Near-Optimal Trace Reconstruction for Mildly Separated Strings

Anders Aamand[*]   Allen Liu[†]   Shyam Narayanan[‡]

September 14, 2025

Abstract

In the *trace reconstruction* problem our goal is to learn an unknown string $x \in \{0,1\}^n$ given independent *traces* of x. A trace is obtained by independently deleting each bit of x with some probability δ and concatenating the remaining bits. It is a major open question whether the trace reconstruction problem can be solved with a polynomial number of traces when the deletion probability δ is constant. The best known upper bound and lower bounds are respectively $\exp(\tilde{O}(n^{1/5}))$ [Cha21b] and $\tilde{\Omega}(n^{3/2})$ [Cha21a]. Our main result is that if the string x is *mildly separated*, meaning that the number of zeros between any two ones in x is at least polylog n, and if δ is a sufficiently small constant, then the trace reconstruction problem can be solved with $O(n \log n)$ traces and in polynomial time.

1   Introduction

Trace reconstruction is a well-studied problem at the interface of string algorithms and learning theory. Informally, the goal of trace reconstruction is to recover an unknown string x given several independent noisy copies of the string.

Formally, fix an integer $n \geq 1$ and a deletion parameter $\delta \in (0,1)$. Let $x \in \{0,1\}^n$ be an unknown binary string with x_i representing the ith bit of x. Then, a *trace* $\tilde{x}$ of x is generated by deleting every bit x_i independently with probability δ (and retaining it otherwise), and concatenating the retained bits together. For instance, if $x = 01001$ and we delete the second and third bits, the trace would be 001 (from the first, fourth, and fifth bits of x). For a fixed string x, note that the trace follows some distribution over bitstrings, where the randomness

comes from which bits are deleted. In *trace reconstruction*, we assume we are given N i.i.d. traces $\tilde{x}^{(1)}, \ldots, \tilde{x}^{(N)}$, and our goal is to recover the original string x with high probability.

The trace reconstruction problem has been a very well studied problem over the past two decades [Lev01a, Lev01b, BKKM04, KM05, HMPW08, VS08, MPV14, DOS19, NP17, PZ17, HHP18, HL20, HPP18, Cha21a, CDL+21b, CDL+21a, Cha21b, Rub23]. There have also been numerous generalizations or variants of trace reconstruction studied in the literature, including coded trace reconstruction [CGMR20, BLS20], reconstructing mixture models [BCF+19, BCSS19, Nar21], reconstructing alternatives to strings [DRR19, KMMP21, NR21, MS22, SY23, MS24], and approximate trace reconstruction [DRSR21, CP21, CDK21, CDL+22, CDL+23].

In perhaps the most well-studied version of trace reconstruction, x is assumed to be an arbitrary n-bit string and the deletion parameter δ is assumed to be a fixed constant independent of n. In this case, the best known algorithm requires $e^{\tilde{O}(n^{1/5})}$ random traces to reconstruct x with high probability [Cha21b]. As we do not know of any polynomial-time (or even polynomial-sample) algorithms for trace reconstruction, there have been many works making distributional assumptions on the string x, such as x being a uniformly random string [HMPW08, MPV14, PZ17, HPP18, Rub23] or x being drawn from a "smoothed" distribution [CDL+21b]. An alternative assumption is that the string x is parameterized, meaning that x comes from a certain "nice" class of strings that may be amenable to efficient algorithms [KMMP21, DRSR21].

In this work, we also wish to understand parameterized classes of strings for which we can solve trace reconstruction efficiently. Indeed, we give an algorithm using polynomial traces and runtime, that works for a general class of strings that we call L-separated strings. This significantly broadens the classes of strings for which polynomial-time algorithms are known [KMMP21].

Main Result Our main result concerns trace reconstruction of strings that are *mildly separated*. We say that a string x is L-separated if the number of zeros between any two consecutive ones is at least L. Depicting a string $x \in \{0,1\}^n$ with t ones as

$$\underbrace{0 \ldots 0}_{a_0 \text{ times}} 1 \underbrace{0 \ldots 0}_{a_1 \text{ times}} 1 \cdots 1 \underbrace{0 \ldots 0}_{a_t \text{ times}},$$

it is L-separated if and only if $a_i \geq L$ for each i with $1 \leq i \leq t-1$. Note that we make no assumptions on a_0 or a_t. Our main result is as follows.

Theorem 1.1. There exists an algorithm that solves the trace reconstruction problem with high probability in n on any L-separated string x, provided that $L \geq C(\log n)^8$ for a universal constant C, and that the deletion probability is at most some universal constant c_0. The algorithm uses $N = O(n \log n)$ independently sampled traces of x, $\tilde{x}^{(1)}, \ldots, \tilde{x}^{(N)}$, and runs in $\text{poly}(n)$ time.

