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 when the original string is chosen uniformly at random from all strings of length n, there exists a constant γ such that for $\delta < \gamma$, 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.

[*]Microsoft Research, Silicon Valley. Email: thomahol@microsoft.com

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

[‡]Microsoft Research, Silicon Valley. Email: rina@microsoft.com

[§]Microsoft Research, Silicon Valley. Email: uwieder@microsoft.com

1 Introduction

In this paper, we consider the following problem: a binary string $X = x_1x_2 \dots x_n$ yields a collection of *traces* $Y^1, Y^2, \dots, Y^m$, where each Y^i is independently obtained from 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* X exactly with high probability. The question is how many traces are necessary for reconstruction. Our primary new result is the following: when X is chosen uniformly at random, there exists a constant γ such that for $\delta < \gamma$, reconstruction is possible with polynomially many traces with high probability (over both 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. That is, let $z_j(w)$ be the probability that the jth bit of a trace is 1 when the original string is w (and the deletion probability δ is implicit). We show that for any constant $\delta < 1$, for sufficiently large n there exist 0-1 strings w^1 and w^2 of length n such that $z_j(w^1)$ and $z_j(w^2)$ are superpolynomially close for all j. As we argue in Section 4, 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 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 X is *not* chosen from an agreed codebook, but instead chosen at random; the coding problem provides an interesting variation for future work.

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 (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 X, as opposed to randomly chosen 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.

2 A polynomial trace algorithm for random X and small δ

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.

We also emphasize that while we are considering uniform random strings here, our methods should extend naturally to other situations, such as when our strings are biased so that each bit is 0 independently with some probability $q > 1/2$.

2.2 An exponential algorithm

We begin with an algorithm for reconstructing the first h bits of the original string X of length n using $e^{O(h)}$ traces (and similar time). This algorithm works for any X; we do not assume X is random here. While such an algorithm is obviously expensive, it allows us to recover the first $O(\log n)$ bits of X, from which we bootstrap our algorithm. We improve on this algorithm in Section 3, but the algorithm in this section is simpler.

Theorem 2.1. *There is an algorithm that determines the first h bits of X with $\tilde{O}(e^{6h\delta\ln(\frac{1}{\delta})})$ traces with high probability when $\delta < \frac{1}{3}$.*

The algorithm determines the bits sequentially. The main observation is that the influence of x_i on bit $j = i(1-3\delta)$ in a trace is greater than the cumulative influence of all subsequent bits $x_{i+1}, x_{i+2}, .., x_n$ combined. That is, there exists a threshold S which can be computed given $x_1, \ldots, x_{i-1}$, such that if $x_i = 1$ then $\Pr[Y_i = 1] > S$, and if $x_i = 0$, then $\Pr[Y_j = 1] < S$, regardless of the values of the later bits.

Let $P(i,j)$ denote the probability that x_i ends up as the jth bit in a trace. We have

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

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

The following simple lemma states two properties which we can directly use to reconstruct X. We state a more general form than what we need here that we reuse in Section 2.3.

Lemma 2.2. *If $j \le (1-3\delta)i$ then $P(i,j) \ge 2\sum_{i'>i} P(i',j)$. If $(1-4\delta)i < j < (1-3\delta)i$ then $P(i,j) \ge e^{-6\delta i}$.*

Proof. For the first statement it is sufficient to note that for any $i' > i$ the inequality $\frac{P(i',j)}{P(i'-1,j)} = \frac{\binom{i'}{j}}{\binom{i'-1}{j}}\delta = \frac{i'}{i'-j}\delta \le \frac{1}{3}$ holds, since this implies that the series is upper bounded by a geometric series with $q = \frac{1}{3}$. When $(1-4\delta)i < j < (1-3\delta)i$, the second statement follows from $P(i,j) = \binom{i-1}{i-j}(1-\delta)^j\delta^{i-j} \ge \left(\frac{i-1}{i-j}\right)^{i-j}(1-\delta)^j\delta^{i-j} \ge \left(\frac{1}{4\delta}\right)^{i-j}(1-\delta)^j\delta^{i-j} \ge \left(\frac{1}{4}\right)^{i-j}(1-\delta)^j \ge \left(\frac{1}{4}\right)^{3\delta i} e^{\ln(1-\delta)(1-4\delta)i} \ge e^{-6\delta i}$. □

Using this, we can prove Theorem 2.1

Proof of Theorem 2.1. Assume that the algorithm has already determined $x_1, \ldots, x_{i-1}$. To recover x_i we set $j = (1-3\delta)i$ and write

$$\Pr[Y_j = 1] = \sum_{\ell=j}^{n} P(\ell,j)x_\ell = \sum_{\ell=j}^{i-1} P(\ell,j)x_\ell + P(i,j)x_i + \sum_{\ell=i+1}^{n} P(\ell,j)x_\ell.$$

From Lemma 2.2 we get $0 \le \sum_{\ell=i+1}^{n} P(\ell,j)x_\ell \le \frac{1}{2}P(i,j)$, and since $\sum_{\ell=j}^{i-1} P(\ell,j)x_\ell$ can be easily computed given $x_1, \ldots x_{i-1}$ we see that if we know $Pr[Y_j = 1]$ up to an additive error of $\frac{P(i,j)}{4} \ge \frac{1}{4}e^{-3\delta i \ln(\frac{1}{\delta})}$ we can learn x_i. Using standard Chernoff bounds, this means that $\tilde{O}(e^{6h\delta \ln(\frac{1}{\delta})})$ independent traces suffice to reconstruct the first h bits with high probability. □

