epsilon$ from $l_0$ and $f_q$

This appendix describes the details of how we calculate $A$ and $\epsilon$ from $f_q$ and $l_0$, as discussed in Section 6.2. We first go over the general procedure, then illustrate it with an example.

As noted in the main text, the fraction of the input queried by EstOPL is

$$F_q = \left(\text{AmpCount} \left(\frac{3}{4}, 1 - \frac{1}{3l_0}\right) \cdot \left[\frac{40(n - l_0 + 1) \log l_0}{A^2}\right] \cdot l_0\right) / n.$$

With $l_0$ and $f_q$ fixed, we want to choose $A$ and $\epsilon$ so that $F_q$ is as close to $f_q$ as possible.

To begin, remove the ceiling function from the expression on the right, and rearrange to solve for $A$:

$$F_q = \left(\text{AmpCount} \left(\frac{3}{4}, 1 - \frac{1}{3l_0}\right) \cdot \frac{40(n - l_0 + 1) \log l_0}{A^2} \cdot l_0\right) / n\n\Leftrightarrow A^2 = \left(\text{AmpCount} \left(\frac{3}{4}, 1 - \frac{1}{3l_0}\right) \cdot 40 \left(1 - \frac{l_0 + 1}{n}\right) \log l_0 \cdot l_0\right) / F_q\n\Leftrightarrow A = \sqrt{\left(\text{AmpCount} \left(\frac{3}{4}, 1 - \frac{1}{3l_0}\right) \cdot 40 \left(1 - \frac{l_0 + 1}{n}\right) \log l_0 \cdot l_0\right) / F_q}. \quad (12)$$

So now we can get a value for $A$ as a function of $F_q$, $l_0$ and $n$.

Note that $\frac{l_0 + 1}{n}$ is generally negligibly small, at least with the values of $l_0$ and $n$ used in our experiments (where $l_0$ is at most 512, $n$ is at least 100,000,000). So the dependence on $n$ is also negligibly small. So $A$ depends mostly on $F_q$ and $l_0$. For our purposes, we use $n = 100,000,000$ in (12).

Given $A$ and $l_0$, we can calculate a value for $\epsilon$ using the following fact:

$$l_0 = \left\lceil 2/(A\epsilon) \right\rceil \Leftrightarrow 2/(A\epsilon) \leq l_0 < 2/(A\epsilon) + 1 \Leftrightarrow 2/(l_0A) \leq \epsilon < 2/(A(l_0 - 1)).$$

So we get a narrow range for $\epsilon$. Arbitrarily, we set

$$\epsilon = \frac{1}{10}2/(A l_0) + \frac{9}{10}2/(A(l_0 - 1)).$$

So, for example, for $f_q = \frac{3}{4}$ and $l_0 = 32$ we get $A = 335.235$ and $\epsilon = 0.000191849$.

Then with $n = 100,000,000$, $F_q$, the actual fraction queried, becomes 0.749998, which is very close to $f_q$ as desired. With $n = 450,000,000$ and the same $A$ and $\epsilon$, $F_q$ becomes 0.749999 – negligibly different and also very close to $f_q$. 


Table 6: The test files.

<table>
<thead>
<tr>
<th>name</th>
<th>size (bytes)</th>
<th>OPL</th>
<th>as %</th>
</tr>
</thead>
<tbody>
<tr>
<td>enwik8</td>
<td>100,000,000</td>
<td>8,220,688</td>
<td>8.22%</td>
</tr>
<tr>
<td>einstein.en.txt</td>
<td>467,626,544</td>
<td>89,467</td>
<td>0.02%</td>
</tr>
<tr>
<td>kernel</td>
<td>257,961,616</td>
<td>793,915</td>
<td>0.31%</td>
</tr>
<tr>
<td>random100</td>
<td>100,000,000</td>
<td>35,484,830</td>
<td>35.48%</td>
</tr>
<tr>
<td>almostuniform</td>
<td>100,000,000</td>
<td>959,042</td>
<td>0.96%</td>
</tr>
</tbody>
</table>

Appendix 4. Full results of test runs

This appendix contains the full tables of results for the test runs of EstOPL. These results were discussed in Section 6.3.

The files used are as described in Section 6.1; Table 6 lists them again. The parameters $A$ and $\epsilon$ are set as described in Section 6.2.

As noted in the main text, the tests were run using a Python implementation of EstOPL which is available at https://github.com/oneb/estcompr. The machine on which the tests were run had an 2.60Ghz Intel Core i5-7300U (Dual Core, 3M Cache) CPU and 8GB RAM. The runtimes of the runs were between 2 seconds and 2 hours. We do not include these runtimes in the tables and do not discuss them elsewhere because our implementation of EstOPL is not strongly optimized; we expect that the runtimes could be cut heavily with an optimized implementation of EstOPL.

The columns of the tables below are as follows:

- $f_q$ is the target for the number of characters queried during the execution of the algorithm, as a fraction of the size of the file. $A$ and $\epsilon$ are determined from $f_q$ and $l_0$ as described in Section 6.2. The actual fraction of characters queried differed slightly from $f_q$, but the difference was never greater than 0.13% (and we believe even this was mostly an artifact caused by premature rounding in our test setup.)

