Trace reconstruction with constant deletion probability and related results

Thomas Holenstein* Michael Mitzenmacher† Rina Panigrahy‡ Udi Wieder§

Abstract

We provide several new results for the *trace reconstruction* problem. In this setting, a binary string yields a collection of *traces*, where each trace is independently obtained by independently deleting each bit with a fixed probability δ. Each trace therefore consists of a random subsequence of the original sequence. Given the traces, we wish to reconstruct the original string with high probability. The questions are how many traces are necessary for reconstruction, and how efficiently can the reconstruction be performed.

Our primary result is that for some universal constant γ and uniformly chosen strings of length n, for any $\delta < \gamma$ reconstruction is possible with poly(n) traces in poly(n) time with high probability. We also obtain algorithms that require a number of traces exponential in $\tilde{O}(\sqrt{n})$ for any $\delta < 1$ even for worst case strings, and we derive lower bound results for simpler classes of algorithms based on summary statistics from the traces.

1 Introduction

In this paper, we consider the following problem: a binary string $\mathbf{x} = x_1 x_2 \dots x_n$ yields a collection of *traces* $Y^1, Y^2, \dots, Y^m$, where each Y^i is independently obtained from $\mathbf{x}$ by passing through a *deletion channel*, under which each bit is independently deleted with fixed probability δ. Each trace therefore consists of a random subsequence of the original sequence. Given the traces (and the value of n and δ), we wish to reconstruct the *original string* $\mathbf{x}$ exactly with high probability. The question is how many traces are necessary for reconstruction. Our primary new result is the following: for some universal constant γ and uniformly at random chosen $\mathbf{x}$, for $\delta < \gamma$, reconstruction is possible with polynomially many traces with high probability (over both $\mathbf{x}$ and the traces). The polynomial can be taken to be independent of δ, and the reconstruction itself also takes polynomial time. As we clarify below, this represents a substantial advance over previous work.

We also obtain additional results for this setting. We derive an algorithm that works with high probability for *any* original string and constant $\delta < 1$, which requires a number of traces exponential in $\tilde{O}(\sqrt{n})$. Finally, we derive lower bound results for simpler classes of algorithms based on summary statistics from the traces. In particular, we show that for any constant $\delta > 0$, for sufficiently large n there exist two distinct 0-1 strings w^1 and w^2 of length n such that for any j the probability that bit j is 1 in a trace of w^1 is nearly the same as the probability that bit j is 1 in a trace of w^2; in particular, the difference is less than any polynomial. As we argue in Section 3, this demonstrates that natural algorithms based on the L_1 distance between observed and expected summary statistics are doomed to fail.

1.1 Background and previous work

This general class of *trace reconstruction problems* arises naturally in multiple contexts. First, it can be viewed as a purely information theoretic problem. Combinatorial variations have been studied by Levenshtein [7, 8]; in these variations, the question is in the worst case how many distinct subsequences are required to reconstruct the initial string $\mathbf{x}$. That work examines more extensive error models, including insertions and other errors. While Levenshtein also considers the problem of reconstruction over probabilistic channels, he does so only for memoryless channels. The harder case with deletions and/or insertions was not considered.

In this information theoretic context, the trace reconstruction problem is also tangentially related to corresponding coding problems [4, 5]. For example, one could ask for the code capacity when sending a message to multiple cooperating receivers, where the transmission to each receiver behaves as though it passes through an independent deletion channel. Notice that in our setting, our base string $\mathbf{x}$ is *not* chosen from an agreed codebook, but instead chosen at random; the coding problem provides an interesting variation for future work.

*Microsoft Research, Silicon Valley. thomahol@microsoft.com

†Harvard School of Engineering and Applied Sciences, Cambridge, MA. Supported in part by NSF Grant CCR-0634923. michaelm@eecs.harvard.edu

‡Microsoft Research, Silicon Valley. rina@microsoft.com

§Microsoft Research, Silicon Valley. uwieder@microsoft.com

As suggested in [3, 6], trace reconstruction can also be viewed as a restricted version of the multiple sequence alignment problem, a fundamental problem of computational biology. It arises, for example, when one is given a set of related DNA sequences (the Y_i), and one wants to determine the common ancestor from which these sequences might have arisen ($\mathbf{x}$). In our setting, the evolutionary process is taken to be random deletion; of course, other evolutionary processes can be considered, but the problem is already theoretically quite challenging even when only considering deletions. The main result of [3] focuses exactly on the scenario we consider.

Trace reconstruction also appear naturally in the context of sensor networks. An array of sensors could attempt to record a sequence of events, where each event gives a positive or negative result – that is, a 0/1 outcome. However, for various reasons, such as noise or mechanical imperfections, not all sensors may detect each event, giving rise to deletions in each sensor's trace. Reconstructing the correct sequence from a collection of individually inaccurate sensors, specifically sensors that independently miss each event with fixed probability, corresponds to the trace reconstruction problem.

For the specific problems we consider, formal results were obtained by Batu, Kannan, Khanna, and McGregor [3]. They showed that for strings of length n chosen uniformly at random, with $O(\log n)$ traces the original string can be reconstructed with high probability for deletion probabilities $\delta = O(1/\log n)$. They also present further results for arbitrary inputs $\mathbf{x}$, as opposed to randomly chosen $\mathbf{x}$; their analysis shows that with $O(n \log n)$ traces the original string can be reconstructed with high probability for deletion probabilities $\delta = O(1/\sqrt{n})$. The main result of [3] is based on sequential majority voting, which we describe in order to later compare with our approach. Each trace maintains a pointer, representing the current guess as to the next bit in the original string; initially, each pointer points to the first bit of each trace. At each step, the traces take a majority vote of what the next bit is, based upon their pointers. Traces that match the majority vote increment their pointer to point to the next bit; traces that do not match leave their pointers in the same position.

Further results appear in [6], which considers insertions and errors as well as deletions. This work introduces the ideas of *anchors*, or substrings that appear (with possibly some small deviations) in most or all of the traces. We adopt a similar idea in our approaches.

For deletions only, [3] contains the best previous results we are aware of. Specifically, the existence of a reconstruction scheme for constant deletion probabilities has remained open until our work.

As a further remark, we note that all our algorithms have the property that they do not use the fact that the string is of finite length. Thus, they also work in a model where the string and the traces are infinite sequences. In this case, n would just be the last bit the algorithms reconstruct. We note that we do not know of a trivial argument that in this setting trace reconstruction is possible for arbitrary $\delta < 1$, even using arbitrarily many strings, but as mentioned before, our analysis in Section 3 shows that indeed for any δ reconstruction is possible.

2 A polynomial trace algorithm for random bitstring and small deletion probabilities

2.1 Intuition Our algorithm, in contrast to that of [3], does not use majority voting, but a different voting-based scheme. Inherently, one limitation of majority voting is that every trace has an equal vote, regardless of how sure it is of its vote. A belief-based scheme (e.g., belief propagation) would instead utilize votes combined with probabilistic measures of their certainty. As of this point, we do not have the tools to analyze such schemes. Instead, we adopt a different approach. We only let traces vote if we think they are likely to have the right value; we determine this inductively by checking if the trace matches the last $O(\log n)$ bits, so that we are confident that we have an accurate assessment of the trace's current position.

In non-technical terms, John F. Kennedy reportedly said in a speech, "The ignorance of one voter in a democracy impairs the security of all." Similarly, ignorant voters impair voting-based reconstruction algorithms, so we attempt to restrict the vote to appropriately knowledgeable traces. We suspect this general approach may prove useful in developing and analyzing voting-based algorithms in other contexts.