In particular, since $\delta \ln(1/\delta) \le \frac{1}{2}$, Theorem 2.1 implies that for any c the first $c \log n$ bits of 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

In this section, we now assume that X is random, and δ is a sufficiently small constant. The string X is determined inductively; the exponential trace algorithm provides our base to start from. The essential idea is to look for a copy of a substring from the input string in the output as an anchor point. For example after we have learnt $x_1, x_2, .., x_{i-1}$, we could use the substring $S = x_{i-w}x_{i-w+1}...x_{i-1}$ of width w (where w will be $O(\log n)$) and look for output strings containing this substring S. So if an output string Y contains a substring $y_{j-w}y_{j-w+1}...y_{j-1}$ that matches S, presumably y_j can be used to learn x_i.

Unfortunately this is not quite the case. Even conditioned on matching this anchor substring, it is possible that the bit y_j is influenced most not by x_i but by some later bits. This happens for instance when the last few bits of the anchor string S are all zeros. In this case even if $x_i = 0$, it is possible that the conditional probability that $y_j = 1$ is large (if $x_{i+1}, x_{i+2}, \ldots$ are all 1). We rectify this problem by using the anchor to help determine a bit further on in X, rather than the bit immediately following it.

We first show that the chance that an anchor substring appears in a completely different region of the input string is negligible when its width w is chosen appropriately. It is here that we use the fact that X is a uniform random string. In what follows, we use $X(a : b)$ to represent the substring $x_a, x_{a+1}, \ldots, x_{b-1}$. Also, we emphasize that essentially no effort has been made to optimize the constants in our proof.

Definition 2.3. *A string X of length n is* w-substring unique *if for all a, b, one of the following holds:*

- *The substring $X(a : a+w)$ cannot be obtained by deleting some symbols in $X(b : b+1.1w)$.*
- *$b \le a$ and $b + 1.1w \ge a + w$.*

We show later (namely in Lemma 2.6) that most strings of length n are $O(\log(n))$-substring-unique, and our algorithm in fact recovers all substring-unique strings correctly. Next we show that in a substring-unique string it is possible to find "anchors" to help determine from where trace bits arose.

Lemma 2.4. *Let δ be a small enough constant and let X of length n be w-substring unique. Let Y be a trace of X after application of a deletion channel with deletion probability δ.*

Conditioned on the event that Y contains a substring $Y(j-w:j)$ that matches $X(i-w:i)$ then the probability that the bit y_{j-1} does not come from a bit in the range $x_{i-1}\ldots x_{i-1+0.1w}$ is at most $n\delta^{0.001w}$.

Proof. Assume for the moment that for no a more than $0.1w$ symbols in $X(a:a+1.1w)$ are deleted. Since X is w-substring unique this implies that y_{j-1} must come from the range $x_{i-1}\ldots x_{i-1+0.1w}$.

The probability that for a fixed a more than $0.1w$ symbols in $X(a:a+1.1w)$ are deleted is at most, using a Chernoff bound, $\delta^{1.1w0.05^2}\leq\delta^{0.002w}$ and thus the probability that this happens at any position a is at most $n\delta^{0.002w}$.

On the other hand, the probability that no symbol in $X(i-w:i)$ is deleted is $(1-\delta)^w$, and thus the conditioned probability we look for is at most $n\delta^{0.002w}/(1-\delta)^w\geq n\delta^{0.001w}$. ☐

Assume for the moment that we could detect strings in which bit y_{j-1} came from bit x_i. In this case, Lemma 2.2 immediately provides a way to learn bit x_{i+k} as long as $k\in O(\log(n))$, since we can just look at position $j+k(1-3\delta)$. However, we can only recognize that y_{j-1} comes from a certain range of bits with high probability. Nevertheless, this is enough to recover the next bit.

Theorem 2.5. *Let δ be a small enough constant, and let X be a $100\log(n)$-substring unique string of length n. There exists a polynomial time algorithm which recovers X with probability $1-o(1)$ from $\operatorname{poly}(n)$ independent traces of X which are obtained by independently deleting each bit in X with probability δ.*

Proof. Assume that the algorithm has already obtained bits $x_1,\ldots,x_{i-1}$. We show how to obtain x_i. We assume that $i\geq\frac{100}{\delta}\log(n)$ (otherwise, the algorithm from Theorem 2.1 can be used).

Our algorithm first obtains sample traces and discards those which do not contain $x_{i-v-w},\ldots,x_{i-v}$ verbatim in an arbitrary position (in other words, it only keeps traces which contain the above substring without deletion). For the traces which do contain the substring, we set Y to the remainder of the trace. In other words, the algorithm deletes all the bits up to and including the last bit in the match.

Note that we match a substring of length w, and do it in a position which is v bits ahead of the position to reconstruct.