We note that the number of traces is nearly optimal. Even distinguishing between two strings x, x' which contain only a single one at positions $\lfloor n/2 \rfloor$ and $\lfloor n/2 \rfloor + 1$ respectively, requires $\Omega(n)$ traces to succeed with probability $1/2 + \Omega(1)$.

2

While trace reconstruction is known to be solvable very efficiently for random strings [HPP18, Rub23], there are certain structured classes of strings that appear to be natural hard instances for existing approaches. Our algorithm can be seen as solving one basic class of hard instances. It is worth noting the work by [CDL$^+$21b] which studies the trace reconstruction problem when the deletion probability δ is sub-constant. They show that the simple Bitwise Majority Alignment (BMA) algorithm from [BKKM04] can succeed with $1/n^{o(1)}$ deletion probability as long as the original string does not contain *deserts* – which are highly repetitive blocks where some short substring is repeated many times. They then combine this with an algorithm for reconstructing repetitive blocks – but this part of their algorithm requires a significantly smaller deletion probability of $\delta \leq 1/n^{1/3+\varepsilon}$. This suggests that strings containing many repetitive blocks are a natural hard instance and good test-bed for developing new algorithms and approaches. L-separated strings can be thought of as the simplest class of highly repetitive strings (where the repeating is pattern is just a 0), where every repetition has length at least L.

Comparison to Related Work Most closely related to our work is the result by Krishnamurthy et al. [KMMP21] stating that if x has at most k ones and if each pair of ones is separated by a *run* of zeros of length $\Omega(k \log n)$, then x can be recovered in polynomial time from poly(n) many traces. In particular, for strings with $k = O((\log n)^7)$ ones, the required separation is milder than ours, albeit not below $\Omega(\log n)$. Our algorithm works in general assuming a polylog n separation of the ones but with no additional requirement on the number of ones: indeed, we could even have $\frac{n}{\text{polylog } n}$ ones. Assuming no sparsity assumptions, [KMMP21] would need to set $L \geq \Omega(\sqrt{n \log n})$, as a $\sqrt{n \log n}$-separated string can be $\Theta(\sqrt{n/\log n})$-sparse in the worst case. The techniques of [KMMP21] are also very different than ours. They recursively cluster the positions of the ones in the observed traces to correctly align a large fraction of the ones in the observed traces to ones in the string x. In contrast, our algorithm works quite differently and is of a more sequential nature processing the traces from left to right (or right to left). See Section 1.1 for a discussion of our algorithm.

Another paper studying strings with large runs is by Davies et al. [DRSR21]. They consider *approximate* trace reconstruction, specifically how many traces are needed to approximately reconstruct x up to edit distance εn under various assumptions on the lengths of the runs of zeros and ones in x. Among other results but most closely related to ours, they show that one can ε-approximately reconstruct x using $O((\log n)/\varepsilon^2)$ traces if the runs of zeros have length $\gg \frac{\log n}{\varepsilon}$ and if the runs of ones are all of length $\leq C \log n$ or $\gg 3C \log n$ for a constant C (e.g. they could have length one as in our paper). However, for exact reconstruction, they would need to set $\varepsilon < 1/n$, which means they do not provide any nontrivial guarantees in our setting.

1.1 Technical Contributions

In this section, we give a high level overview of our techniques. Recall that we want to reconstruct a string $x \in \{0, 1\}^n$ from independent traces $\tilde{x}$ where we assume that x is mildly separated. More concretely, we assume that there are numbers $a_0, \ldots, a_t \gg \text{polylog}\, n$ such that x consists of a_0 zeros followed by a one, followed by a_1 zeros followed by a one and so on, with the last a_t bits of x being zero. Writing $a_{\leq i} = \sum_{0 \leq j \leq i} a_j$, we thus have that there are t ones in x at positions $a_{\leq i} + i + 1$ for $0 \leq i \leq t - 1$.

Note that a retained bit in a trace $\tilde{x}$ naturally corresponds to a bit in x. More formally, for a trace $\tilde{x}$ of length ℓ, let $i_1 < \cdots < i_\ell$ be the ℓ positions in x where the bit was retained when generating $\tilde{x}$ so that $\tilde{x} = x_{i_1} \cdots x_{i_\ell}$. Then, the correspondence is defined by the map from $[\ell]$ to $[n]$ mapping $j \mapsto i_j$. We think of this map as the correct *alignment* of $\tilde{x}$ to x.

Our main technical contribution is an alignment algorithm (see Algorithm 1) which takes in some $m \leq t$ and estimates $b_0, \ldots, b_{m-1}$ of $a_0, \ldots, a_{m-1}$ satisfying that for all i, $|b_i - a_i| = O(\sqrt{a_i \log n})$, and correctly aligns the one in a trace $\tilde{x}$ corresponding to the m'th one of x with probability $1 - O(\delta)$ (where the randomness is over the draw of $\tilde{x}$ –naturally, this requires that the m'th one of x was not deleted).