2.2 An exponential algorithm We begin with an algorithm for reconstructing the first h bits of the original string $\mathbf{x}$ of length n using $e^{O(h)}$ traces (and similar time). This algorithm works for any $\mathbf{x}$; we do not assume $\mathbf{x}$ is random here. While such an algorithm is obviously expensive, it allows us to recover the first $O(\log n)$ bits of $\mathbf{x}$, from which we bootstrap our main algorithm. We improve on this algorithm in Section 3, but the algorithm in this section is simpler. Also, we use some results from this section in our polynomial trace algorithm. Further, we remark that no effort has been made to optimize the constants in the analysis that follows.

THEOREM 2.1. *For any $\delta < \frac{1}{3}$, there is an algorithm that determines the first h bits of $\mathbf{x}$ from*

$O(he^{14h\delta}\log(1/\epsilon))$ *independent traces correctly with probability* $1-\epsilon$.

The algorithm determines the bits sequentially. Denoting by Y_j the random variable of the jth position in a trace (where $Y_j \in \{0,1,\perp\}$, with $\perp$ being the value if a trace is shorter than j bits), the main observation is that the value of x_i influences Y_j for $j = i(1-3\delta)$ more than all the subsequent bits $x_{i+1}x_{i+2}\dots x_n$ combined.[1] That is, there exists a threshold S which can be computed given $x_1\dots x_{i-1}$, such that if $x_i = 1$ then $\Pr[Y_j = 1] > S$, and if $x_i = 0$, then $\Pr[Y_j = 1] < S$, regardless of the values of the later bits.

Let $\mathrm{Inf}(i,j)$ denote the probability that x_i ends up as the jth bit in a trace, i.e., the influence of input bit i onto bit j in a trace. We have

$$\mathrm{Inf}(i,j) = \binom{i-1}{j-1}(1-\delta)^j\delta^{i-j}, \tag{2.1}$$

where we keep the dependence on δ implicit throughout. Also, $\Pr[Y_j = 1] = \sum_{i\geq j}\mathrm{Inf}(i,j)x_i$.

The following simple lemma states two properties which we can directly use to reconstruct $\mathbf{x}$ (the lemma is slightly stronger than what we need here, since we reuse the lemma in Section 2.3).

LEMMA 2.1. *Let* $\delta < \frac{1}{3}$ *and* $j \leq (1-3\delta)i$. *Then,* $\mathrm{Inf}(i,j) \geq 2\sum_{i'>i}\mathrm{Inf}(i',j)$. *If* $(1-4\delta)i \leq j \leq (1-3\delta)i$ *then* $\mathrm{Inf}(i,j) \geq e^{-7\delta i}$.

Proof. For the first statement it is sufficient to note that for any $i' > i$ the inequality $\frac{\mathrm{Inf}(i',j)}{\mathrm{Inf}(i'-1,j)} = \frac{\binom{i'}{j}}{\binom{i'-1}{j}}\delta = \frac{i'}{i'-j}\delta < \frac{1}{3}$ holds, since this implies that the series is upper bounded by a geometric series with $q = \frac{1}{3}$. When $(1-4\delta)i \leq j \leq (1-3\delta)i$, the second statement follows from $\mathrm{Inf}(i,j) = \binom{i-1}{i-j}(1-\delta)^j\delta^{i-j} \geq \left(\frac{i-1}{i-j}\right)^{i-j}(1-\delta)^j\delta^{i-j} \geq \left(\frac{1}{4\delta}\right)^{i-j}(1-\delta)^j\delta^{i-j} \geq \left(\frac{1}{4}\right)^{i-j}(1-\delta)^j \geq \left(\frac{1}{4}\right)^{4\delta i}e^{(1-3\delta)\ln(1-\delta)\cdot i} \geq e^{-7\delta i}$. □

Using this, we can prove Theorem 2.1

Proof. (of Theorem 2.1) Assume that the algorithm has already determined $x_1x_2\dots x_{i-1}$. To recover x_i we set $j = (1-3\delta)i$ and write

$$\begin{aligned}\Pr[Y_j = 1] &= \sum_{\ell=j}^{n}\mathrm{Inf}(\ell,j)x_\ell \\ &= \sum_{\ell=j}^{i-1}\mathrm{Inf}(\ell,j)x_\ell + \mathrm{Inf}(i,j)x_i + \\ &\qquad \sum_{\ell=i+1}^{n}\mathrm{Inf}(\ell,j)x_\ell.\end{aligned}$$

From Lemma 2.1 we get $0 \leq \sum_{\ell=i+1}^{n}\mathrm{Inf}(\ell,j)x_\ell \leq \frac{1}{2}\mathrm{Inf}(i,j)$, and since $\sum_{\ell=j}^{i-1}\mathrm{Inf}(\ell,j)x_\ell$ can be computed given $x_1\dots x_{i-1}$ we see that if we know $\Pr[Y_j = 1]$ up to an additive error of $\frac{\mathrm{Inf}(i,j)}{4} \geq \frac{1}{4}e^{-7\delta i}$ we can learn x_i. Using standard Chernoff bounds the theorem follows.□

We note that this implies that for any $\delta < \frac{1}{3}$ and for any constant c the first $c\log n$ bits of $\mathbf{x}$ can be determined from a polynomial number of traces with high probability, where the polynomial is independent of δ (but dependent on c).

2.3 Polynomial traces, random x We now assume that $\mathbf{x}$ is random, and $\delta \leq \gamma$, where γ will be a sufficiently small universal constant.

The essential idea of our algorithm is to designate a substring of length $w = O(\log(n))$ of the already recovered input string as "anchor". We then discard all traces which do not contain the anchor, and use a voting on the remaining strings to predict x_i.

In order for this technique to work, we will need to assume that $\mathbf{x}$ has the property that an anchor substring is unlikely to appear from deletions in another part of the string. This holds for w-substring unique strings, which we define next.

DEFINITION 2.1. *A string* $\mathbf{x}$ *of length* n *is* w-substring unique *if for all* a, b *for which* $\{a,\dots,a+w\} \not\subseteq \{b,\dots,b+1.1w\}$, *the substring* $x_a\dots x_{a+w}$ *cannot be obtained by deleting some symbols in* $x_b\dots x_{b+1.1w}$.

We show later (Lemma 2.4) that most strings of length n are $O(\log(n))$-substring-unique, and we will see that our algorithm recovers all substring-unique strings correctly. We now show that in a substring-unique string anchors help determine from where trace bits arose, in the sense that a match implies that we can get an approximate guess of the origin of the last bit of the match. We remark that an exact guess is impossible in some cases: for example, if the substring ends with more than $1/\delta^2$ zeros (which happens with constant probability).

[1] While x_i has the strongest influence on $Y_{j(1-\delta)}$, the combined effect of the subsequent bits influences this particular output value more. The influence breaks even at $Y_{j(1-2\delta)}$, and any position before that could be used to predict; we chose $j(1-3\delta)$ for simplicity.

LEMMA 2.2. *Let δ be a small enough constant and let $\mathbf{x}$ of length n be w-substring unique. Let Y be a trace of $\mathbf{x}$ after application of a deletion channel with deletion probability δ.*

Condition on the event that there exists a j^* for which

$$Y_{j^*-w} \dots Y_{j^*-1} = x_{i-w} \dots x_{i-1} \tag{2.2}$$

holds, and set j^* to an arbitrary value for which this is true. Then, the probability that Y_{j^*} does not originate from a bit in the range $x_i \dots x_{i+0.1w-1}$ is at most $n\delta^{0.001w}$.