Let R be the random variable which denotes the position of the last bit matched in the original string. We set $w=100\log(n)$ and $v=\frac{w}{\delta}$ and let $j=(v-0.1w)(1-3\delta)$. We have

$$\Pr[Y_j=1]=\sum_r\Pr[R=r]\sum_{\ell=1}^{n}P(\ell,j)x_{r+\ell}$$

Now, according to Lemma 2.4, with probability at least $n\delta^{0.001w}$ it holds that $i-v\leq R\leq i-v+0.1w$. It holds therefore that

$$\Pr[Y_j=1]=A_i(X)+\sum_{r=i-v}^{i-v+0.1w}\Pr[R=r]\sum_{\ell=1}^{n}P(\ell,j)x_{r+\ell},$$

where $0\leq A_i(X)\leq n\delta^{0.0001w}\leq n\frac{1}{100^w}$. Continuing,

$$=A_i(X)+\sum_{r=i-v}^{i-v+0.1w}\Pr[R=r]\sum_{\ell=r+1}^{n}P(\ell-r,j)x_\ell$$

$$
\begin{aligned}
&= A_i(X) + \sum_{r=i-v}^{i-v+0.1w} \Pr[R=r]\Big(\sum_{\ell=r+1}^{i-1} P(\ell-r,j)x_\ell + P(i-r,j)x_i + \sum_{\ell=i+1}^{n} P(\ell-r,j)x_\ell\Big) \\
&= A_i(X) + \underbrace{\sum_{r=i-v}^{i-v+0.1w} \Pr[R=r] \sum_{\ell=r+1}^{i-1} P(\ell-r,j)x_\ell}_{=:S} \\
&\qquad + \sum_{r=i-v}^{i-v+0.1w} \Pr[R=r]\Big(P(i-r,j)x_i + \sum_{\ell=i+1}^{n} P(\ell-r,j)x_\ell\Big) \qquad (2)
\end{aligned}
$$

We have $j \le (v-0.1w)(1-3\delta) \le (i-r)(1-3\delta)$ for all r, and thus we can use the first part of Lemma 2.2 which implies that $P(i-r,j) \ge \frac{1}{2}\sum_{\ell=i+1}^{n} P(\ell-r,j)$.

Thus, in case $x_i = 0$ equation (2) implies

$$\Pr[Y_j = 1] \le A_i(X) + S + \sum_{r=i-v}^{i-v+0.1w} \Pr[R=r]\frac{P(i-r,j)}{2}.$$

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

$$\Pr[Y_j = 1] \ge A_i(X) + S + \sum_{r=i-v}^{i-v+0.1w} \Pr[R=r]P(i-r,j).$$

Now, the difference between $A_i(X)$ when $x_i = 0$ and $A_i(X)$ when $x_i = 1$ is at most $n\frac{1}{100^w}$. Furthermore, given that $x_0, \ldots, x_{i-1}$ are known, S can be calculated up to an accuracy of $(\frac{1}{100})^w$ in polynomial time. Furthermore, for all r in the range above we have, using Lemma 2.2: $P(i-r,j) \ge (\frac{1}{2})^{-(i-r-j)} \ge \frac{1}{2}^{-(v-j)} = \frac{1}{2}^{0.1w+3\delta v-0.3\delta w} = \frac{1}{2}^{0.1w+3w-0.3\delta w} \ge \frac{1}{2}^{4w}$. So, since $\sum_{r=i-v}^{i-v+0.1w} \Pr[R=r]$ is $1-o(1)$, the gap between the two cases is at least $\frac{1}{2}^{4w+1}$. □

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

Lemma 2.6. *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 $[a, a+w]$ and $[b, b+1.1w]$ for which not both $b \le a$ and $b+1.1w \ge a+w$; w.l.o.g. we consider the case $b > a$. Since for a fixed deletion pattern, one can choose the string X by selecting bits in increasing order, the probability that $X(a : a+w)$ can be obtained by deleting symbols in $X(b : b+1.1w)$ is at most $\binom{1.1w}{0.1w}2^{-w} \le (11e)^{0.1w}2^{-w} \le (1.415)^{-w}$. Since there are fewer than n^2 disjoint intervals, we get that the probability that two contradicting substrings exist is at most $n^2(1.415)^{-w} \le n^{-d}$. □

Corollary 2.7. *Let δ be a small enough constant. There exists a polynomial time algorithm which recovers fraction $1-n^{-48}$ of all strings X with probability $1-o(1)$ from $\mathrm{poly}(n)$ independent traces of X which are obtained by independently deleting each bit in X with probability δ.*

Proof. According to Lemma 2.6 this large fraction of the strings of length n are substring unique, and we apply Theorem 2.5. □

3 An Algorithm for Large Deletion Probabilities

The algorithm of Section 2.2 reconstructed each bit x_h based on the fraction of ones at one location in the traces. If bit x_h is not deleted, it is expected to land in location $(1-\delta)h$ and with high probability lands in locations $(1-\delta)h \pm O(\sqrt{h}\log h)$. In this section we show that by examining some linear combination of the fraction of ones in a collection of locations in the traces we can reconstruct X using $\exp(O(\sqrt{n}\log n))$ traces for every X and every deletion probability $\delta < 1$. Note that this algorithm does not require X to be substring unique.

We let $z_i(X)$ be the probability that the ith bit of a trace is 1 when X is the original string; we use z_i in place of $z_i(X)$ where the meaning is clear.