Moreover, we ensure that the alignment procedure has that with high probability, say $1 - O(n^{-100})$, it never aligns a one in $\tilde{x}$ too far to the right in x: if the one in $\tilde{x}$ corresponding to the m_0'th one of x is aligned to the m'th one of x, then $m \leq m_0$. We will refer to this latter property by saying that the algorithm is *never ahead* with high probability. If $m < m_0$, we say that the algorithm is *behind*. Thus, to show that the algorithm correctly aligns the m'th one, it suffices to show that the probability that the algorithm is behind is $O(\delta)$.

We first discuss how to implement this alignment procedure and then afterwards we discuss how to complete the reconstruction by using this alignment procedure.

The alignment procedure of Algorithm 1. The main technical challenge of this paper is the analysis of Algorithm 1. Let us first describe on a high level how the algorithm works. For $0 \leq j \leq j' \leq m$, we write $b_{j:j'} = \sum_{i=j}^{j'-1} b_j$. Suppose that the trace $\tilde{x}$ consists of s_0 zeros followed by a one followed by s_1 zeros followed by a one and so on. The algorithm first attempts to align the first one in $\tilde{x}$ with a one in x by finding the minimal j_0 such that $(1 - \delta) \cdot b_{0:j_0}$ is within $C \log n \sqrt{b_{0:j_0}}$ of s_0 for a sufficiently large C. Inductively, having determined j_i (that is the alignment of the i'th one of $\tilde{x}$), it looks for the minimal $j_{i+1} > j_i$ satisfying that there is a $j_i \leq j' < j_{i+1}$ such that $b_{j':j_{i+1}} \cdot (1 - \delta)$ is within $C \log n \sqrt{b_{j':j_{i+1}}}$ of s_{i+1}. Intuitively, when looking at the i'th one in the trace, we want to find the *earliest* possible location in the real string (which has gaps estimated by $b_0, b_1, \ldots$) that could plausibly align with the one in the trace.

It is relatively easy to check that the algorithm is never ahead with very high probability. Indeed, by concentration bounds on the number of deleted zeros and the fact that $|b_j - a_j| = O(\sqrt{a_j \log n})$ for all $j \leq m$, it always has the option of aligning the $(i + 1)$'st one in $\tilde{x}$ to the correct one in x. However, it might align to an earlier one in x since it is looking for the *minimum* j_{i+1} such that an alignment is possible. For a very simple example, suppose that $a_0 = n^{\Omega(1)}$ and $a_1 = \cdots = a_m = b_1 = \cdots = b_m = \text{polylog}(n)$. If the first $k < m$ ones

4

of x are deleted and the $(k+1)$'st one is retained, the algorithm will align the retained one (which corresponds to the $(k+1)$'st one of x) with the first one of x resulting in the aligning algorithm being k steps behind. Moreover, the algorithm will remain k steps behind all the way up to the m'th one of x. The probability of this happening is $\Theta(\delta^k)$. To prove that the probability of the algorithm being behind when aligning the m'th one of x is at most $1 - O(\delta)$, we prove a much stronger statement which is amenable to an inductive proof, essentially stating that this is the worst that can happen: The probability of the algorithm being k steps behind at any fixed point is bounded by $(C\delta)^k$ for a constant C. In particular, we show that there is a sort of amortization – whenever there is a substring that can cause the algorithm to fall further behind with some probability (i.e. if certain bits are deleted), the substring also helps the algorithm catch back up if it is already behind.

Reconstructing x using Algorithm 1. Using Algorithm 1 we can iteratively get estimates $b_0, \ldots, b_t$ with $|b_i - a_i| = O(\sqrt{a_i \log n})$. Namely, suppose that we have the estimates $b_0, \ldots, b_m$. We then run Algorithm 1 on $O(\log n)$ independent traces and with high probability, for a $1 - O(\delta)$ fraction of them, we have that the m'th and $(m + 1)$'st one of x are retained in $\tilde{x}$ and correctly aligned. In particular, with probability $1 - O(\delta)$ we can identify both the m'th and $(m + 1)$'st one of x in $\tilde{x}$ and taking the median over the gaps between these (and appropriately rescaling by $\frac{1}{1-\delta}$), we obtain an estimate of b_{m+1} such that $|b_{m+1} - a_{m+1}| = O(\sqrt{a_{m+1} \log n})$. Note that the success probability of $1 - O(\delta)$ is enough to obtain the coarse estimates using the median approach but we cannot obtain a fine estimate by taking the average since with constant probability $O(\delta)$, we may have misaligned the gap completely and then our estimate can be arbitrarily off.