Proof. Let $\mathcal{A}$ be the event that there exists a j^* for which (2.2) holds, and let $\mathcal{B}$ the the event that Y_{j^*} originates from outside the given range. We have to upper bound $\Pr[\mathcal{B}|\mathcal{A}] = \frac{\Pr[\mathcal{A}\wedge\mathcal{B}]}{\Pr[\mathcal{A}]} \leq \frac{\Pr[\mathcal{B}]}{\Pr[\mathcal{A}]}$. We first observe that $\Pr[\mathcal{A}] \geq (1-\delta)^w$, since with this probability none of the symbols $x_{i-w} \dots x_{i-1}$ is deleted.

Because $\mathbf{x}$ is w-substring unique, Y_{j^*} can only come from outside the range $x_{i-1} \dots x_{i-1+0.1w}$ if more than $0.1w$ symbols of $x_a \dots x_{a+1.1w}$ are deleted for some a. The probability that for a fixed a more than $0.1w$ symbols in this sequence are deleted is at most, using a Chernoff bound, $\delta^{(1.1w)(0.05^2)} \leq \delta^{0.002w}$ and thus the probability that this happens at any position a is at most $\Pr[\mathcal{B}] \leq n\delta^{0.002w}$. In total we get $\Pr[\mathcal{B}|\mathcal{A}] = \frac{\Pr[\mathcal{A}\wedge\mathcal{B}]}{\Pr[\mathcal{A}]} \leq \frac{\Pr[\mathcal{B}]}{\Pr[\mathcal{A}]} \leq \frac{n\delta^{0.002w}}{(1-\delta)^w} \leq n\delta^{0.001w}$. □

In the simplest approach, we would now use the substring $x_{i-w}x_{i-w+1} \dots x_{i-1}$ of width w (with $w \in O(\log n)$) as anchor, and only consider traces containing this substring. Then, if in a particular trace Y there exists a j^* such that $Y_{j^*-w}Y_{j^*-w+1} \dots Y_{j^*-1} = x_{i-w}x_{i-w+1} \dots x_{i-1}$, it seems reasonable to hope that Y_{j^*} would yield a good prediction for x_i to use in our voting.

However, this is not always the case: again, if the last $\approx 1/\delta^2$ bits $x_{i-1/\delta^2} \dots x_i$ (including x_i) are all zero (note that this happens with constant probability), then it is likely that at least one of these zeros was deleted in traces which match $x_{i-w}x_{i-w+1} \dots x_{i-1}$, and with a bit of additional work one can obtain a counterexample which contradicts the above hope. We avoid this problem by searching for an anchor of the form $x_{i-w-v} \dots x_{i-v-1}$ for some $v \in O(\log(n))$ in the trace, and using a position which is slightly after the match of the anchor to predict x_i (cf. Figure 1).

The algorithm used to successively find the next bit is given in Algorithm 1. The first $O(\log n)$ bits are found using the approach of Theorem 2.1. After that, the algorithm first discards all strings which do not match a certain anchor substring of the previously recovered string, and then counts how many of the remaining strings have a one in a certain position.

LEMMA 2.3. *Let δ be a small enough constant and let $\mathbf{x}$ be of length n. Let w, v be as in Algorithm 1, and assume $\mathbf{x}$ is w-substring unique. Let Y be a trace of $\mathbf{x}$ after application of a deletion channel with deletion probability δ. Condition on the event that $Y \in$ matching, and let j^* be the smallest position in which Y matches.*

Then, there exists a polynomial time computable threshold T such that,

$$x_i = 1 \implies \Pr[Y_{j^*+(1-3\delta)(v-0.1w)} = 1] \geq T + \frac{1}{n^{1100}},$$
$$x_i = 0 \implies \Pr[Y_{j^*+(1-3\delta)(v-0.1w)} = 1] \leq T - \frac{1}{n^{1100}}$$

Proof. Let J^* be the random variable corresponding to j^* and I^* be the random variable which denotes the position from which Y_{j^*} originates.

We get, for any $j \geq 0$, and after conditioning on a match,

$$\Pr[Y_{j^*+j} = 1] = \sum_{i^* \geq 0} \Pr[I^* = i^*] \sum_{\ell \geq 0} \mathrm{Inf}(\ell, j) x_{i^*+\ell} \tag{2.3}$$

According to Lemma 2.2, we have $\Pr\big[i - v \leq I^* \leq i - v + 0.1w - 1\big] \geq 1 - n\delta^{0.001w}$. We can therefore restrict the outer sum in (2.3) to this range with negligible loss. In other words, there exists a $A_i(\mathbf{x}) \in [0, n\delta^{0.0001w}] \subset [0, \frac{n}{100^w}]$ such that

$$\begin{aligned}
&\Pr[Y_{j^*+j} = 1] \\
&\quad = A_i(\mathbf{x}) + \sum_{i^*=i-v}^{i-v+0.1w} \Pr[I^* = i^*] \sum_{\ell \geq 0} \mathrm{Inf}(\ell, j) x_{i^*+\ell}, \\
&\quad = A_i(\mathbf{x}) + \sum_{i^*=i-v}^{i-v+0.1w} \Pr[I^* = i^*] \sum_{\ell \geq i^*}^{n} \mathrm{Inf}(\ell - i^*, j) x_\ell \\
&\quad = A_i(\mathbf{x}) + \sum_{i^*=i-v}^{i-v+0.1w} \Pr[I^* = i^*] \Big(\sum_{\ell=i^*}^{i-1} \mathrm{Inf}(\ell - i^*, j) x_\ell \\
&\qquad\qquad + \mathrm{Inf}(i - i^*, j) x_i \\
&\qquad\qquad + \sum_{\ell \geq i+1} \mathrm{Inf}(\ell - i^*, j) x_\ell \Big)
\end{aligned}$$

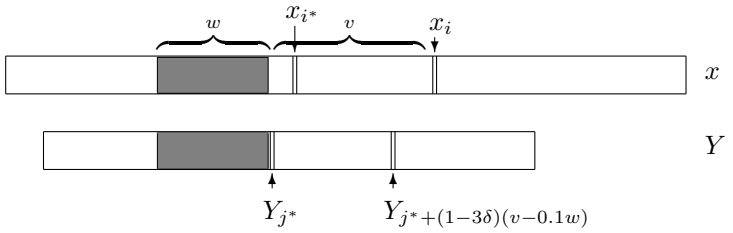

Figure 1: The initial string $\mathbf{x}$ and a trace Y which matches at position j^*. The bit x_{i^*} is the actual origin of Y_{j^*}, while $Y_{j^*+(1-3\delta)(v-0.1w)}$ is used to predict x_i.

1 **procedure** ObtainNextBit($x_1 \dots x_{i-1}$, $Y^1, \dots, Y^{\text{poly}(n)}$)
2 $w := 100 \log(n)$
3 $v := \frac{w}{\delta}$
4 **if** $i \le v + w$ **then**
5 Use Theorem 2.1 to predict x_i
6 **else**
7 matching $:= \{Y \in \{Y^1, \dots, Y^{\text{poly}(n)}\} \mid \exists j^* : (Y_{j^*-v} \dots Y_{j^*-1}) = (x_{i-v-w} \dots x_{i-v-1})\}$
8 ones $:= \{Y \in \text{matching} \mid Y_{j^*+(1-3\delta)(v-0.1w)} = 1\}$ where j^* is the smallest j^* as in Line (7)
9 **return** $\left(\frac{|\text{ones}|}{|\text{matching}|} > T\right)$, where T is as in Lemma 2.3
10 **fi**
11 **od**

Algorithm 1: The polynomial time algorithm which finds x_i.

$$= A_i(\mathbf{x}) + \underbrace{\sum_{i^*=i-v}^{i-v+0.1w} \Pr[I^* = i^*] \sum_{\ell=i^*}^{i-1} \text{Inf}(\ell - i^*, j) x_\ell}_{=:S} + \sum_{i^*=i-v}^{i-v+0.1w} \Pr[I^* = i^*] \Big(\text{Inf}(i - i^*, j) x_i + \sum_{\ell \ge i+1} \text{Inf}(\ell - i^*, j) x_\ell \Big) \tag{2.4}$$

For $j = (v - 0.1w)(1 - 3\delta)$ we see that $j \le (i - i^*)(1 - 3\delta)$ for all $i^* \in [i - v, i - v + 0.1w]$, and thus we can use the first part of Lemma 2.1 which implies that $\text{Inf}(i - i^*, j) \ge 2 \sum_{\ell \ge i+1} \text{Inf}(\ell - i^*, j)$.