For ease of notation we set $\mu = 1-\delta$, i.e., μ is the probability of a bit not being deleted. As before we assume we know $x_1, \ldots, x_{h-1}$ and wish to recover x_h; in fact we can assume w.l.o.g. that all the previous bits were 0 because our algorithm is entirely linear. The bit x_h has the most influence on the probability $z_{\mu h}$. We show how x_h can be obtained by a linear combination of the values $z_{\mu h}, z_{\mu h+1}, .., z_{\mu h+k}$ where k is $O(\sqrt{h}\log(h))$. In particular, we show a linear combination $S = \sum_{j=0}^{k} c_j z_{\mu h+j}$ with the following properties:

1. There is a number a such that $x_h = 1$ implies $S \geq a+1$ and $x_h = 0$ implies $S \leq a$.
2. For each j it holds that $|c_j| \leq \exp(O(\sqrt{h}\log h))$ and $k \in O(\sqrt{h}\log h)$.

Item (1) implies that x_h can be reconstructed by estimating S within a small constant additive error. Item (2) implies that with high probability $\exp(\tilde{O}(\sqrt{h}))$ traces suffice to obtain such an estimation.

Similarly to the exponential algorithm, the main idea is to produce coefficients c_j so that the influence of x_h on S is large and the cumulative influence of all the later bits is small.

Let $P_h(i,j) = \binom{h+i-1}{\mu h+j-1}\mu^{\mu h+j}(1-\mu)^{h+i-\mu h-j} = P(h+i, \mu h+j)$ denote the probability x_{h+i} ends up at location $\mu h + j$. Define the influence on S of bit x_{h+i} as $\mathrm{Inf}(i) = \sum_{j=0}^{k} c_j P_h(i,j)$. It holds that $S = \sum_{i\geq 0}\mathrm{Inf}(i)x_{h+i}$. Note that if $x_h = 1$ then $|S| \geq |\mathrm{Inf}(0)|$ and if $x_h = 0$ then $|S| \leq \sum_{i\geq 1}|\mathrm{Inf}(i)|$. It is therefore sufficient to show the following theorem:

Theorem 3.1. *There exists a linear combination* $\mathrm{Inf}(i) = \sum_{j=0}^{k} c_j P_h(i,j)$ *such that* $k \in O(\sqrt{h}\log h)$*, for each* j *it holds that* $|c_j| \in e^{O(\sqrt{h}\log h)}$*, and* $|\mathrm{Inf}(0)| - \sum_{i\geq 1}|\mathrm{Inf}(i)| > 1$.

The above theorem implies that by measuring the distribution of the output bits $z_{\mu h}, z_{\mu h+1}, .., z_{\mu h+k}$ up to an accuracy of $e^{-\tilde{O}(\sqrt{h})}$ the bit x_h can be reconstructed.

The main idea of the proof is to express $\mathrm{Inf}(i)$ as a polynomial that takes a large value at $i = 0$ and a small value at all $i > 0$. This is done by reducing $\mathrm{Inf}(i)$ to the Chebyshev polynomials of the first kind. Chebyshev polynomials are known to be small in the interval $[-1,1]$ and increase fast outside the interval, while having small degree and small coefficients.

We proceed by a series of variable changes aimed at expressing $\mathrm{Inf}(i)$ in a cleaner form. Note that

$$\begin{aligned}
\mathrm{Inf}(i) &= \sum_{j=0}^{k} c_j P_h(i,j) \\
&= P_h(i,0)\sum_{j=0}^{k} c_j P_h(i,j)/P_h(i,0) \\
&= P_h(i,0)\sum_{j=0}^{k} c_j \binom{h+i-1}{\mu h+j-1}\Big/\binom{h+i-1}{\mu h-1}\mu^j(1-\mu)^{-j} \\
&= P_h(i,0)\sum_{j=0}^{k} c_j \frac{(h-\mu h+i)(h-\mu h+i-1)..(h-\mu h+i-j+1)}{(\mu h)(\mu h+1)..(\mu h+j-1)}\mu^j(1-\mu)^{-j} \\
&= P_h(i,0)\sum_{j=0}^{k} c_j \frac{(1+\frac{i}{h-\mu h})(1+\frac{i-1}{h-\mu h})..(1+\frac{i-j+1}{h-\mu h})}{(1+\frac{0}{\mu h})(1+\frac{1}{\mu h})..(1+\frac{j-1}{\mu h})}
\end{aligned}$$

Substituting, $c'_j = \frac{c_j}{(1+\frac{0}{\mu h})(1+\frac{1}{\mu h})..(1+\frac{j-1}{\mu h})}$ and $w_i = 1 + \frac{i}{h-\mu h}$, and using h' to denote $(1-\mu)h$, we get

$$\text{Inf}(i) = P_h(i,0)\sum_{j=0}^{k} c'_j w_i(w_i - \frac{1}{h'})(w_i - \frac{2}{h'})\dots(w_i - \frac{j-1}{h'})$$

Note that

$$c_j \le c'_j e^{j^2/(\mu h)} \le c'_j e^{k^2/(\mu h)} \in \exp(O(\frac{\log^2 h}{\mu}))c'_j \tag{3}$$

so a bound $|c'_j| \in \exp(\sqrt{h}\log h)$ implies that $|c_j| \in \exp(\sqrt{h}\log h)$.

Observe that $w_0 = 1$ and for $i > 0$, $w_i \ge 1 + 1/h'$. For every i we examine the polynomial $Q(w_i) = \sum_{j=0}^{k} c'_j w_i(w_i - \frac{1}{h'})(w_i - \frac{2}{h'})\dots(w_i - \frac{j-1}{h'})$. Recall that our goal is to find coefficients c'_j so that $\text{Inf}(i) = P_h(i,0)Q(w_i)$ is large at $w_i = 0$ and small for $w_i \ge 1 + 1/h'$ for all $i > 0$.