The values of $f_q$ used are $4, 2, \frac{3}{4}, \frac{1}{5}, \frac{1}{32}$.

- $l_0$ is the value of the internal parameter of the same name in EstOPL. (It is the maximum substring length for which the number of distinct substrings of that length is estimated.) The values of $l_0$ used are $2, 8, 32, 128, 512$.

- $A$ and $\epsilon$ are the approximation parameters we provide to EstOPL, computed from $f_q$ and $l_0$. Recall that with probability $\geq 2/3$, the output of EstOPL is between $\text{OPL}(w)/A - \epsilon n$ and $\text{OPL}(w)A + \epsilon n$, as proved in Proposition 4.5. Note that in our tests, the output was always between these bounds.

- %low and %high are the lower and upper bounds $\text{OPL}(w)/A - \epsilon n$ and $\text{OPL}(w)A + \epsilon n$ as a percentage of $n$, the size of the file in bytes. Note that
the formula for the lower bound gives a negative result in some instances. In practice, this naturally just means 0.

- \# is the number of the run. For \( f_q \) of \( \frac{3}{4}, \frac{1}{8} \) and \( \frac{1}{32} \) we ran EstOPL three times for each parameter setting, so that we can see how the output randomly varies. For \( f_q \) of 2 and 4 we ran EstOPL only one time because of how time-consuming these runs were on our test machine. (Given how consistently small the spread of the estimate is with smaller \( f_q \), additional runs would likely produce nearby estimates.)

- \( \text{est} \) is the output of EstOPL.

- \%\( n \) is the output as a percentage of the size of the input file.

- \%\( \text{real} \) is the output as a percentage of the correct answer, i.e. \( \text{OPL}(w) \), the exact optimal parsing length of the file.

- \( \text{ilb} \), for “implicit lower bound”, is the numerical value of the expression \( \hat{m}/B \) at the end of the execution of EstOPL. (Recall that at the end of EstOPL, we know that with probability \( \geq 2/3 \), \( \hat{m}/B \leq \text{OPL}(w) \leq 4 \left( \hat{m}B \log l_0 + \frac{n}{\hat{m}} \right) \). These implicit bounds are discussed in Section 6.5.