Thus, in case $x_i = 0$ equation (2.4) implies

$$\Pr[Y_j = 1] \le A_i(\mathbf{x}) + S + \sum_{r=i-v}^{i-v+0.1w} \Pr[I^* = i^*] \frac{\text{Inf}(i - r, j)}{2}.$$

On the other hand, if $x_i = 1$ we have

$$\Pr[Y_j = 1] \ge A_i(\mathbf{x}) + S + \sum_{r=i-v}^{i-v+0.1w} \Pr[I^* = i^*] \text{Inf}(i - r, j).$$

Now, the difference between $A_i(\mathbf{x})$ when $x_i = 0$ and $A_i(\mathbf{x})$ when $x_i = 1$ is at most $n \frac{1}{100^w}$. Furthermore, given that $x_0 \dots x_{i-1}$ are known, S can be calculated up to an accuracy of $(\frac{1}{100})^w$ in polynomial time. Also, for all $i^* \in [i - v, i - v + 0.1w]$, using Lemma 2.1: $\text{Inf}(i - i^*, j) \ge \exp(-7\delta(i - i*)) \ge \exp(-7\delta v) = \exp(-7w) \ge n^{-1099}$. Finally, it is possible to compute $\Pr[I^* = i^*]$, and since $\sum_{i^*=i-v}^{i-v+0.1w} \Pr[I^* = i^*] \in 1 - o(1)$, the gap between the two cases is at least $\frac{1}{n^{1100}}$. □

THEOREM 2.2. *Let δ be a small enough constant, and let $\mathbf{x}$ be a $100 \log(n)$-substring unique string of length n. There exists a polynomial time algorithm which recovers $\mathbf{x}$ from $\text{poly}(n)$ independent traces with probability $1 - o(1)$.*

Proof. We will predict the bits sequentially, using the procedure ObtainNextBit as given in Algorithm 1. For every call to ObtainNextBit we use a fresh set of $\text{poly}(n)$ traces (to make sure the errors are independent). Starting with a high enough polynomial, and using a Chernoff bound, we see that with very high probability we have enough traces in the set matching in Line (7). Using Lemma 2.3 we see that the next bit will predicted correctly with overwhelming probability. The theorem follows. □

We conclude by showing that most strings are substring-unique, which implies that our algorithm works for most strings.

LEMMA 2.4. *At least a fraction* $1 - \frac{1}{n^d}$ *of all strings of length* n *are* $(2d+4)\log(n)$*-substring unique.*

Proof. Consider two intervals $x_a \dots x_{a+w}$ and $x_b \dots x_{b+1.1w}$ for which $\{a, \dots, a+w\} \not\subseteq \{b, \dots, b+1.1w\}$; w.l.o.g. we consider the case $b > a$. Since for a fixed deletion pattern, one can choose the string $\mathbf{x}$ by selecting bits in increasing order, the probability that $x_a \dots x_{a+w}$ can be obtained by deleting symbols in $x_b \dots x_{b+1.1w}$ is at most $\binom{1.1w}{0.1w} 2^{-w} \leq (11e)^{0.1w} 2^{-w} \leq (1.415)^{-w}$. Since there are fewer than n^2 disjoint intervals, we get that the probability that two contradicting substrings exist is at most $n^2(1.415)^{-w} \leq n^{-d}$. □

COROLLARY 2.1. *Let* δ *be a small enough constant. There exists a polynomial time algorithm which recovers fraction* $1 - n^{-48}$ *of all strings* $\mathbf{x}$ *correctly with probability* $1 - o(1)$ *from* $\mathrm{poly}(n)$ *independent traces.*

Proof. According to Lemma 2.4 this large fraction of the strings of length n are $100\log(n)$-substring unique, and we apply Theorem 2.2. □

Clearly, the error probability of the algorithm can be made arbitrarily small by running it multiple times on independent traces.

3 Algorithms Based on the Probability Vectors

The algorithm from Theorem 2.1 reconstructs each bit x_i based on an estimate of $p_j := \Pr[Y_j = 1]$ for $j = (1-3\delta)i$. It seems natural to ask how much one can generalize this approach; in particular, how good does an approximation $\hat{p}_1 \dots \hat{p}_n$ of $p_1 \dots p_n$ need to be such that recovering $x_1 \dots x_n$ is possible.[2] Approximate values $\hat{p}_j$ are obtained with the desired accuracy by counting the number of of ones in position j in many independent traces; Chernoff bounds control the quality of the approximation.[3]

Without loss of generality, we assume that we try to recover the original string bitwise, starting from the front. Thus, we have already recovered bits $x_1 \dots x_{i-1}$, and we want to recover x_i. We can formulate the following integer linear program (3.5), where the p_j and the $x_{i'}$ for $i' \geq i$ are variables (the p_j are variables denoting the actual probabilities $\Pr[Y_j = 1]$, while the $x_{i'}$ are placeholders for the yet to be determined values of x), the $\hat{p}_j$ and the $x_{i'}$ for $i' < i$ are constants (these $x_{i'}$ are the already determined values of $\mathbf{x}$, the $\hat{p}_j$ are the estimates of $\Pr[Y_j = 1]$). The objective function is not linear, but could be easily replaced by a linear function by introducing additional variables a_j with $a_j \geq p_j - \hat{p}_j$ and $a_j \geq -(p_j - \hat{p}_j)$, and minimizing $\sum a_j$, we omit this replacement in the presentation for simplicity.

(3.5)

$$\begin{aligned} \text{Minimize} \quad & \sum_{j=1}^{n} |p_j - \hat{p}_j| \\ \text{Subject to} \quad & p_j = \sum_{i'=j}^{n} \mathrm{Inf}(i', j) x_{i'} \quad (j = 1, \dots, n) \\ & x_{i'} \in \{0, 1\} \quad (i' = i, \dots, n) \end{aligned}$$

Actually, solving this ILP recovers the entire string for $i = 1$ already. However, it is unclear how to solve it efficiently, even if the estimates $\hat{p}_j$ are exact. Thus, we consider the following relaxation:

(3.6)

$$\begin{aligned} \text{Minimize} \quad & \sum_{j=1}^{n} |p_j - \hat{p}_j| \\ \text{Subject to} \quad & p_j = \sum_{i'=j}^{n} \mathrm{Inf}(i', j) x_{i'} \quad (j = 1, \dots, n) \\ & x_i \in \{0, 1\} \\ & x_{i'} \in [0, 1] \quad (i' = i+1, \dots, n) \end{aligned}$$

Clearly, this second ILP can be solved in polynomial time by solving the linear programs which result if x_i is fixed to either 0 or 1. The question we study in this section is how good the approximation $\hat{p}_j$ needs to be in order to guarantee that solving (3.5) or (3.6) recovers the original bit string.

We first define the linear map which maps a bit string $\mathbf{x}$ to the corresponding probability vector $\mathbf{y}$.