To obtain fine estimates, we first obtain coarse estimates, say $b_0, \ldots, b_t$, for all of the gaps. Next, we show that we can identify the m'th and $(m + 1)$'st one in x in a trace $\tilde{x}$ (if they are retained) and we can detect if they were deleted not just with probability $1 - O(\delta)$ but with very high probability. The trick here is to run Algorithm 1 both from the left and from the right on $\tilde{x}$ looking for respectively the one in $\tilde{x}$ aligned to the m'th one in x and the one in $\tilde{x}$ aligned to the $(m + 1)$'st one in x (which is the $(t - m)$'th one when running the algorithm from the right). If either of these runs fails to align a one in $\tilde{x}$ to respectively the m'th and $(m + 1)$'st one in x or the runs disagree on their alignment, then we will almost certainly know. To see why, assuming that we are never ahead in the alignment procedure from the left, if we believe we have reached the m'th one in x, then we are truly at some m_0'th one where $m_0 \geq m$. By a symmetric argument, if we believe we have reached the $(m + 1)$'st one in x after running the procedure from the right, we are truly at the m_1'th one in x, where $m_1 \leq m + 1$. The key observation now is that $m_0 \leq m_1$ *if and only if* $m_0 = m$ and $m_1 = m + 1$, meaning that both runs succeeding is equivalent to the one found in the left-alignment procedure being strictly earlier than the one found in the right-alignment procedure. So, if we realize that either run fails to align the ones properly, we discard the trace and repeat on a newly sampled trace.

Finally, we can ensure that the success of the runs of the alignment algorithm is independent of the deletion of zeros between the m'th and $(m + 1)$'st ones in x. If a trace is

not discarded, then with very high probability, the gap between the ones in $\tilde{x}$ aligned to the m'th and $(m+1)$'st ones in x (normalized by $\frac{1}{1-\delta}$) is an unbiased estimator for a_{m+1}. By taking the average of the gap over $\tilde{O}(n)$ traces, normalizing by $\frac{1}{1-\delta}$, and rounding to the nearest integer, we determine a_{m+1} exactly with very high probability. Doing so for each m, reconstructs x.

Road map of our paper   In Section 2, we introduce notation. In Section 3, we describe and analyse our main alignment procedure. We first prove that with high probability it is never ahead (Lemma 3.1). Second, in Section 3.2, we bound the probability that it is behind (Lemma 3.2). Finally, in Section 4, we describe our full trace reconstruction algorithm and prove Theorem 1.1.

2   Notation

We note a few notational conventions and definitions.

- We recall that a bitstring x is *L-separated* if the gap between any consecutive 1's in the string contains at least L 0's.

- Given an string x, we say that a *run* is a contiguous sequence of 0's in x. For $x = \underbrace{0\ldots0}_{a_0 \text{ times}} 1 \underbrace{0\ldots0}_{a_1 \text{ times}} 1 \cdots 1 \underbrace{0\ldots0}_{a_t \text{ times}}$, the ith run of x is the sequence $\underbrace{0\ldots0}_{a_i \text{ times}}$, and has length a_i.

- For any bitstring $x = x_1 x_2 \cdots x_n$, we use $\mathrm{rev}(x) := x_n x_{n-1} \cdots x_1$ to denote the string where the bits have been reversed.

- We use $\mathbf{a} = a_0, a_1, \ldots, a_{m-1}$ to denote an integer sequence of length m. For notational convenience, for any $0 \le j < j' \le m$, we write $\mathbf{a}_{j:j'}$ to denote the subsequence $a_j, a_{j+1}, \ldots, a_{j'-1}$, and $a_{j:j'} := \sum_{i=j}^{j'-1} a_i$.

We will define some sufficiently large constants C_0, C_1, C_2, C_3 and a small constant c_0. We will assume the separation parameter $L = C_3 \cdot \log^8 n$, and the deletion parameter $\delta \le c_0$, where $c_0 = \frac{1}{3 \cdot 10^6}$. We did not make significant effort to optimize the constant c_0 or the value 8 in $\log^8 n$, though we believe that any straightforward modifications to our analysis will not obtain bounds such as $c_0 \ge \frac{1}{2}$ or a separation of $L = O(\log n)$.

3   Main Alignment Procedure

3.1   Description and Main Lemmas

In this section, we consider a probabilistic process that models a simpler version of the trace reconstruction problem that we aim to solve. In the simpler version of the trace reconstruction problem, suppose that we never delete any 0's, but delete each 1 independently

with δ probability. Let $a_0, \ldots, a_{m-1} \leq n$ represent the true lengths of the first m gaps (so the first 1 is at position $1 + a_0$, the second 1 is at position $2 + a_0 + a_1$, and so on). Moreover, suppose we have some current predictions $b_0, \ldots, b_{m-1} \leq n$ of the gaps $a_0, \ldots, a_{m-1}$. The high level goal will be, given a single trace (where the trace means only 1s are deleted), to identify the mth 1 in the trace from the the original string with reasonably high probability. (Note that the mth 1 is deleted with δ probability, in which case we cannot succeed.)