Note that $P_h(i,0)$ decreases exponentially with i, so clearly at some point $P_h(i,0)$ dominates $Q(w_i)$ and $\text{Inf}(i)$ is small. We show that this happens when $i \ge h$, i.e. when $w_i \ge 3$. The main difficulty is to find coefficients such that $Q(w_i)$ is large when $w_i = 1$ and small when $1 + 1/h' \le w_i \le 3$. Since the value of w_i is such bounded, we may drop the subscript i and treat w_i as a variable.

At this point the Chebyshev Polynomials come to play. Chebyshev Polynomials have the property of being small in the range $[-1,1]$ and increase in value very fast outside this range. We show how the polynomial $Q(w)$ could be reduced to the Chebyshev Polynomials.

Lemma 3.2. *There exists a polynomial $R(w) = \sum_{j=0}^{d} r_j w^j$ of degree $d = \tilde{O}(\sqrt{h})$ with the following properties:*

1. *Each coefficient is bounded in size, $|r_j| \in e^{O(\sqrt{h}\log h)}$*
2. *$R(1) \ge h^2$ and $|R(w)| \le 1$ for $1 + 1/h \le w \le 3$.*

Proof. We make use of the Chebyshev's Polynomials of the first kind: $T_d(x) = \frac{1}{2}[(x + \sqrt{x^2-1})^d + (x + \sqrt{x^2-1})^{-d}]$. The following properties of Chebyshev's Polynomials are well known (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 small $\epsilon > 0$, $T_d(1+\epsilon) = \Theta(e^{d\sqrt{\epsilon}})$
3. The absolute value of the coefficients of $T_d(x)$ are $O(\exp(d))$

Note that there is a constant c such that if $d \ge c\sqrt{h}\log h$, we get that $T_d(1 + 1/h) \ge h^2$. Observe the polynomial

$$R(w) = T_d(2 + 1/h - w)$$

It is straightforward to verify that $R(1) \ge h^2$ and that $|R(w)| \le 1$ for $1 + 1/h \le w \le 3$. The coefficients of this polynomial are at most $e^{\tilde{O}(\sqrt{h})}$ □

Now all that remains is to convert the polynomial $R(w)$ into the format of $Q(w) = \sum_{j=0}^{k} c'_j w(w - \frac{1}{h'})(w - \frac{2}{h'})\dots(w - \frac{j-1}{h'})$ without blowing up the size of the coefficients. This is again done by variable change.

Lemma 3.3. *Let $Z_j = w(w - \frac{1}{h'})(w - \frac{2}{h'})\dots(w - \frac{j-1}{h'})$. Then, there exists $a_i^{(j)}$ such that $w^j = \sum_{i=0}^{j} a_i^{(j)} Z_i$ and $a_i^{(j)} \in O(\exp(j))$.*

Proof. Use induction on j, clearly $w = Z_1$. Now, by the induction hypothesis $w^{j-1} = \sum_{i=0}^{j-1} a_i^{(j-1)} Z_i$. It holds that $w^j = \sum_{i=0}^{j-1} a_i^{(j-1)} w Z_i$ 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 $w^j = \sum_{i=0}^{j-1} a_i^{(j-1)}(Z_{i+1} + \frac{i}{h'}Z_i) = a_{j-1}^{(j-1)} Z_j + \sum_{i=0}^{j-1}(a_i^{(j-1)}\frac{i}{h'} + a_{i-1}^{(j-1)})Z_i$ □

Combining Lemmas 3.2 and 3.3 we get:

Lemma 3.4. *There exists a polynomial $Q(w) = \sum_{j=0}^{k} c'_j w^j$ where $k \in O(\sqrt{h})$ and c'_j are $e^{\tilde{O}(\sqrt{h})}$ so that $Q(1) \geq h^2$ and $|Q(w)| \leq 1$ for $1 + 1/h \leq w \leq 3$. For $w > 3$, $Q(w) \in w^{\tilde{O}(\sqrt{h})}$*

In order to complete the proof of Theorem 3.1 recall that $\mathrm{Inf}(i) = P_h(i,0)Q(w_i)$. It holds that $P_h(0,0) \in \Omega(1/\sqrt{h})$ and $Q(w_0) \geq h^2$ implying $\mathrm{Inf}(0) \in \Omega(h^{3/2})$.

It remains to bound $\sum_{i>0} |\mathrm{Inf}(i)|$. Now, in the range $1 \leq i \leq h$ it holds that $|Q(w_i)| \leq 1$ and therefore $|\mathrm{Inf}(i)| \leq 1$. In the range $i \geq h$ it holds $P_h(i,0)$ is so small it dominates $Q(w_i)$. In particular, there is a constant $c > 1$ such that $P_h(i,0) \leq c^{-i}$ for every $i \geq h$. Clearly it holds that $|Q(w_i)| \in \exp(\tilde{O}(\sqrt{h}))$. We conclude that at $i = h$ it holds that $\mathrm{Inf}(i)$ is exponentially small and drops at least geometrically with increasing i. We conclude that $\sum_{i>0} |\mathrm{Inf}(i)| \leq 2h$ completing the proof of Theorem 3.1.