DEFINITION 3.1. *The linear map* $M^{(n)} : \mathbb{R}^n \to \mathbb{R}^n$ *is given by* $\mathbf{y} := M^{(n)}\mathbf{x}$ *such that* $y_j := \sum_{i \geq j} \mathrm{Inf}(i, j) x_i$.

The definition is such that, when Y is obtained by sending $\mathbf{x}$ through a deletion channel, and $\mathbf{y} = M^{(n)}\mathbf{x}$, then $\Pr[Y_j = 1] = y_j$. The definition can be extended without problems for infinite bitstrings; we will use the notation M^{∞} in this case.

It is now of relevance to ask how close the probability vectors of two bitstrings $\mathbf{x}^{(0)}$ and $\mathbf{x}^{(1)}$ can be, assuming that they differ at position i.

[2] Recall that $Y_j \in \{0, 1, \perp\}$, where $\perp$ is the value taken if the trace is shorter than j bits. Note that we generally do not distinguish between the case where $Y_j = \perp$ and $Y_j = 0$; however one doesn't lose much not distinguishing these cases; a slightly longer string padded with zeros will behave the same way in the relevant range.

[3] One can ensure independence of the noise in the estimates by using new traces for every position j, which only increases the number of traces by at most a factor of n.

DEFINITION 3.2. *The L_1 gap at position i of $M^{(n)}$ is*

$$g_{i,n} := \min_{\substack{\mathbf{x}^{(0)},\mathbf{x}^{(1)}\in\{0,1\}^n \\ x_{i'}^{(0)}=x_{i'}^{(1)}=0 \quad (i'<i) \\ x_i^{(0)}=0, x_i^{(1)}=1}} \|M^{(n)}\mathbf{x}^{(0)} - M^{(n)}\mathbf{x}^{(1)}\|_1$$
$$= \min_{\substack{\mathbf{x}\in\{-1,0,1\}^n \\ x_{i'}=0 \quad (i'<i) \\ x_i=1}} \|M^{(n)}\mathbf{x}\|_1$$

The fractional L_1 gap of $M^{(n)}$ at position i is

$$g_{i,n}^f := \min_{\substack{\mathbf{x}^{(0)},\mathbf{x}^{(1)}\in[0,1]^n \\ x_{i'}^{(0)}=x_{i'}^{(1)}=0 \quad (i'<i) \\ x_i^{(0)}=0, x_i^{(1)}=1}} \|M^{(n)}\mathbf{x}^{(0)} - M^{(n)}\mathbf{x}^{(1)}\|_1$$
$$= \min_{\substack{\mathbf{x}\in[-1,1]^n \\ x_{i'}=0 \quad (i'<i) \\ x_i=1}} \|M^{(n)}\mathbf{x}\|_1$$

Obviously we have $g_{i,n}^f \le g_{i,n}$. We also remark that one can formulate a linear program similar to (3.6) which computes the value of $g_{i,n}^f$.

In this section, we will prove the following bounds on the L_1 gap:

THEOREM 3.1. *For any constant δ we have*

$$g_{i,\infty}^f \ge \exp(-\tilde{O}(\sqrt{i})) \tag{3.7}$$

$$g_{i,2i} \le \exp(-O(\log^2(i)/(\log\log(i))^2)). \tag{3.8}$$

Inequality (3.7) immediately gives an algorithm which recovers the initial string with $\exp(\tilde{O}(\sqrt{i}))$ traces: for every position j we get an estimate $\hat{p}_j \approx p_j$ (where we use new traces for every j), and then we solve (3.6) iteratively to get the next position. Note that this algorithm works for any $\delta < 1$, and even if the initial bitstring is infinitely long and we want to recover it up to position n.

Inequality (3.8) gives an upper bound on how well natural algorithms which only consider estimates $\hat{p}_j$ of the probabilities p_j can perform. If such an algorithm works even if *independent Gaussian noise*[4] is added to each p_j, (3.8) simply states that unless the noise added is smaller than $\exp(-\tilde{O}(\log^2(i)))$, for worst case inputs the two resulting distributions are information theoretically nearly indistinguishable.

We can conclude that any reconstruction algorithm based only on summary estimates $\hat{p}_j$ will, in the worst case, require superpolynomially many traces, so that $|\hat{p}_j - p_j| < \exp(-\tilde{O}(\log^2(i)))$ with high probability.

[4]Depending how one estimates the probabilities, the noise may not be independent, in which case our lowerbound may in principle be overcome – such an algorithm is however hard to imagine.

3.1 Proof of (3.7) We remarked above that $g_{i,n}^f$ can be found by a linear program. Consequently, the weak duality theorem of linear programming gives an immediate way to prove (3.7): one finds an appropriate solution of the dual linear program. Because it's simple to do, we state and prove the weak linearity theorem for our particular problem.

LEMMA 3.1. *Let $\mathbf{z} \in [-1,1]^\infty$ be any vector which has only finitely many non-zero entries. Let $\mathbf{x}^* = (M^\infty)^T\mathbf{z}$. Then,*

$$g_{i,\infty}^f \ge x_i^* - \sum_{i'>i} |x_{i'}^*|.$$

Proof. For any $\mathbf{x} \in [-1,1]^n$ with $x_i = 1$, $x_{i'} = 0$ for $i' < i$ it holds that

$$\sum_{j=1}^{\infty} |M^\infty\mathbf{x}|_j \ge \sum_{j=1}^{\infty} z_j(M^\infty\mathbf{x})_j = \mathbf{z}^T M^\infty\mathbf{x}$$
$$= (\mathbf{x}^*)^T\mathbf{x} \ge x_i^* - \sum_{i'>i} |x_{i'}^*|,$$

in particular for the $\mathbf{x}$ which minimizes $\|M^{(n)}\mathbf{x}\|_1$. □

REMARK 3.1. *A vector $\mathbf{z}$ as above (in case it is short enough) also gives an immediate algorithm to find x_i, as long as $x_i^* - \sum_{i'>i}|x_{i'}^*|$ is positive: the linear combination of the probabilities $\sum z_j p_j$ is larger than a given threshold in case $x_i = 1$ and smaller otherwise.*

We now construct a vector for use in Lemma 3.1.

LEMMA 3.2. *Fix δ and i. Then, there is a $\mathbf{z} \in \mathbb{R}^\infty$ such that $z_j = 0$ for $j > (1-\delta)(i + 8\sqrt{\delta i}\log(\delta i))$ and $|z_j| < \exp(\tilde{O}(\sqrt{\delta i}\frac{\delta}{1-\delta}))$ for all j. Furthermore, for $\mathbf{x}^* = (M^\infty)^T\mathbf{z}$ we have*

$$x_i^* > \Omega((\delta i)^{3/2}(1-\delta)^{1/2})$$
$$x_{i+\tilde{i}}^* < \begin{cases} 1 & 1 \le \tilde{i} \le \delta i \\ \exp(-\tilde{O}(\delta\tilde{i})) & \tilde{i} > \delta i. \end{cases}$$

We first show how this lemma can be used to prove (3.7).