We will describe and analyze the probabilistic process, and then explain how the analysis of the process can help us solve the trace reconstruction problem in Section 4.

In the process, we fix $m \leq n$ and two sequences $\mathbf{a} = a_0, \ldots, a_{m-1}$ and $\mathbf{b} = b_0, \ldots, b_{m'-1}$ where $\mathbf{a}$ has length m but $\mathbf{b}$ has some length m' which may or may not equal m. Moreover, we assume $L \leq a_i \leq n$ and $L \leq b_j \leq n$ for every term $a_i \in \mathbf{a}$ and $b_j \in \mathbf{b}$.

Now, for each $1 \leq i \leq m - 1$, let $w_i \in \{0, 1\}$ be i.i.d. random variables, with $w_i = 1$ with $1 - \delta$ probability and $w_i = 0$ with δ probability. Also, let $w_0 = w_m = 1$ with probability 1. For each $0 \leq i \leq m$ with $w_i = 1$, we define a value f_i as follows. First, we set $f_0 = 0$. Next, for each index $i \geq 1$ such that $w_i = 1$, let i_0 denote the previous index with $w_{i_0} = 1$. We define f_i to be the smallest index $j' > f_{i_0}$ such that there exists $f_{i_0} \leq j < j'$ with $|b_{j:j'} - a_{i_0:i}| \leq C_0 \cdot \log n \cdot \sqrt{b_{j:j'}}$, where C_0 is a sufficiently large constant. (If such an index does not exist, we set $f_i = \infty$.)

Our goal will be for $f_m = m$. In general, for any i with $w_i = 1$, we would like $f_i = i$. If $f_i < i$, we say that we are $i - f_i$ steps behind at step i, and if $f_i > i$, we say that we are $f_i - i$ steps ahead at step i.

First, we note the following lemma, which states that we will never be ahead with very high probability, as long as the sequences $\mathbf{a}$ and $\mathbf{b}$ are similar enough.

Lemma 3.1. Set $C_1 = C_0/4$. Let $\mathbf{a}, \mathbf{b}$ be sequences of lengths m, m', respectively, where $m' \geq m$. Suppose that $|b_i - a_i| \leq C_1 \cdot \sqrt{b_i \log n}$ for all $0 \leq i < m$. Then, with probability at least $1 - \frac{1}{n^{10}}$ (over the randomness of the w_i), for all $0 \leq i \leq m$ with $w_i = 1$, $f_i \leq i$.

Proof. TOPROVE 0 $\qquad \square$

The main technical result will be showing that $f_m \geq m$ with reasonably high probability, i.e., with reasonably high probability we are not behind. This result will hold for *any* choice of $\mathbf{a}, \mathbf{b}$ and does not require any similarity between these sequences. In other words, our goal is to prove the following lemma.

Lemma 3.2. Let $\mathbf{a}, \mathbf{b}$ be strings of length at most n with every $\mathbf{a}_i, \mathbf{b}_j$ between L and n, where $L = C \cdot \log^8 n$ for a sufficiently large constant C. Define $m = |\mathbf{a}|$. Then, for any $\delta \leq \frac{1}{3 \cdot 10^6}$, with probability at least $1 - 200 \cdot \delta$ over the randomness of $w_1, \ldots, w_{m-1}$, $f_m \geq m$.

3.2   Proof of Lemma 3.2

In this section, we prove Lemma 3.2.

We will set a parameter $K = C_2 \log n$, where C_2 is a sufficiently large constant. For any $k \geq 0$, given the sequences $\mathbf{a} = a_0, \ldots, a_{m-1}$ and $\mathbf{b} = b_0, \ldots, b_{m'-1}$ (of possibly differing lengths), we define $p_k(\mathbf{a}, \mathbf{b})$ to be the probability (over the randomness of $w_1, \ldots, w_{m-1}$) that

7

- $f_m \leq m - k$.

- For any indices $0 \leq i \leq i' \leq m$ with $w_i, w_{i'} = 1$, $f_{i'} - f_i \geq (i' - i) - K$.

Equivalently, this is the same as the probability that we fall behind at least k steps from step 0 to step m, but we never fall behind $K + 1$ or more steps (relatively) from any (possibly intermediate) steps i to i'. For any $m \geq 1$, we define $p_k(m)$ to be the supremum value of $p_k(\mathbf{a}, \mathbf{b})$ over any sequences $\mathbf{a}, \mathbf{b}$ where $\mathbf{a}$ has length at most m and every a_i and b_j is between L and n, and we also define $p_k := \sup_{m \geq 1} p_k(m)$.

Note that for any $k > K$, $p_k(\mathbf{a}, \mathbf{b}) = 0$, as $f_m = m - k$ means $f_m - f_0 < (m - 0) - K$. So, $p_k(m)$ and p_k also equal 0 for any $k > K$.