- \( \text{iub} \), for “implicit upper bound”, is the corresponding upper bound, i.e. \( 4 \left( \hat{m}B \log l_0 + \frac{n}{\hat{m}} \right) \).
<table>
<thead>
<tr>
<th>$f_q$</th>
<th>$l_0$</th>
<th>$A$</th>
<th>$\epsilon$</th>
<th>%low</th>
<th>%high</th>
<th>#</th>
<th>est</th>
<th>%n</th>
<th>%real</th>
<th>ilb</th>
<th>iub</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6.45</td>
<td>0.2946</td>
<td>-28.19%</td>
<td>82.48%</td>
<td>29.45077</td>
<td>29.50%</td>
<td>358.79%</td>
<td>5.098</td>
<td>200.212.025</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>42.78</td>
<td>0.006596</td>
<td>-0.41%</td>
<td>352.30%</td>
<td>4.473,483</td>
<td>4.47%</td>
<td>54.42%</td>
<td>89.157</td>
<td>213.149.710</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>145.16</td>
<td>0.0004431</td>
<td>0.01%</td>
<td>1,193.00%</td>
<td>2,181,519</td>
<td>2.18%</td>
<td>26.54%</td>
<td>14.723</td>
<td>322.740.062</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>409.5</td>
<td>0.00003843</td>
<td>0.02%</td>
<td>3,366.00%</td>
<td>1,150,458</td>
<td>1.15%</td>
<td>13.99%</td>
<td>2.800</td>
<td>472.658.303</td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>1087.1</td>
<td>0.0000036</td>
<td>0.01%</td>
<td>8937.00%</td>
<td>659,459</td>
<td>0.66%</td>
<td>8.02%</td>
<td>0.606</td>
<td>717.287.891</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9.12</td>
<td>0.2083</td>
<td>-19.93%</td>
<td>95.81%</td>
<td>20.875,761</td>
<td>20.88%</td>
<td>253.94%</td>
<td>4.700</td>
<td>200.390.900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>60.5</td>
<td>0.004664</td>
<td>-0.33%</td>
<td>497.80%</td>
<td>3,625,380</td>
<td>3.63%</td>
<td>44.10%</td>
<td>52.218</td>
<td>241.108.180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>205.29</td>
<td>0.0003133</td>
<td>0.01%</td>
<td>1,688.00%</td>
<td>1,696,394</td>
<td>1.70%</td>
<td>20.64%</td>
<td>8.111</td>
<td>354.317.843</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>579.11</td>
<td>0.00002717</td>
<td>0.01%</td>
<td>4761.00%</td>
<td>914,526</td>
<td>0.91%</td>
<td>11.12%</td>
<td>1.574</td>
<td>531.165.138</td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>1537.39</td>
<td>0.000002545</td>
<td>0.01%</td>
<td>12640.00%</td>
<td>529,001</td>
<td>0.53%</td>
<td>6.43%</td>
<td>344</td>
<td>813.670.509</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_q$</td>
<td>$l_0$</td>
<td>$\lambda$</td>
<td>$\epsilon$</td>
<td>%low</td>
<td>%high</td>
<td>#</td>
<td>est</td>
<td>%n</td>
<td>%real</td>
<td>ilb</td>
<td>iub</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>14.89</td>
<td>0.1276</td>
<td>-12.21%</td>
<td>135.20%</td>
<td>1</td>
<td>12,817,701</td>
<td>12.82%</td>
<td>155.92%</td>
<td>4,042</td>
<td>200,896,587</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>12,817,674</td>
<td>12.82%</td>
<td>155.92%</td>
<td>4,040</td>
<td>200,896,188</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>12,817,630</td>
<td>12.82%</td>
<td>155.92%</td>
<td>4,037</td>
<td>200,895,522</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>98.79</td>
<td>0.002856</td>
<td>-0.20%</td>
<td>812.40%</td>
<td>1</td>
<td>2,612,437</td>
<td>2.61%</td>
<td>31.78%</td>
<td>23,553</td>
<td>279,986,856</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2,612,565</td>
<td>2.61%</td>
<td>31.78%</td>
<td>23,555</td>
<td>279,988,154</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2,613,130</td>
<td>2.61%</td>
<td>31.79%</td>
<td>23,560</td>
<td>279,987,302</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>335.24</td>
<td>0.0001918</td>
<td>0.01%</td>
<td>2756.00%</td>
<td>1</td>
<td>1,194,250</td>
<td>1.19%</td>
<td>14.53%</td>
<td>3,505</td>
<td>406,423,049</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,192,630</td>
<td>1.19%</td>
<td>14.51%</td>
<td>3,500</td>
<td>405,879,744</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1,193,401</td>
<td>1.19%</td>
<td>14.52%</td>
<td>3,503</td>
<td>406,138,461</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>945.69</td>
<td>0.00001664</td>
<td>0.01%</td>
<td>7774.00%</td>
<td>1</td>
<td>668,410</td>
<td>0.67%</td>
<td>8.13%</td>
<td>705</td>
<td>633,658,741</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>669,289</td>
<td>0.67%</td>
<td>8.14%</td>
<td>706</td>
<td>634,470,756</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>667,981</td>
<td>0.67%</td>
<td>8.13%</td>
<td>705</td>
<td>633,252,733</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>2510.55</td>
<td>0.000001559</td>
<td>0.00%</td>
<td>20640.00%</td>
<td>1</td>
<td>387,147</td>
<td>0.39%</td>
<td>4.71%</td>
<td>154</td>
<td>972,342,311</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>389,660</td>
<td>0.39%</td>
<td>4.74%</td>
<td>155</td>
<td>978,651,068</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>387,398</td>
<td>0.39%</td>
<td>4.71%</td>
<td>154</td>
<td>972,973,187</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>241.99</td>
<td>0.001166</td>
<td>-0.08%</td>
<td>1989.00%</td>
<td>1</td>
<td>1,354,374</td>
<td>1.35%</td>
<td>16.48%</td>