Proof. (of Theorem 2.2, (3.7)) We now get (3.7) by combining Lemma 3.1 and Lemma 3.2, after scaling $\mathbf{z}$ with a factor in $\exp(-\tilde{O}(\sqrt{\delta i}\frac{\delta}{1-\delta}))$ and using the linearity of M^∞. □

Proof. (of Lemma 3.2) We set $\mathbf{z}$ to zero for all values outside the range $[(1-\delta)i, (1-\delta)i + 8\sqrt{\delta i}\log(\delta i)]$. Because of this, the value of $x_{i+\tilde{i}}^*$ can be written as

(where we abbreviate $\mu = 1-\delta$)

$$x^*_{i+\tilde{i}} = \sum_{\tilde{j}=0}^{8\sqrt{\delta i}\log(\delta i)} \mathrm{Inf}(i+\tilde{i},\mu i+\tilde{j}) z_{\mu i+\tilde{j}}$$

$$= \mathrm{Inf}(i+\tilde{i},\mu i) \sum_{\tilde{j}=0}^{8\sqrt{\delta i}\log(\delta i)} \frac{\mathrm{Inf}(i+\tilde{i},\mu i+\tilde{j})}{\mathrm{Inf}(i+\tilde{i},\mu i)} z_{\mu i+\tilde{j}}$$

$$= \mathrm{Inf}(i+\tilde{i},\mu i) \sum_{\tilde{j}=0}^{8\sqrt{\delta i}\log(\delta i)} \frac{\binom{i+\tilde{i}-1}{\mu i+\tilde{j}-1}}{\binom{i+\tilde{i}-1}{\mu i-1}} \mu^{\tilde{j}}(1-\mu)^{-\tilde{j}} z_{\mu i+\tilde{j}}$$

$$= \mathrm{Inf}(i+\tilde{i},\mu i) \sum_{\tilde{j}=0}^{8\sqrt{\delta i}\log(\delta i)} \frac{(i-\mu i+\tilde{i})\cdots(i-\mu i+\tilde{i}-\tilde{j}-1)}{(\mu i)\cdots(\mu i+\tilde{j}-1)} \frac{\mu^{\tilde{j}}}{(1-\mu)^{\tilde{j}}} z_{\mu i+\tilde{j}}$$

$$= \mathrm{Inf}(i+\tilde{i},\mu i) \sum_{\tilde{j}=0}^{8\sqrt{\delta i}\log(\delta i)} \frac{(1+\frac{\tilde{i}}{i-\mu i})\cdots(1+\frac{\tilde{i}-\tilde{j}+1}{i-\mu i})}{(1+\frac{0}{\mu i})\cdots(1+\frac{\tilde{j}-1}{\mu i})} z_{\mu i+\tilde{j}}$$

We abbreviate $h := (1-\mu)i = \delta i$ and $w_{\tilde{i}} := 1+\frac{\tilde{i}}{h}$, and get

$$x^*_{i+\tilde{i}} = \mathrm{Inf}(i+\tilde{i},\mu i)\times \underbrace{\sum_{\tilde{j}=0}^{8\sqrt{\delta i}\log(\delta i)} \frac{z_{\mu i+\tilde{j}}}{(1+\frac{0}{\mu i})\cdots(1+\frac{\tilde{j}-1}{\mu i})}(w_{\tilde{i}}-\tfrac{0}{h})\cdots(w_{\tilde{i}}-\tfrac{\tilde{j}-1}{h})}_{Q(w_{\tilde{i}})}.$$

Thus, $x^*_{i+\tilde{i}}$ can be written as a product $\mathrm{Inf}(i+\tilde{i},\mu i)Q(1+\frac{\tilde{i}}{\delta i})$, where Q is a polynomial of degree $8\sqrt{\delta i}\log(\delta i)$. Recall that our goal is to make x^*_i much bigger than the absolute value of $x^*_{i+\tilde{i}}$ for any $\tilde{i}>0$. Once $\tilde{i}$ is very large, the term $\mathrm{Inf}(i+\tilde{i},\mu i)$ will be very small. For smaller $\tilde{i}$, we will make sure that the polynomial Q is small.

We use the following claim, which we prove right after we finish the proof of this lemma.

Claim 3.1. *For any $h>1$ there exist coefficients z'_j, $j=0,\ldots,8\sqrt{h}\log(h)$ with $|z'_j|\le\exp(\tilde{O}(\sqrt{h}))$, such that the polynomial*

$$Q_h(w) = \sum_{j=0}^{8\sqrt{h}\log(h)} z'_j w\Big(w-\frac{1}{h}\Big)\cdots\Big(w-\frac{j-1}{h}\Big) \tag{3.9}$$

satisfies $|Q_h(w)|\le 1$ for $1+\frac{1}{h}\le w\le 3$, $Q_h(w)\le w^{\tilde{O}(\sqrt{h})}$ for $w>3$, and $Q_h(1)\ge h^2$.

Using these coefficients z'_j we set (for $h=\delta i$, $j_{\max}=8\sqrt{\delta i}\log(\delta i)$)

$$z_{\mu i+\tilde{j}} := \begin{cases} z'_{\tilde{j}}(1+\frac{0}{\mu i})\cdots(1+\frac{\tilde{j}-1}{\mu i}) & \text{if } \tilde{j}\in\{0,\ldots,j_{\max}\} \\ 0 & \text{otherwise,}\end{cases}$$

in which case $x^*_{i+\tilde{i}} = \mathrm{Inf}(i+\tilde{i},\mu i)Q_{\delta i}(1+\frac{\tilde{i}}{\mu i})$ for $Q_{\delta i}$ as in Claim 3.1. Since $(1+\frac{\tilde{j}}{\mu i})\le\exp(\frac{\tilde{j}}{\mu i})$, the coefficients z_j satisfy $|z_j|\le|z'_j|\exp(\frac{j_{\max}}{\mu i})^{j_{\max}}\le\exp(\tilde{O}(\sqrt{\delta i}))\exp\big(\frac{\delta}{\mu}\log^2(\delta i)\big)$.

Further, we get $x^*_i = \mathrm{Inf}(i,\mu i)Q_{\delta i}(1)\ge\mathrm{Inf}(i,\mu i)(\delta i)^2\ge\Omega((\delta i)^{3/2}(1-\delta)^{1/2})$. For $\tilde{i}\in\{1,\ldots,\delta i\}$ it holds that

$$|x^*_{i+\tilde{i}}| = \Big|\mathrm{Inf}(i+\tilde{i},\mu i)Q_{\delta i}\big(1+\frac{\tilde{i}}{h}\big)\Big| \le \Big|Q_{\delta i}(1+\frac{\tilde{i}}{h})\Big| \le 1.$$

Finally, for $\tilde{i}>\delta i$ we use $\mathrm{Inf}(i,\mu_i+\tilde{i})<e^{-(\frac{\tilde{i}}{i})^2 i}<e^{-\delta\tilde{i}}$ (which is a Chernoff bound) and get

$$|x^*_{i+\tilde{i}}| = |\,\mathrm{Inf}(i+\tilde{i},\mu i)Q_{\delta i}(1+\frac{\tilde{i}}{h})| \le e^{-\delta\tilde{i}}(1+\frac{\tilde{i}}{\delta i})^{O(\delta i)}. \quad\square$$

Proof. (of Claim 3.1) The Chebyshev Polynomials of the first kind are $T_d(x) = \frac{1}{2}\big((x+\sqrt{x^2-1})^d + (x+\sqrt{x^2-1})^{-d}\big)$. We use the following properties (see for instance Chapter 22 in [1]).

1. In the range $-1\le x\le 1$, $T_d(x)=\cos(d\arccos x)$, implying $|T_d(x)|\le 1$
2. For $0\le\epsilon<1$, $T_d(1+\epsilon)\ge\exp(d\sqrt{\epsilon}/4)$.
3. The absolute value of the coefficients of $T_d(x)$ are all at most 3^d.

We set thus

$$Q_h(w) = T_{8\sqrt{h}\log(h)}\big(2+\frac{1}{h}-w\big).$$

Using above properties, we see that $|Q_h(w)|\le 1$ for $1+\frac{1}{h}\le w\le 3$, $Q_h(w)\le w^{\tilde{O}(\sqrt{h})}$ for $w>3$, and $Q_h(1)\ge h^2$.