4 Lower bounds on L_1 distance of summary trace data

4.1 The importance of L_1 distance

Our algorithms for reconstruction involve examining the individual traces. A natural question that arises is whether one needs to actually examine the traces in this way. A natural summary of the samples can be obtained by taking the fraction of ones in each position over all of the traces. The expected value α_k of the fraction of ones in the kth position is given by

$$\alpha_k = \sum_{j=k}^{n} P(j,k) x_j .$$

Given sample values $\hat{\alpha_k}$, one might hope that the original string X could be recovered easily. A natural approach is to consider the following $0-1$ integer linear program with variables x_j to choose the n-bit string that minimizes the L_1 distance between the expected values α_k and the sampled values $\hat{\alpha_k}$:

$$\min \sum_{j=1}^{n} \beta_j$$

$$\begin{aligned} \beta_k &\geq \sum_{j=k}^{n} P(j,k) x_j - \hat{\alpha_k},\ 1 \leq k \leq n; \\ \beta_k &\geq \hat{\alpha_k} - \sum_{j=k}^{n} P(j,k) x_j,\ 1 \leq k \leq n; \\ & x_k \in \{0,1\},\ 1 \leq k \leq n. \end{aligned}$$

One might hope that with a polynomial number of estimates the sampled values $\hat{\alpha_k}$ could be sufficiently accurate that some relaxation of the above program could yield the original string with high probability. Our results in this section show that this approach is doomed to failure. Indeed, based on just L_1 distance, even an exhaustive examination of all possible strings X cannot succeed, because we show that there are 0-1 strings w^1 and w^2 such that the vector of α_k values for the two strings differ in L_1 distance inverse superpolynomially in n.

We note that technically we cannot yet exclude the possibility that there is some alternative means of distinguishing strings w^1 and w^2 from samples, as our current proof focuses solely on L_1 distance. Strengthening this result by using a stronger notion of statistical indistinguishability remains a point for future work.

4.2 L_1 distance lower bounds

The map from original bit string X to the corresponding probabilities for each bit in the trace that the bit takes on the value 1 is linear, and it is possible to extend it to a linear map $M_\delta : \mathbb{R}^\infty \to \mathbb{R}^\infty$. For a vector $v \in \mathbb{R}^\infty$ we use $\|v\|_1$ to denote $\sum_{i=1}^{\infty} |v_i|$.

We prove the following Theorem in this section.

Theorem 4.1. *Fix a constant $\delta \in [0,1]$, For any n there exist two strings $w^1, w^2 \in \{0,1\}^\infty$ which differ at position n and for which*

$$\|M_\delta w^1 - M_\delta w^2\|_1 \leq n^{-\tilde{\Omega}_\delta(\log^2(n))}.$$

Since M_δ is linear, it is sufficient to construct a single string $w \in \{-1,0,1\}^\infty$ for which $\|M_\delta w\|_1 \leq n^{-\tilde{\Omega}_\delta(\log^2(n))}$. We get w by starting with the vector $v^{(0)} = (1)$ of length 1, recursively defining $v^{(i)} = (v^{(i-1)}, -v^{(i-1)})$, and then setting w to the vector which contains $n-1$ zeros in the beginning, and then $v^{(k)}$ for some appropriately chosen k, and then infinitely many zeros.

For a function f and $d > 0$ we define $\Delta_d f$ 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 this definition, and for $y = M_\delta w$ we get for Δ operating on the first argument of the function $y_j = \sum_{i=1}^\infty \Delta_{2^{k-1},2^{k-2},\ldots,1} P(n+i,j)$. Therefore our plan we will be to get a bound on $\Delta_{2^{k-1},2^{k-2},\ldots,1} P(n+i,j)$. The mean value theorem from calculus gives a relation between Δf and $\frac{d}{dx} f$, and analogously we can give the following generalization of it to get a relation between $\Delta_{c_1,\ldots,c_k} f$ and $\frac{d^k}{dx^k} f$. A proof is given in Appendix A.

Lemma 4.2. *Let $f : [0,\infty) \to \mathbb{R}$ be C^∞ and fix $c_1, \ldots, c_k$ with $c_i > 0$. Then,*

$$(\Delta_{c_1,c_2,\ldots,c_k} f)(x) = \left(\prod_{i=1}^k c_i\right) \cdot \frac{d^k}{dx^k} f(x + \chi_x)$$

where $\chi_x \in [0, \sum_i c_i]$.

Consequently, we are interested in getting bounds on the absolute value of $\frac{d^k}{di^k} P(i,j)$.

To be able to express these derivatives we use the (standard) definitions $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$, $\Psi^{(0)}(x) = \frac{d}{dx} \ln(\Gamma(x))$ and $\Psi^{(k)}(x) = \frac{d^k}{dx^k} \Psi^{(0)}(x)$.

A straightforward calculation yields

$$\frac{d}{di} P(i,j) = P(i,j) g(i,j), \text{ where} \tag{4}$$

$$g(i,j) = \ln(\delta) + \Psi^{(0)}(i) - \Psi^{(0)}(i-j+1). \tag{5}$$

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

We prove the following bounds on the absolute value of g and $\frac{d^k}{dx^k} g$ in Appendix B.