<td>5,115</td>
<td>349,525,119</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,353,319</td>
<td>1.35%</td>
<td>16.46%</td>
<td>5,111</td>
<td>349,269,871</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1,354,745</td>
<td>1.35%</td>
<td>16.48%</td>
<td>5,117</td>
<td>349,615,036</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>821.15</td>
<td>0.00007832</td>
<td>0.00%</td>
<td>6750.00%</td>
<td>1</td>
<td>662,192</td>
<td>0.66%</td>
<td>8.06%</td>
<td>797</td>
<td>549,830,527</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>662,325</td>
<td>0.66%</td>
<td>8.06%</td>
<td>797</td>
<td>549,939,188</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>662,104</td>
<td>0.66%</td>
<td>8.05%</td>
<td>797</td>
<td>549,758,087</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>2316.45</td>
<td>0.000006793</td>
<td>0.00%</td>
<td>19040.00%</td>
<td>1</td>
<td>376,246</td>
<td>0.38%</td>
<td>4.58%</td>
<td>162</td>
<td>873,105,380</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>377,034</td>
<td>0.38%</td>
<td>4.59%</td>
<td>162</td>
<td>874,932,408</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>375,457</td>
<td>0.38%</td>
<td>4.57%</td>
<td>162</td>
<td>871,808,939</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>6149.57</td>
<td>0.000006363</td>
<td>0.00%</td>
<td>50550.00%</td>
<td>1</td>
<td>220,429</td>
<td>0.22%</td>
<td>2.68%</td>
<td>36</td>
<td>1,355,934,446</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>222,481</td>
<td>0.22%</td>
<td>2.71%</td>
<td>36</td>
<td>1,368,551,974</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>220,429</td>
<td>0.22%</td>
<td>2.68%</td>
<td>36</td>
<td>1,355,934,446</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>72.96</td>
<td>0.02604</td>
<td>-2.49%</td>
<td>602.40%</td>
<td>1</td>
<td>2,729,791</td>
<td>2.73%</td>
<td>33.21%</td>
<td>1,722</td>
<td>209,169,138</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2,731,369</td>
<td>2.73%</td>
<td>33.23%</td>
<td>1,744</td>
<td>209,284,230</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2,730,536</td>
<td>2.73%</td>
<td>33.22%</td>
<td>1,733</td>
<td>209,223,487</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>483.97</td>
<td>0.000583</td>
<td>-0.04%</td>
<td>3979.00%</td>
<td>1</td>
<td>833,849</td>
<td>0.83%</td>
<td>10.14%</td>
<td>1,602</td>
<td>425,345,392</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>835,861</td>
<td>0.84%</td>
<td>10.11%</td>
<td>1,607</td>
<td>426,318,419</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>835,056</td>
<td>0.84%</td>
<td>10.16%</td>
<td>1,605</td>
<td>425,928,588</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1642.31</td>
<td>0.00003916</td>
<td>0.00%</td>
<td>13500.00%</td>
<td>1</td>
<td>428,910</td>
<td>0.43%</td>
<td>5.22%</td>
<td>259</td>
<td>710,653,213</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>428,359</td>
<td>0.43%</td>
<td>5.21%</td>
<td>258</td>
<td>709,566,603</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>428,910</td>
<td>0.43%</td>
<td>5.22%</td>
<td>259</td>
<td>710,472,111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>4632.91</td>
<td>0.000003397</td>
<td>0.00%</td>
<td>38090.00%</td>
<td>1</td>
<td>152,688</td>
<td>0.15%</td>
<td>1.86%</td>
<td>12</td>
<td>1,878,318,601</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>153,919</td>
<td>0.15%</td>
<td>1.87%</td>
<td>13</td>
<td>1,893,495,355</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>150,226</td>
<td>0.15%</td>
<td>1.83%</td>
<td>12</td>
<td>1,848,036,731</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_q$</td>
<td>$b_0$</td>
<td>$A$</td>
<td>$\epsilon$</td>
<td>%low</td>
<td>%high</td>
<td>$\neq$</td>
<td>%est</td>
<td>$\neq$</td>
<td>%real</td>
<td>$\neq$</td>
<td>ilb</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------</td>
<td>---------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>2</td>
<td>6.45</td>
<td>0.2946</td>
<td>-29.46%</td>
<td>29.59%</td>
<td>1</td>
<td>137,781,104</td>
<td>29.46%</td>
<td>154002.15%</td>
<td>1.246</td>
<td>935,304,913</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>42.78</td>
<td>0.000596</td>
<td>-0.66%</td>
<td>1.48%</td>
<td>1</td>
<td>3,384,777</td>
<td>0.72%</td>
<td>3783.27%</td>
<td>7.025</td>
<td>246,667,864</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>145.16</td>
<td>0.0004431</td>
<td>-0.04%</td>
<td>2.82%</td>
<td>1</td>
<td>827,631</td>
<td>0.18%</td>
<td>925.07%</td>
<td>4.274</td>
<td>148,517,920</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>409.5</td>
<td>0.00003843</td>
<td>0.00%</td>
<td>7.84%</td>
<td>1</td>
<td>1,112,532</td>
<td>0.24%</td>
<td>1243.96%</td>
<td>2.674</td>
<td>462,995,040</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>1087.1</td>
<td>0.0000036</td>
<td>0.00%</td>
<td>20.80%</td>
<td>1</td>
<td>1,351,233</td>
<td>0.29%</td>
<td>1510.31%</td>
<td>1.241</td>
<td>1,470,748,746</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9.12</td>
<td>0.2083</td>
<td>-20.83%</td>
<td>21.01%</td>
<td>1</td>
<td>97,430,825</td>
<td>20.84%</td>