To finish the proof, we need to check that $Q_h(w)$ can be written in the form $Q_h(w)=\sum_{j=0}^{8\sqrt{h}\log(h)} z'_j w(w-\frac{1}{h})\cdots(w-\frac{j}{h})$ with $|z'_j|\le e^{\tilde{O}(\sqrt{h})}$. First, one checks (using property 3 above) that the coefficients of the monomials in $Q_h(w)$ are at most $\exp(\tilde{O}(\sqrt{h}))$.

Furthermore, we can express w^j as $w^j = \sum_{i=0}^{j} a_{j,i}w(w-\frac{1}{h})\cdots(w-\frac{i-1}{h})$, where $|a_{j,i}|\le(j+1)!$, which we now show per induction on j. Abbreviate $Z_i = w(w-\frac{1}{h})\cdots(w-\frac{i-1}{h})$. First, clearly $w^0=1=Z_0$. Now, by the induction hypothesis $w^{j-1}=\sum_{i=0}^{j-1}a_{j-1,i}Z_i$. It holds that $w^j=\sum_{i=0}^{j-1}a_{j-1,i}wZ_i$ and observe that $Z_{i+1}=(w-\frac{i}{h})Z_i$ implying $wZ_i = Z_{i+1}+\frac{i}{h}Z_i$. This gives $w^j=\sum_{i=0}^{j-1}a_{j-1,i}(Z_{i+1}+\frac{i}{h}Z_i) = a_{j-1,j-1}Z_j + \sum_{i=0}^{j-1}(a_{j-1,i}\frac{i}{h}+a_{j-1,i-1})Z_i$, and thus we set $a_{j,j}=a_{j-1,j-1}$ and $a_{j,i}=\frac{i}{h}a_{j-1,i}+a_{j-1,i-1}$ for $i<j$.

Thus, we write every monomial in $Q_h(w)$ in this form, and obtain an expression of the form (3.9) for $Q_h(w)$, with $|z'_j|\le\exp(\tilde{O}(\sqrt{h}))$. $\square$

3.2 Proof of (3.8) We will explicity construct an $\mathbf{x} \in \{-1,0,1\}^{2i}$ with $x_i = 1$ such that $\|M^{(2i)}\mathbf{x}\|_1 \leq \exp(-O(\log^2(i)))$. For this, we start with the vector $\mathbf{v}^{(0)} := (1)$ of length 1, recursively define $\mathbf{v}^{(\alpha)} = (\mathbf{v}^{(\alpha-1)}, -\mathbf{v}^{(\alpha-1)})$, and then set $\mathbf{x}$ to the vector which contains $i-1$ zeros in the beginning, and then $\mathbf{v}^{(k)}$ for (chosen with foresight) $k := \log(i)/(\log\log(i))^2$ and then some more zeros (to pad the length to $2i$). The intuition of this construction is that after every iteration, $\mathbf{v}^{(\alpha)}$ will mapped under M onto something which is a bit closer to the zero vector.

For a function f and $c > 0$ we define $\Delta_c f$ to be the function

$$(\Delta_c f)(x) := f(x+c) - f(x).$$

Furthermore, we inductively define $\Delta_{c_1,c_2,\ldots,c_k}(f) = \Delta_{c_1}(\Delta_{c_2,\ldots,c_k} f)$.

Using above definition of $\mathbf{x}$ we get for $\mathbf{y} = M^{(2i)}\mathbf{x}$ (where by convention Δ operates on the first argument of the function Inf, i.e., on i)

$$y_j = (-1)^k \Delta_{2^{k-1},2^{k-2},\ldots,1} \operatorname{Inf}(i,j). \tag{3.10}$$

Therefore our plan will be to get a bound on $\Delta_{2^{k-1},2^{k-2},\ldots,1} \operatorname{Inf}(i,j)$. To this end we the use a generalization of the mean value theorem from calculus, which gives a relation between Δf and $\frac{d}{dx} f$. We omit the proof due to space constraints.

LEMMA 3.3. *Let $f : [0,\infty) \to \mathbb{R}$ be C^∞ and fix $c_1, \ldots, c_k > 0$. Then,*

$$(\Delta_{c_1,c_2,\ldots,c_k} f)(x) = \Big(\prod_{m=1}^{k} c_m\Big) \cdot \frac{d^k}{dx^k} f(x + \chi_x)$$

where $\chi_x \in [0, \sum_m c_m]$.

With (3.10) in mind we are interested in getting bounds on the absolute value of $\frac{d^k}{di^k} \operatorname{Inf}(i,j) = \frac{d^k}{di^k}\big(\frac{\Gamma(i)}{\Gamma(j)\Gamma(i-j+1)}(1-\delta)^j \delta^{i-j}\big)$, where $\Gamma(x) := \int_0^\infty t^{x-1} e^{-t} dt$ with $\Gamma(n) = (n-1)!$ is the gamma function. We also use the standard definitions $\Psi^{(0)}(x) := \frac{d}{dx}\ln(\Gamma(x))$ and $\Psi^{(k)}(x) := \frac{d^k}{dx^k}\Psi^{(0)}(x)$. A straightforward calculation yields

$$\frac{d}{di} \operatorname{Inf}(i,j) = \operatorname{Inf}(i,j) g(i,j) \tag{3.11}$$

where we define

$$g(i,j) := \ln(\delta) + \Psi^{(0)}(i) - \Psi^{(0)}(i-j+1),$$

and we note that $\frac{d^k}{di^k} g(i,j) = \Psi^{(k)}(i) - \Psi^{(k)}(i-j+1)$.

We have the following bounds on the absolute value of of g and $\frac{d^k}{dx^k} g$. We omit the proof, which uses a result by Alzer [2], due to space constraints.

LEMMA 3.4. *Let $|\epsilon| + \frac{1}{\delta i} < \frac{1}{2}$. For $j = i(1-\delta-\delta\epsilon)$ and $j \in \mathbb{N}$ we have $|g(i,j)| \leq |2\epsilon| + \frac{2}{\delta i}$.*

LEMMA 3.5. *If $k \geq 1$, $i > j > 1$ we have*

$$\Big|\frac{d^k}{di^k} g(i,j)\Big| < \frac{4 \cdot k!}{(i-j)^k}.$$

From these bounds we obtain a bound on $|\frac{d^k}{di^k} \operatorname{Inf}(i,j)|$. For this, we first describe how equation (3.11) implies that we can write $\frac{d^k}{di^k} \operatorname{Inf}(i,j)$ as a product of $\operatorname{Inf}(i,j)$ and a polynomial of bounded size in derivatives of $g(i,j)$. The proof of the following lemma is omitted in this version due to space constraints.

LEMMA 3.6. *Let $f(x)$ be C^∞ with $f(x) > 0$, and $\frac{d}{dx} f(x) = f(x) g^{(1)}(x)$. Then $g^{(1)}$ is C^∞, and defining $g^{(k)}(x) = \frac{d}{dx} g^{(k-1)}(x)$ we have for any $k \geq 1$:*

$$\frac{d^k}{dx^k} f(x) = f(x) \sum_{\alpha=1}^{s(k)} \prod_{\beta} g^{(p_{\alpha,\beta})}(x) \tag{3.12}$$

where $p_{\alpha,\beta} \in \mathbb{N}_{>0}$, $s(k) \leq (k+1)!$, and $\sum_\beta p_{\alpha,\beta} = k$ for all α (i.e., for any fixed α, the $p_{\alpha,\beta}$ form a partition of k).