First, we note a simple proposition, that will only be useful for simplifying the argument at certain places.

Proposition 3.3. For any $m \geq 1$, $p_0(m) = 1$.

Proof. TOPROVE 1 □

We now aim to bound the probabilities p_k for $k \leq K$. We will do this via an inductive approach on the length of m, where the high-level idea is that if we fall back by k steps, there is a natural splitting point where we can say first we fell back by k_1 steps, and then by k_2 steps, for some $k_1, k_2 > 0$ with $k_1 + k_2 = k$ – see Lemmas 3.5 and 3.6. This natural splitting point will be based on the structure of the similarity of $\mathbf{a}$ and $\mathbf{b}$, and will not work if $\mathbf{a}$ and $\mathbf{b}$ share a k-periodic structure. But in the periodic case, we can give a more direct argument that we cannot fall back by k steps (i.e., a full period), even with $\frac{1}{\text{poly}(n)}$ probability – see Lemma 3.4. We can then compute a recursive formula for the probability of falling back k steps, by saying we need to first fall back k_1 steps and then fall back k_2 steps. In Lemma 3.8, we bound the terms of this recursion.

Lemma 3.4. Fix any $m \geq k \geq 1$ such that $k \leq K$, and suppose that $L \geq C_3 \cdot \log^8 n$, where C_3 is a sufficiently large multiple of $C_0^2 \cdot C_2^6$. Suppose that $\mathbf{a}, \mathbf{b}$ are sequences such that for every $0 \leq i < m - k$, $|b_i - a_i| \leq C_0 \log n \cdot \sqrt{b_i}$ and $|b_i - a_{i+k}| \leq C_0 \log n \cdot \sqrt{b_i}$. Then, the probability $p_k(\mathbf{a}, \mathbf{b}) \leq (2\delta)^K$.

Proof. TOPROVE 2 □

Lemma 3.5. Fix any $m \geq k$ such that $k \leq K$, and suppose that $L \geq C_3 \cdot \log^2 n \cdot K^6$. Suppose that $\mathbf{a}, \mathbf{b}$ are sequences of length m, such that for every $0 \leq i < m - k$, $|b_i - a_{i+k}| \leq C_0 \log n \cdot \sqrt{b_i}$. Then, the probability

$$p_k(\mathbf{a}, \mathbf{b}) \leq (2\delta)^K + \sum_{\substack{h_1, h_2, k_2, k_2 \geq 0 \\ h_1 + h_2 + k_1 + k_2 \geq k \\ k_1, k_2 \leq K \\ (h_1, h_2, k_1, k_2) \neq (0,0,0,k),(0,0,k,0)}} \delta^{h_1 + h_2} p_{k_1}(m - 1) p_{k_2}(m - 1).$$

Proof. TOPROVE 3 □

Lemma 3.6. Fix any $m \geq k$ such that $k \leq K$, and suppose that $L \geq C_3 \cdot \log^2 n \cdot K^6$. Suppose that $\mathbf{a}, \mathbf{b}$ are sequences of length m. Then, the probability

$$p_k(\mathbf{a}, \mathbf{b}) \leq (2\delta)^K + \sum_{\substack{h_1,h_2,k_2,k_2 \geq 0 \\ h_1+h_2+k_1+k_2 \geq k \\ k_1,k_2 \leq K \\ (h_1,h_2,k_1,k_2) \neq (0,0,0,k),(0,0,k,0)}} \delta^{h_1+h_2} p_{k_1}(m-1) p_{k_2}(m-1).$$

Proof. TOPROVE 4 □

Overall, this implies that

$$p_k(m) \leq (2\delta)^K + 2 \cdot \sum_{\substack{h_1,h_2,k_2,k_2 \geq 0 \\ h_1+h_2+k_1+k_2 \geq k \\ k_1,k_2 \leq K \\ (h_1,h_2,k_1,k_2) \neq (0,0,0,k),(0,0,k,0)}} \delta^{h_1+h_2} p_{k_1}(m-1) p_{k_2}(m-1).$$

We now can universally bound p_k for all $0 \leq k \leq K$. To do so, we first recall some basic properties of the Catalan numbers.

Fact 3.7. For $n \geq 0$, the Catalan numbers $\mathfrak{C}_n$ [1] are defined as $\mathfrak{C}_n = \binom{2n}{n}/(n+1)$. They satisfy the following list of properties.

1. $\mathfrak{C}_0 = 1$ and for all $n \geq 0$, $\mathfrak{C}_{n+1} = \sum_{i=0}^{n} \mathfrak{C}_i \mathfrak{C}_{n-i}$.