<td>108901.41%</td>
<td>1.168</td>
<td>935,350,262</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>60.5</td>
<td>0.004664</td>
<td>-0.47%</td>
<td>1.62%</td>
<td>1</td>
<td>2,568,206</td>
<td>0.55%</td>
<td>2870.56%</td>
<td>6.402</td>
<td>257,243,150</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>205.29</td>
<td>0.0003133</td>
<td>-0.03%</td>
<td>3.96%</td>
<td>1</td>
<td>943,677</td>
<td>0.20%</td>
<td>1054.78%</td>
<td>3.883</td>
<td>222,103,750</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>579.11</td>
<td>0.00002717</td>
<td>0.00%</td>
<td>11.08%</td>
<td>1</td>
<td>1,276,860</td>
<td>0.27%</td>
<td>1427.19%</td>
<td>2.183</td>
<td>746,701,027</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>1537.39</td>
<td>0.000002545</td>
<td>0.00%</td>
<td>29.41%</td>
<td>1</td>
<td>1,282,629</td>
<td>0.27%</td>
<td>1433.63%</td>
<td>834</td>
<td>1,973,723,839</td>
<td></td>
</tr>
<tr>
<td>$f_q$</td>
<td>$l_0$</td>
<td>$A$</td>
<td>$\epsilon$</td>
<td>%low</td>
<td>%high</td>
<td>#</td>
<td>est</td>
<td>%n</td>
<td>%real</td>
<td>lib</td>
<td>iub</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>---</td>
<td>-----</td>
<td>-----</td>
<td>--------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>2</td>
<td>14.89</td>
<td>0.1276</td>
<td>-12.76%</td>
<td>13.04%</td>
<td>1</td>
<td>59,673,280</td>
<td>12.76%</td>
<td>66698.65%</td>
<td>1.963</td>
<td>935,488,759</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>98.79</td>
<td>0.002856</td>
<td>-0.29%</td>
<td>2.18%</td>
<td>2</td>
<td>59,673,271</td>
<td>12.76%</td>
<td>66698.64%</td>
<td>1.962</td>
<td>935,488,626</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>335.24</td>
<td>0.0001918</td>
<td>-0.02%</td>
<td>6.43%</td>
<td>3</td>
<td>59,673,285</td>
<td>12.76%</td>
<td>66698.65%</td>
<td>1.963</td>
<td>935,488,826</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>945.69</td>
<td>0.0001664</td>
<td>0.00%</td>
<td>18.09%</td>
<td>1</td>
<td>1,357,982</td>
<td>0.29%</td>
<td>1520.33%</td>
<td>1,428</td>
<td>1,291,481,935</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>2510.55</td>
<td>0.00001559</td>
<td>0.00%</td>
<td>48.03%</td>
<td>1</td>
<td>1,087,311</td>
<td>0.23%</td>
<td>1216.32%</td>
<td>433</td>
<td>2,731,572,238</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>241.99</td>
<td>0.001166</td>
<td>-0.12%</td>
<td>4.75%</td>
<td>1</td>
<td>24,388,796</td>
<td>5.22%</td>
<td>27260.10%</td>
<td>923</td>
<td>936,483,597</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>821.15</td>
<td>0.0007832</td>
<td>-0.01%</td>
<td>15.72%</td>
<td>1</td>
<td>1,385,859</td>
<td>0.30%</td>
<td>1547.90%</td>
<td>1,642</td>
<td>1,165,558,626</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>2316.45</td>
<td>0.00006793</td>
<td>0.00%</td>
<td>44.32%</td>
<td>1</td>
<td>1,068,478</td>
<td>0.23%</td>
<td>1193.21%</td>
<td>458</td>
<td>2,482,330,323</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>6149.57</td>
<td>0.000006363</td>
<td>0.00%</td>
<td>117.70%</td>
<td>1</td>
<td>697,085</td>
<td>0.15%</td>
<td>779.15%</td>
<td>113</td>
<td>4,277,240,690</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>72.96</td>
<td>0.02604</td>
<td>-2.60%</td>
<td>4.00%</td>
<td>1</td>
<td>12,239,867</td>
<td>2.62%</td>
<td>13680.87%</td>
<td>854</td>
<td>939,799,353</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>483.97</td>
<td>0.000583</td>
<td>-0.06%</td>
<td>9.32%</td>
<td>1</td>
<td>1,613,536</td>
<td>0.35%</td>
<td>1803.50%</td>
<td>1,642</td>
<td>1,165,558,626</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1642.31</td>
<td>0.0003916</td>
<td>0.00%</td>
<td>31.42%</td>
<td>1</td>
<td>1,622,593</td>
<td>0.25%</td>
<td>1299.47%</td>
<td>697</td>
<td>1,937,715,674</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>4632.91</td>
<td>0.00003397</td>
<td>0.00%</td>
<td>88.64%</td>
<td>1</td>
<td>767,704</td>
<td>0.16%</td>
<td>858.09%</td>
<td>165</td>
<td>3,563,956,852</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>12299.1</td>
<td>0.000003182</td>
<td>0.00%</td>
<td>235.30%</td>
<td>1</td>
<td>478,626</td>
<td>0.10%</td>
<td>534.97%</td>
<td>39</td>
<td>5,888,491,421</td>
<td></td>
</tr>
<tr>
<td>$f_q$</td>
<td>$l_0$</td>
<td>$A$</td>
<td>$\epsilon$</td>
<td>%low</td>
<td>%high</td>
<td>#</td>
<td>est</td>
<td>%tile</td>
<td>%real</td>
<td>ilb</td>
<td>iub</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>---------</td>
<td>-------------</td>
<td>------</td>
<td>------</td>
<td>---</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>2</td>
<td>6.45</td>
<td>0.2946</td>
<td>-29.41%</td>
<td>31.45%</td>
<td>1</td>
<td>76,014,930</td>
<td>29.47%</td>
<td>9574.69%</td>
<td>2,134</td>
<td>516,011,985</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>42.78</td>
<td>0.006596</td>
<td>-0.65%</td>
<td>13.82%</td>
<td>1</td>
<td>5,225,925</td>
<td>2.03%</td>
<td>658.25%</td>
<td>82.392</td>
<td>279,750,331</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>145.16</td>
<td>0.0004431</td>
<td>-0.04%</td>
<td>44.72%</td>
<td>1</td>
<td>3,714,632</td>