Lemma 4.3. *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 4.4. *If $k \geq 1$, $i > j > 1$ we have $\left|\frac{d^k}{di^k} g(i,j)\right| < \frac{4 \cdot k!}{(i-j)^r}$.*

From these bounds we want to obtain a bound on $|\frac{d^k}{di^k} P(i,j)|$. For this, we first describe how equation (4) implies that we can write $\frac{d^k}{di^k} P(i,j)$ as a product of $P(i,j)$ and a polynomial of bounded size in derivatives of $g(i,j)$. The proof of the following Lemma is given in Appendix C

Lemma 4.5. *Let $f(x)$ be C^∞ with $f(x) > 0$, and $f'(x) = f(x) g^{(1)}(x)$. Then $g^{(1)}$ is C^∞, and defining $g^{(k)}(x) = \frac{d}{dx} g^{(k-1)}(x)$ we have for any $k \geq 1$:*

$$\frac{d^k}{dx^k} f(x) = f(x) \sum_{\alpha=1}^{s(k)} \prod_\beta g^{(p_{\alpha,\beta})}(x) \tag{6}$$

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 $|\frac{d^k}{di^k}P(i,j)|$.

Lemma 4.6. *Let $i-j=(1+\epsilon)\delta i$, $i \geq \frac{1}{\delta}$, $\epsilon < 1/2k$ and $j \in \mathbb{N}$. Then,*

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

Proof. Define $g^{(1)}(i,j) := g(i,j)$ and $g^{(t)}(i,j) := \frac{d}{di}g^{(t-1)}(i,j)$. From Lemmas 4.3 and 4.4 we get

$$|g^{(1)}(i,j)| \leq 2|\epsilon| + \frac{2}{\delta i} \leq 4\max(|\epsilon|, \tfrac{1}{\sqrt{\delta i}}) \tag{7}$$

$$|g^{(t)}(i,j)| \leq \frac{4\cdot(t-1)!}{(i-j)^{t-1}} \leq \frac{4\cdot(t-1)!}{(i-j)^{t/2}} \leq \frac{8\cdot(t-1)!}{(\sqrt{\delta i})^t} \qquad \text{(for } t\geq 2\text{)} \tag{8}$$

From Lemma 4.5 we know that we can write

$$\frac{d^k}{di^k}P(i,j) = P(i,j)\sum_{\alpha=1}^{s(k)}\prod_{\beta} g_k^{(p_{\alpha,\beta})}$$

for $s(k) < (k+1)!$ and some partition $p_{\alpha,\beta}$ of k. Now, for any fixed α we have

$$\prod_{\beta} g^{(p_{\alpha,\beta})}(i,j) < (k-1)!\cdot 8^k\Big(\max\big(|\epsilon|, \tfrac{1}{\sqrt{\delta i}}\big)\Big)^k, \tag{9}$$

which follows from (7) and (8).

We have at most $(k+1)!$ terms of the form (9), and since $(k+1)!(k-1)! < 2^{2k\log k}$ we get the lemma. ☐

We can now prove Theorem 4.1.

Proof of Theorem 4.1. Define recursively the vectors $v^{(\alpha)}$ of length 2^α as $v^{(0)} = 1$, $v^{(\alpha)} := (v^{(\alpha-1)}, -v^{(\alpha-1)})$ and set $w := (0^{n-1}, v^{(k)}, 0^\infty)$ (we will determine k later).

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

$$\sum_{j=1}^{\infty}|(M_\delta w)_j| = \sum_{j<j_-}|(M_\delta w)_j| + \sum_{j=j_-}^{j_+}|(M_\delta w)_j| + \sum_{j>j_+}|(M_\delta w)_j|$$

We see that

$$\sum_{j<j_-}|(M_\delta w)_j| \leq 2^k \Pr[\text{At least } \sqrt{kn\ln(n)/2} \text{ deletions occur in the first } n \text{ bits}],$$

and thus this is at most $2^k e^{-k\ln(n)/2} = 2^k n^{-k/2}$ using Hoeffdings bound. The same reasoning yields

$$\sum_{j>j_+}|(M_\delta w)_j| \leq 2^k n^{-k/2}.$$

To get a bound on $\sum_{j=j_-}^{j_+}|(M_\delta w)_j|$, we first note that

$$(M_\delta w)_j = \Delta_{2^{k-1},2^{k-2},\ldots,1}(P(n,j)), \tag{10}$$

which one easily shows per induction on k. From Lemmas 4.2 and 4.6 we thus get the bound (note that in the following equations $|\epsilon| < \sqrt{\frac{k\ln(n)}{2n\delta^2}} + \frac{2^k}{n\delta}$):

$$\begin{aligned}
\sum_{j=j_-}^{j_+} \Big|(M_\delta w)_j\Big| &\leq 2^{\frac{k(k+1)}{2}} \sum_{j=j_-}^{j_+} \Big|\frac{d^k}{dn^k} P(n+\xi, j)\Big| \\
&\leq 2^{\frac{k(k+1)}{2}} \sum_{j=j_-}^{j_+} P(n,j) 2^{k\log k}\Big(\max\big(|\epsilon|, \frac{1}{\sqrt{\delta n}}\big)\Big)^k \\
&\leq 2^{\frac{k(k+1)}{2} + k\log(k)} \sum_{j=j_-}^{j_+} P(n,j)\Big(\max\Big(\sqrt{\frac{k\ln(k)}{2n\delta^2}} + \frac{2^k}{\delta n}, \frac{1}{\sqrt{\delta n}}\Big)\Big)^k \\
&= 2^{\frac{k(k+1)}{2} + k\log(k)} \Big(\sqrt{\frac{k\ln(k)}{2n\delta^2}} + \frac{2^k}{\delta n}\Big)^k \sum_{j=j_-}^{j_+} P(n,j) \\
&\leq 2^{k^2}\Big(\sqrt{\frac{k\ln(k)}{2n\delta^2}} + \frac{2^k}{\delta n}\Big)^k
\end{aligned}$$

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

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

□

5 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 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.