2. For all $n \geq 1$, $2 \leq \frac{\mathfrak{C}_{n+1}}{\mathfrak{C}_n} \leq 4$.

3. For all $n \geq 0$, $\mathfrak{C}_n \leq 4^n$.

Lemma 3.8. Assume $\delta \leq \frac{1}{3 \cdot 100^3}$, and define $\mathfrak{D}_k := 100^{2k-1} \mathfrak{C}_k$ for $k \geq 1$ and $\mathfrak{D}_0 = 1$. Then, for all $0 \leq k \leq K$, $p_k \leq \mathfrak{D}_k \cdot \delta^k$.

Proof. TOPROVE 5 □

We are now ready to prove Lemma 3.2.

Proof. TOPROVE 6 □

4 Full algorithm/analysis

Let us depict the true string $x \in \{0,1\}^n$ as $\underbrace{0 \ldots 0}_{a_0 \text{ times}} 1 \underbrace{0 \ldots 0}_{a_1 \text{ times}} 1 \cdots 1 \underbrace{0 \ldots 0}_{a_t \text{ times}}$, i.e., there are $t-1$ ones, and the string starts and ends with a run of 0's. This assumption can be made WLOG by padding the string with L 0's at the front and the end. For any L-separated string, doing this padding maintains the L-separated property, and we can easily simulate the padded

---

[1] We use $\mathfrak{C}_n$ rather than the more standard C_n to avoid confusion with the constants $C_0, C_1, \ldots$ we have defined.

trace by adding $\text{Bin}(L, 1 - \delta)$ 0's at the front and $\text{Bin}(L, 1 - \delta)$ 0's at the back. Once we reconstruct the padded string, we remove the padding to get x.

We assume we know the value of t. Indeed, the number of 1's in a single trace $\tilde{x}$ is distributed as $\text{Bin}(t, 1 - \delta)$. So, by averaging the number of 1's over $O(n \log n)$ random traces and dividing by $1 - \delta$, we get an estimate of $t - 1$ that is accurate within 0.1 with $1 - \frac{1}{n^{10}}$ probability. Thus, by rounding, we know t exactly with $1 - \frac{1}{n^{10}}$ probability.

The main goal is now to learn the lengths $a_0, a_1, \ldots, a_t$. If we learn these exactly just using the traces, this completes the proof. Our algorithm runs in two phases: a coarse estimation phase and a fine estimation phase. In the coarse estimation phase, we sequentially learn each a_i up to error $O(\sqrt{a_i \log n})$. In the fine estimation phase, we learn each a_i exactly, given the coarse estimates.

4.1 Coarse estimation

Fix some $0 \leq m \leq t$, and suppose that for all $i < m$, we have estimates b_i satisfying $|b_i - (1 - \delta)a_i| \leq 10\sqrt{a_i}$. (If $m = 0$, then we have no estimates yet.) Our goal will be to provide an estimate b_m such that $|b_m - (1 - \delta)a_m| \leq 10\sqrt{a_m}$.

Consider a trace $\tilde{x}$ of x. Let $w_0 = w_{t+1} = 1$ and for each $1 \leq i \leq t$, let w_i be the indicator that the ith 1 is retained. Next, for each $0 \leq i \leq t$, let $\tilde{a}_i \sim \text{Bin}(a_i, 1 - \delta)$ represent the number of 0s in the ith run that were not deleted. Note that with at least 0.99 probability, $|\tilde{a}_i - (1 - \delta)a_i| \leq 10\sqrt{\log n \cdot a_i}$ for all i. Since $|b_i - (1 - \delta)a_i| \leq 10\sqrt{a_i}$ for all $i < m$, this implies that $|\tilde{a}_i - b_i| \leq 20\sqrt{\log n \cdot b_i}$ for all $i < m$.

Now, even though we have no knowledge of $\tilde{a}_i$ or a_i, we can still simulate the probabilistic process of Section 3. Let $0 = i_0 < i_1 < \cdots < i_h = t+1$ be the list of all indices $i : 0 \leq i \leq t+1$ with $w_i = 1$. While we do not know the values $\tilde{a}_i$, for every pair of consecutive indices i_q, i_{q+1}, the value $\tilde{a}_{i_q : i_{q+1}}$ is exactly the number of 0's between the qth and $(q + 1)$st 1 in the trace $\tilde{x}$ (where we say that the 0th 1 is at position 0 and the $(t + 1)$st 1 is at position $|\tilde{x}| + 1$). In other words, if r_q represents the position of the qth 1, then $\tilde{a}_{i_q : i_{q+1}} = r_{q+1} - r_q - 1$. Hence, because computing each $f_{i_{q+1}}$ only requires knowledge of $\mathbf{b}$ and the value of $\tilde{a}_{i_q : i_{q+1}}$, and since $f_{i_0} = f_0 = 0$, the algorithm can in fact compute $g_q := f_{i_q}$ for all $0 \leq q \leq h$, using the same process as described in Section 3, even if the values i_q are not known.