<td>1.44%</td>
<td>467.89%</td>
<td>24.802</td>
<td>554,874,237</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>409.5</td>
<td>0.0003843</td>
<td>0.00%</td>
<td>126.00%</td>
<td>1</td>
<td>2,310,390</td>
<td>0.90%</td>
<td>291.01%</td>
<td>5.618</td>
<td>950,095,034</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>1087.1</td>
<td>0.000036</td>
<td>0.00%</td>
<td>334.60%</td>
<td>1</td>
<td>1,377,314</td>
<td>0.53%</td>
<td>173.48%</td>
<td>1.266</td>
<td>1,498,284,230</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9.12</td>
<td>0.2083</td>
<td>-20.80%</td>
<td>23.64%</td>
<td>1</td>
<td>53,760,307</td>
<td>20.84%</td>
<td>6771.54%</td>
<td>2,129</td>
<td>516,100,357</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>60.5</td>
<td>0.004664</td>
<td>-0.46%</td>
<td>19.09%</td>
<td>1</td>
<td>5,018,411</td>
<td>1.95%</td>
<td>632.11%</td>
<td>63.067</td>
<td>359,794,664</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>205.29</td>
<td>0.003133</td>
<td>-0.03%</td>
<td>63.21%</td>
<td>1</td>
<td>3,138,575</td>
<td>1.22%</td>
<td>395.33%</td>
<td>14.895</td>
<td>659,966,323</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>579.11</td>
<td>0.0002717</td>
<td>0.00%</td>
<td>178.20%</td>
<td>1</td>
<td>1,875,469</td>
<td>0.73%</td>
<td>236.23%</td>
<td>3.226</td>
<td>1,090,110,791</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>1537.39</td>
<td>0.00002545</td>
<td>0.00%</td>
<td>473.20%</td>
<td>1</td>
<td>1,117,728</td>
<td>0.43%</td>
<td>140.79%</td>
<td>727</td>
<td>1,719,389,118</td>
<td></td>
</tr>
<tr>
<td>$f_q$</td>
<td>$l_0$</td>
<td>$A$</td>
<td>$\epsilon$</td>
<td>%low</td>
<td>%high</td>
<td>%est</td>
<td>%real</td>
<td>ilb</td>
<td>iub</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>14.89</td>
<td>0.1276</td>
<td>-12.74%</td>
<td>17.34%</td>
<td>32,940,957</td>
<td>12.77%</td>
<td>4149.18%</td>
<td>2.115</td>
<td>516,392,419</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>32,940,966</td>
<td>12.77%</td>
<td>4149.18%</td>
<td>2.116</td>
<td>516,392,552</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>32,940,961</td>
<td>12.77%</td>
<td>4149.18%</td>
<td>2.116</td>
<td>516,392,485</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>98.79</td>
<td>0.002856</td>
<td>-0.28%</td>
<td>30.69%</td>
<td>32,940,957</td>
<td>12.77%</td>
<td>4149.18%</td>
<td>36.78</td>
<td>487,819,147</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>335.24</td>
<td>0.0001918</td>
<td>-0.02%</td>
<td>103.20%</td>
<td>2,390,073</td>
<td>0.93%</td>
<td>301.09%</td>
<td>6,983</td>
<td>817,015,637</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>945.69</td>
<td>0.00001664</td>
<td>0.00%</td>
<td>291.10%</td>
<td>1,390,673</td>
<td>0.54%</td>
<td>175.17%</td>
<td>1,466</td>
<td>1,319,144,734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>2510.55</td>
<td>0.00001559</td>
<td>0.00%</td>
<td>772.70%</td>
<td>1,390,673</td>
<td>0.54%</td>
<td>175.17%</td>
<td>1,466</td>
<td>1,319,144,734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>36.48</td>
<td>0.05208</td>
<td>-5.20%</td>
<td>16.44%</td>
<td>1,390,673</td>
<td>0.54%</td>
<td>175.17%</td>
<td>1,466</td>
<td>1,319,144,734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>241.99</td>
<td>0.001166</td>
<td>-0.12%</td>
<td>74.59%</td>
<td>1,390,673</td>
<td>0.54%</td>
<td>175.17%</td>
<td>1,466</td>
<td>1,319,144,734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>821.15</td>
<td>0.00007832</td>
<td>-0.01%</td>
<td>252.70%</td>
<td>1,390,673</td>
<td>0.54%</td>
<td>175.17%</td>
<td>1,466</td>
<td>1,319,144,734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>2316.45</td>
<td>0.000006793</td>
<td>0.00%</td>
<td>712.90%</td>
<td>1,390,673</td>
<td>0.54%</td>
<td>175.17%</td>
<td>1,466</td>
<td>1,319,144,734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>6149.57</td>
<td>0.000006363</td>
<td>0.00%</td>
<td>1893.00%</td>
<td>1,390,673</td>
<td>0.54%</td>
<td>175.17%</td>
<td>1,466</td>
<td>1,319,144,734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>72.96</td>
<td>0.02604</td>
<td>-2.60%</td>
<td>25.06%</td>
<td>1,390,673</td>
<td>0.54%</td>
<td>175.17%</td>
<td>1,466</td>
<td>1,319,144,734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>483.97</td>
<td>0.0000583</td>
<td>-0.06%</td>
<td>149.00%</td>
<td>1,390,673</td>
<td>0.54%</td>
<td>175.17%</td>
<td>1,466</td>
<td>1,319,144,734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1642.31</td>
<td>0.00003916</td>
<td>0.00%</td>
<td>505.40%</td>
<td>1,390,673</td>
<td>0.54%</td>
<td>175.17%</td>
<td>1,466</td>
<td>1,319,144,734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>4632.91</td>
<td>0.000003397</td>
<td>0.00%</td>
<td>1426.00%</td>
<td>1,390,673</td>
<td>0.54%</td>
<td>175.17%</td>
<td>1,466</td>
<td>1,319,144,734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>12299.1</td>
<td>0.000003182</td>
<td>0.00%</td>
<td>3785.00%</td>
<td>1,390,673</td>
<td>0.54%</td>
<td>175.17%</td>
<td>1,466</td>
<td>1,319,144,734</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### random100 (large $f_q$)