The above estimates on g, derivatives on g, and our rewriting finally allows us to give an upper bound on $|\frac{d^k}{di^k} \operatorname{Inf}(i,j)|$. The proof, which we omit due to space constraints, follows by appropriately combining Lemma 3.4, Lemma 3.5 and Lemma 3.6.

LEMMA 3.7. *Let $i - j = (1+\epsilon)\delta i$, $i \geq \frac{4}{\delta}$, $|\epsilon| < \frac{1}{4}$ and $j \in \mathbb{N}$. Then,*

$$\Big|\frac{d^k}{di^k} \operatorname{Inf}(i,j)\Big| \leq \operatorname{Inf}(i,j) \cdot 2^{2k\log(k)(1+o(1))} \Big(\max\big(|\epsilon|, \tfrac{1}{\sqrt{\delta i}}\big)\Big)^k$$

We can now prove (3.8) in Theorem 2.2.

Proof. (of Theorem 2.2, (3.8)) Define recursively the vectors $\mathbf{v}^{(\alpha)}$ of length 2^α as $\mathbf{v}^{(0)} = 1$, $\mathbf{v}^{(\alpha)} := (\mathbf{v}^{(\alpha-1)}, -\mathbf{v}^{(\alpha-1)})$ and set $\mathbf{x} := (0^{i-1}, \mathbf{v}^{(k)}, 0^m)$ where m is chosen such that $\mathbf{x}$ is of length $2i$.

Let $j_- := i(1-\delta) - \sqrt{ki\ln(i)/2}$ and $j_+ := i(1-\delta) + \sqrt{ki\ln(i)/2} + 2^k$. Then, we have

$$\sum_{j=1}^{\infty} |(M^{(2i)}\mathbf{x})_j| = \sum_{j<j_-} |(M^{(2i)}\mathbf{x})_j| + \sum_{j=j_-}^{j_+} |(M^{(2i)}\mathbf{x})_j| + \sum_{j>j_+} |(M^{(2i)}\mathbf{x})_j|.$$

We see that

$$\sum_{j<j_-} |(M^{(2i)}\mathbf{x})_j| \leq 2^k \Pr[\text{At least } \delta i + \sqrt{ki\ln(i)/2} \text{ deletions occur in the first } i \text{ bits}],$$

and thus this is at most $2^k e^{-k\ln(i)/2} = 2^k i^{-k/2}$ using a Chernoff bound. The same reasoning yields $\sum_{j>j_+} |(M^{(2i)}\mathbf{x})_j| \leq 2^k i^{-k/2}$. To get a bound on $\sum_{j=j_-}^{j_+} |(M^{(2i)}\mathbf{x})_j|$, we first note that

$$(M^{(2i)}\mathbf{x})_j = (-1)^k \Delta_{2^{k-1},2^{k-2},\ldots,1}(\mathrm{Inf}(i,j)), \tag{3.13}$$

which one easily shows per induction on k. From Lemma 3.3 and Lemma 3.7 we thus get the bound (note that in the following equations $|\epsilon| < \sqrt{\frac{k\ln(i)}{2i\delta^2}} + \frac{2^k}{i\delta}$):

$$\begin{aligned}
&\sum_{j=j_-}^{j_+} \left|(M^{(2i)}\mathbf{x})_j\right| \\
&\leq 2^{\frac{k(k+1)}{2}} \sum_{j=j_-}^{j_+} \left|\frac{d^k}{di^k}\mathrm{Inf}(i+\chi_i, j)\right| \\
&\leq 2^{\frac{k(k+1)}{2}} \sum_{j=j_-}^{j_+} \mathrm{Inf}(i,j) 2^{k\log k} \Big(\max\big(|\epsilon|, \frac{1}{\sqrt{\delta i}}\big)\Big)^k \\
&\leq 2^{\frac{k(k+1)}{2}+2k\log(k)(1+o(1))} \sum_{j=j_-}^{j_+} \mathrm{Inf}(i,j) \times \\
&\qquad \Big(\max\Big(\sqrt{\frac{k\ln(i)}{2i\delta^2}} + \frac{2^k}{\delta i}, \frac{1}{\sqrt{\delta i}}\Big)\Big)^k \\
&= 2^{k^2}\Big(\sqrt{\frac{k\ln(i)}{2i\delta^2}} + \frac{2^k}{\delta i}\Big)^k \sum_{j=j_-}^{j_+} \mathrm{Inf}(i,j) \\
&\leq 2^{k^2}\Big(\sqrt{\frac{k\ln(i)}{2i\delta^2}} + \frac{2^k}{\delta i}\Big)^k
\end{aligned}$$

For $k := \log(i)/(\log\log(i))^2$ we have for large enough i:

$$\begin{aligned}
2^{k^2}\Big(\sqrt{\frac{k\ln(i)}{2i\delta^2}} + \frac{2^k}{\delta i}\Big)^k &\leq 2^{k^2}\Big(\sqrt{\frac{k\ln(i)}{i\delta^2}}\Big)^k \\
&= 2^{k\log(i)/(\log\log(i))^2 - \frac{1}{2}k\log(i\delta^2) + \frac{1}{2}k\log(k\log(i))} \\
&= 2^{-O(k\log(i))},
\end{aligned}$$

and this concludes the proof. □

4 Conclusion

We have provided several results enhancing our understanding of the trace reconstruction problem, with our main result being that for original sequences chosen uniformly at random and sufficiently small constant deletion probabilities, a polynomial number of samples suffices. There remain many open questions to pursue. Obviously, the constants in our original argument could be improved. Further generalizing this result to handle insertions, transposition errors, and/or bit flips as well as deletions would be an important step forward. Improved upper and lower bounds for many variations of the problem appear possible.

In practice, we suspect richer algorithms based on iteratively voting with beliefs would perform well in terms of both accuracy and computation. In such schemes, bits are again determined iteratively, with each string computing its belief (a conditional probability) that the next bit of $\mathbf{x}$ is a 0 or 1 based on the prior bits, and these beliefs being combined to determine the next bit. Refining our analysis to obtain results for such algorithms would be an interesting challenge.

Furthermore, the question whether a polynomial time algorithm can recover *any* string (not just a random one) is still open. In particular, we do not know of an efficient algorithm which reliably returns the position of the first 1 in an arbitrary string using a polynomial number of samples.

References

[1] M. Abramowitz, I. A. Stegun. Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. *Dover publication*, 1964.

[2] H. Alzer. Sharp inequalities for the digamma and polygamma functions *Forum Mathematicum*, vol. 16, no. 2, pp. 181–221. 2004.

[3] T. Batu, S. Kannan, S. Khannna, and A. McGregor. Reconstructing strings from random traces. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 910-918, 2004.

[4] R. L. Dobrushin. Shannon's Theorems for Channels with Synchronization Errors. *Problems of Information Transmission*, 3(4):11-26, 1967. Translated from *Problemy Peredachi Informatsii*, vol. 3, no. 4, pp 18-36, 1967.

[5] E. Drinea and M. Mitzenmacher. Improved lower bounds for the capacity of i.i.d. deletion and duplication channels. *IEEE Trans. on Information Theory*, 53:8, pp. 2693-2714, 2007.

[6] S. Kannan and A. McGregor. More on reconstructing strings from random traces: insertions and deletions. In *Proc. of the Int'l. Symp. on Information Theory*, pp. 297–301, 2005.

[7] V. I. Levenshtein. Efficient reconstruction of sequences. *IEEE Transactions on Information Theory*, vol. 47, no. 1, pp. 2–22, 2001.

[8] V. I. Levenshtein. Efficient reconstruction of sequences from their subsequences or supersequences. *Journal of Combinatorial Theory, Series A*, vol. 93, no. 2, pp. 310–332, 2001.