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.

[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. To appear in *IEEE Transactions on Information Theory*.

[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.

A Proof of the Generalization of the Mean Value Theorem

In this section, we give the proof of Lemma 4.2.

Proof of Lemma 4.2. We first note that $\Delta_{c_1,c_2,\ldots,c_k} f$ is C^∞ and $\frac{d^r}{dx^r}\Delta_{c_1,c_2,\ldots,c_k} f(x) = \Delta_{c_1,c_2,\ldots,c_k}\frac{d^r}{dx^r} f(x)$, which one easily checks by noting that $\frac{d}{dx}\Delta_c f(x) = \Delta_c \frac{d}{dx} f(x)$.

We now show the theorem per induction on k. The mean value theorem gives the base case and implies

$$\Delta_{c_1,c_2,\ldots,c_k} f(x) = c_1 \Delta_{c_2,\ldots,c_k} f'(x+\alpha_x) \tag{11}$$

for some $\alpha_x \in [0, c_1]$. The induction hypothesis gives

$$\Delta_{c_2,\ldots,c_k} f'(y) = \left(\prod_{i=2}^{k} c_i\right)\cdot \frac{d^{k-1}}{dy^{k-1}} f'(y+\beta_y),$$

where $\beta_y \in [0, \sum_{i\geq 2} c_i]$. Inserting this into (11) for $y = x+\alpha_x$ finishes the proof. $\square$

B Bounds on g and derivatives of g

In this section, we prove Lemmas 4.3 and 4.4 which upper bound the absolute value of g and $\frac{d^k}{di^k} g$. We utilize the following simple result.

Claim B.1. *Let* $|\epsilon_1| + |\epsilon_2| < \frac{1}{2}$. *Then,* $|\ln(\frac{1+\epsilon_1}{1+\epsilon_2})| \leq 2|\epsilon_1| + 2|\epsilon_2|$.

Proof of Lemma 4.3. Using the functional equation $\Psi^{(0)}(z+1) = \Psi^{(0)}(z) + \frac{1}{z}$ we get $g(i,j) = \ln(\delta) + \frac{1}{i-1} + \frac{1}{i-2} + \cdots + \frac{1}{i-j+1}$, and thus $\ln(\delta) + \int_{i-j}^{i-1}\frac{1}{s}ds \leq g(i,j) \leq \ln(\delta) + \int_{i-j+1}^{i}\frac{1}{s}ds$, i.e.,

$$\ln\left(\frac{\delta(i-1)}{i-j}\right) \leq g(i,j) \leq \ln\left(\frac{\delta i}{i-j+1}\right). \tag{12}$$

Now, $\left|\ln(\frac{\delta i}{i-j+1})\right| = \left|\ln(\frac{1}{1+\epsilon+\frac{1}{\delta i}})\right| \leq \left|2\epsilon + \frac{2}{\delta i}\right|$ and $\left|\ln\left(\frac{\delta(i-1)}{i-j}\right)\right| = \left|\ln\left(\frac{1-\frac{1}{i}}{1+\epsilon}\right)\right| \leq \left|2\epsilon+\frac{2}{i}\right|$ imply the lemma. $\square$

Proof of Lemma 4.4. From [2, Lemma 1 (1)], we get for any $i \geq 1$

$$i^k|\Psi^{(k)}(i)| \leq |\Psi^{(k)}(1)| = k!\zeta(k+1) \leq 2\cdot k!, \tag{13}$$

where $\zeta(k) = \sum_{s=1}^{\infty} \frac{1}{s^k} < 2$ (for $k \geq 2$) is the Riemann zeta function. Thus,

$$\begin{aligned} \left| \frac{d^k}{di^k} g(i,j) \right| &= |\Psi^{(k)}(i)| + |\Psi^{(k)}(i-j+1)| \\ &\leq \frac{2 \cdot k!}{i^k} + \frac{2 \cdot k!}{(i-j+1)^k} < \frac{4 \cdot k!}{(i-j+1)^k}. \end{aligned}$$

□

C Proof of Lemma 4.5

Proof of Lemma 4.5. First, $g^{(1)}$ is C^∞ because we can write it as $g^{(1)}(n) = f'(x)/f(x)$ and then use that f is C^∞.

We show (6) per induction on k. The case $k = 1$ is obvious. For $k > 1$ we use

$$\frac{d}{dx} f(x) \prod_\beta g^{(p_{\alpha,\beta})}(x) = f(x)g(x) \prod_\beta g^{(p_{\alpha,\beta})}(x) + f(x) \frac{d}{dx} \prod_\beta g^{(p_{\alpha,\beta})}(x),$$

and note that $f(x)\frac{d}{dx} \prod_\beta g^{(p_{\alpha,\beta})}(x)$ can be written a sum of at most k terms of the form $f(x) \prod_\beta g_{q_{\alpha,\beta}}(x)$ where q is now a partition of $k+1$. This also implies that $s(k+1) \leq (k+1)s(k)$. □