<table>
<thead>
<tr>
<th>$f_q$</th>
<th>$l_0$</th>
<th>$A$</th>
<th>$\epsilon$</th>
<th>%low</th>
<th>%high</th>
<th>#</th>
<th>est</th>
<th>%n</th>
<th>%real</th>
<th>ilb</th>
<th>iub</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6.45</td>
<td>0.2946</td>
<td>-23.96%</td>
<td>258.30%</td>
<td>1</td>
<td>29,586,932</td>
<td>29.59%</td>
<td>83.38%</td>
<td>19,679</td>
<td>20,818,714</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>42.78</td>
<td>0.006596</td>
<td>0.17%</td>
<td>1541.0%</td>
<td>1</td>
<td>19,961,063</td>
<td>19.96%</td>
<td>56.25%</td>
<td>451,180</td>
<td>875,716,578</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>145.16</td>
<td>0.0004431</td>
<td>0.29%</td>
<td>5151.00%</td>
<td>1</td>
<td>8,402,220</td>
<td>8.40%</td>
<td>23.68%</td>
<td>57,577</td>
<td>1,225,743,172</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>409.5</td>
<td>0.00003843</td>
<td>0.08%</td>
<td>14530.00%</td>
<td>1</td>
<td>3,575,388</td>
<td>3.58%</td>
<td>10.08%</td>
<td>8,722</td>
<td>1,465,655,064</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>1087.1</td>
<td>0.0000036</td>
<td>0.03%</td>
<td>38580.00%</td>
<td>1</td>
<td>1,964,307</td>
<td>1.96%</td>
<td>5.54%</td>
<td>1,807</td>
<td>2,135,788,059</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9.12</td>
<td>0.2083</td>
<td>-16.94%</td>
<td>344.50%</td>
<td>1</td>
<td>21,009,476</td>
<td>21.01%</td>
<td>59.21%</td>
<td>19,679</td>
<td>201,636,820</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>60.5</td>
<td>0.004664</td>
<td>0.12%</td>
<td>2147.00%</td>
<td>1</td>
<td>15,169,734</td>
<td>15.17%</td>
<td>42.75%</td>
<td>243,030</td>
<td>939,551,677</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>205.29</td>
<td>0.0003133</td>
<td>0.14%</td>
<td>7285.00%</td>
<td>1</td>
<td>6,008,248</td>
<td>6.01%</td>
<td>16.93%</td>
<td>29,115</td>
<td>1,239,489,862</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>579.11</td>
<td>0.00002717</td>
<td>0.06%</td>
<td>20550.00%</td>
<td>1</td>
<td>2,533,549</td>
<td>2.53%</td>
<td>7.14%</td>
<td>4,370</td>
<td>1,468,762,370</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>1537.39</td>
<td>0.000002545</td>
<td>0.02%</td>
<td>54550.00%</td>
<td>1</td>
<td>1,501,694</td>
<td>1.50%</td>
<td>4.23%</td>
<td>977</td>
<td>2,309,079,259</td>
<td></td>
</tr>
<tr>
<td>$f_q$</td>
<td>$l_0$</td>
<td>$A$</td>
<td>$\epsilon$</td>
<td>$%_{\text{low}}$</td>
<td>$%_{\text{high}}$</td>
<td>$%_{\text{est}}$</td>
<td>$%_{\text{real}}$</td>
<td>$%_{\text{ib}}$</td>
<td>$%_{\text{ib}}$</td>
<td>$%_{\text{ilb}}$</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>14.89</td>
<td>0.1276</td>
<td>-10.37%</td>
<td>541.20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>98.79</td>
<td>0.002856</td>
<td>0.07%</td>
<td>3506.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>335.24</td>
<td>0.0001918</td>
<td>0.09%</td>
<td>11900.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>945.69</td>
<td>0.00001664</td>
<td>0.04%</td>
<td>33560.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>2510.55</td>
<td>0.000001559</td>
<td>0.01%</td>
<td>89090.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>36.48</td>
<td>0.05208</td>
<td>-4.24%</td>
<td>1300.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>241.99</td>
<td>0.001166</td>
<td>0.03%</td>
<td>8587.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>821.15</td>
<td>0.0007832</td>
<td>0.04%</td>
<td>29140.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>2316.45</td>
<td>0.000006793</td>
<td>0.01%</td>
<td>82200.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>6149.57</td>
<td>0.000006363</td>
<td>0.01%</td>
<td>218200.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>72.96</td>
<td>0.02604</td>
<td>-2.12%</td>
<td>2592.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>483.97</td>
<td>0.000583</td>
<td>0.02%</td>
<td>17170.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1642.31</td>
<td>0.00003916</td>
<td>0.02%</td>
<td>58280.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>4632.91</td>
<td>0.000003397</td>
<td>0.01%</td>
<td>164400.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>12299.1</td>
<td>0.000003182</td>
<td>0.00%</td>
<td>436400.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### almostuniform (large $f_q$)

<table>
<thead>
<tr>
<th>$f_q$</th>
<th>$l_0$</th>
<th>$A$</th>
<th>$\epsilon$</th>
<th>%low</th>
<th>%high</th>
<th>#</th>
<th>est</th>
<th>%n</th>
<th>%real</th>
<th>ilb</th>
<th>iub</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6.45</td>
<td>0.2046</td>
<td>-29.31%</td>
<td>35.65%</td>
<td></td>
<td>1</td>
<td>29,469,978</td>
<td>29.47%</td>
<td>3072.86%</td>
<td>1,547</td>
<td>200,064,359</td>
</tr>
<tr>
<td>8</td>
<td>42.78</td>
<td>0.006596</td>
<td>-0.64%</td>
<td>41.69%</td>
<td></td>
<td>1</td>
<td>685,342</td>
<td>0.69%</td>
<td>71.46%</td>
<td>602</td>
<td>51,101,258</td>
</tr>
<tr>
<td>32</td>
<td>145.16</td>
<td>0.0004431</td>
<td>-0.04%</td>
<td>139.30%</td>
<td></td>
<td>1</td>
<td>86,356</td>
<td>0.09%</td>
<td>9.00%</td>
<td>290</td>
<td>18,604,052</td>
</tr>
<tr>
<td>128</td>
<td>409.5</td>
<td>0.00003843</td>
<td>0.00%</td>
<td>392.70%</td>
<td></td>
<td>1</td>
<td>51,618</td>
<td>0.05%</td>
<td>5.38%</td>
<td>117</td>
<td>22,688,671</td>
</tr>
<tr>
<td>512</td>
<td>1087.1</td>
<td>0.0000036</td>
<td>0.00%</td>
<td>1043.00%</td>
<td></td>
<td>1</td>
<td>31,827</td>
<td>0.03%</td>
<td>3.32%</td>
<td>29</td>
<td>34,989,151</td>
</tr>
<tr>
<td>4</td>
<td>9.12</td>
<td>0.2083</td>
<td>-20.72%</td>
<td>29.58%</td>
<td></td>
<td>1</td>
<td>20,838,806</td>
<td>20.84%</td>
<td>2172.88%</td>
<td>966</td>
<td>200,080,312</td>
</tr>
<tr>
<td>8</td>
<td>60.5</td>
<td>0.0004664</td>
<td>-0.45%</td>
<td>58.49%</td>
<td></td>
<td>1</td>
<td>487,260</td>
<td>0.49%</td>
<td>50.81%</td>
<td>345</td>
<td>51,262,011</td>
</tr>
<tr>
<td>32</td>
<td>205.29</td>
<td>0.0003133</td>
<td>-0.03%</td>
<td>196.90%</td>
<td></td>
<td>1</td>
<td>68,162</td>
<td>0.07%</td>
<td>7.11%</td>
<td>179</td>
<td>20,061,145</td>
</tr>
<tr>
<td>128</td>
<td>579.11</td>
<td>0.00002717</td>
<td>0.00%</td>
<td>555.40%</td>
<td></td>
<td>1</td>
<td>42,156</td>
<td>0.04%</td>
<td>4.40%</td>
<td>68</td>
<td>25,964,237</td>
</tr>
<tr>
<td>512</td>
<td>1537.39</td>
<td>0.000002545</td>
<td>0.00%</td>
<td>1474.00%</td>
<td></td>
<td>1</td>
<td>27,343</td>
<td>0.03%</td>
<td>2.85%</td>
<td>18</td>
<td>42,426,636</td>
</tr>
<tr>
<td>( f_q )</td>
<td>( t_0 )</td>
<td>( A )</td>
<td>( \epsilon )</td>
<td>%low</td>
<td>%high</td>
<td># est</td>
<td>( %n )</td>
<td>%real</td>
<td>( \text{llb} )</td>
<td>( \text{ub} )</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>14.89</td>
<td>0.1276</td>
<td>-12.69%</td>
<td>27.04%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>98.79</td>
<td>0.002856</td>
<td>-0.28%</td>
<td>95.03%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>335.24</td>
<td>0.0001918</td>
<td>-0.02%</td>
<td>321.50%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>945.69</td>
<td>0.00001664</td>
<td>0.00%</td>
<td>907.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>2510.55</td>
<td>0.000001559</td>
<td>0.00%</td>
<td>2408.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>36.48</td>
<td>0.05208</td>
<td>-5.18%</td>
<td>40.19%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>241.99</td>
<td>0.001166</td>
<td>-0.11%</td>
<td>232.20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>821.15</td>
<td>0.00007832</td>
<td>-0.01%</td>
<td>787.50%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>2316.45</td>
<td>0.000006793</td>
<td>0.00%</td>
<td>2222.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>6149.57</td>
<td>0.0000006363</td>
<td>0.00%</td>
<td>5898.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>72.96</td>
<td>0.02604</td>
<td>-2.59%</td>
<td>72.58%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>483.97</td>
<td>0.000583</td>
<td>-0.06%</td>
<td>464.20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1642.31</td>
<td>0.00003916</td>
<td>0.00%</td>
<td>1575.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>4632.91</td>
<td>0.00003397</td>
<td>0.00%</td>
<td>4443.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>12299.1</td>
<td>0.000003182</td>
<td>0.00%</td>
<td>11800.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>