Algorithm 1 simulates this process, assuming knowledge of $m, b_0, \ldots, b_{m-1}$, a single trace $\tilde{x}$, and t. In Algorithm 1, we use the variable val to represent $g_q = f_{i_q}$, i.e., the current prediction of the position i_q. In other words, $\text{val} - i_q$ equals the number of steps ahead (or $i_q - \text{val}$ equals the number of steps behind) we are.

Lemma 4.1. Fix $b_0, \ldots, b_{m-1}$ such that $|b_i - (1 - \delta)a_i| \leq 10\sqrt{a_i}$ for all $0 \leq i \leq m - 1$. With probability at least 0.98 over the randomness of $\tilde{x}$, we have that Algorithm 1 returns q such that the qth 1 in $\tilde{x}$ corresponds to the mth 1 in x. Moreover, conditioned on this event holding, the distribution $r_{q+1} - r_q - 1$ exactly follows $\text{Bin}(a_m, 1 - \delta)$.

Proof. TOPROVE 7 □

Algorithm 1 Locate the mth and $(m+1)$st 1 in x, in the trace $\tilde{x}$, and return the position and length of the gap.

1: **procedure** $\text{ALIGN}(\tilde{x}, t, m, b_0, \ldots, b_{m-1})$
2:      Let r_q be the position of the qth 1 in $\tilde{x}$, for each $1 \le q \le t-1$.
3:      $r_0 \leftarrow 0$, $r_t \leftarrow |\tilde{x}| + 1$.
4:      $\text{val} \leftarrow 0$, $q \leftarrow 0$
5:      **while** $\text{val} < m$ **do**
6:          Find the smallest j' such that $\exists j, j'$ with $\text{val} \le j < j'$ and $|(r_{q+1} - r_q - 1) - b_{j:j'}| \le C_0 \log n \cdot \sqrt{b_{j:j'}}$.
7:          **if** no such j, j' exist **then**
8:              **Return FAIL**
9:          $\text{val} \leftarrow j'$
10:         $q \leftarrow q + 1$
11:      **if** $\text{val} = m$ **then**
12:          **Return** $(q, r_{q+1} - r_q - 1)$.
13:      **else**
14:          **Return FAIL**

Given this, we can crudely estimate every gap, in order. Namely, assuming that that we have estimates $b_0, \ldots, b_{m-1}$ (where $0 \le m \le t$), we can run the ALIGN procedure on $O(\log n)$ independent traces. By a Chernoff bound, with $\frac{1}{n^{15}}$ failure probability, at least 0.9 fraction of the traces will have the desired property of Lemma 4.1, so will output some (q, b) where $b \sim \text{Bin}(a_m, 1 - \delta)$. Since $\text{Bin}(a_m, 1 - \delta)$ is in the range $a_m(1 - \delta) \pm 10\sqrt{a_m}$ with at least 0.99 probability, at least 0.75 fraction of the outputs (q, b) will satisfy $|b - (1 - \delta)a_m| \le 10\sqrt{a_m}$, with $\frac{1}{n^{15}}$ failure probability. Thus, by defining b_m to be the median value of b across the randomly drawn traces, we have that $|b_m - (1 - \delta)a_m| \le 10\sqrt{a_m}$ with at least $1 - \frac{1}{n^{10}}$ probability.

By running this procedure iteratively to provide estimates $b_0, b_1, \ldots, b_t$, we obtain Algorithm 2. The analysis in the above paragraph implies the following result.

Theorem 4.2 (Crude Approximation). *Algorithm 2 uses $O(n \log n)$ traces and polynomial time, and learns estimates $b_0, b_1, \ldots, b_t$ such that with at least $1 - \frac{1}{n^9}$ probability, $|b_m - (1 - \delta)a_m| \le 10\sqrt{a_m}$ for all $0 \le m \le t$.*

4.2   Fine estimation

In this section, we show how to exactly compute each a_m with high probability, given the crude estimates $b_0, b_1, \ldots, b_{t-1}$. This will again be done using an alignment procedure, but this time running the alignment both "forward and backward".

Namely, given a trace $\tilde{x}$, we will try to identify the mth and $(m+1)$st 1's from the original string, but we try to identify the mth 1 by running ALIGN on $\tilde{x}$ and the $(m+1)$st 1 by running ALIGN on the reverse string $\text{rev}(\tilde{x}) := \tilde{x}_{|\tilde{x}|} \cdots \tilde{x}_2 \tilde{x}_1$. The idea is: assuming that we

---

Algorithm 2 Crude Estimation of all gaps

1: **procedure** CRUDE
2:     Use $O(n \log n)$ traces to compute t, where t equals the number of 0s in x.
3:     **for** $m = 0$ to t **do**
4:         **for** $i = 1$ to $O(\log n)$ **do**
5:             Draw trace $\tilde{x}^{(i)}$.
6:             $(q^{(i)}, b^{(i)}) \leftarrow \text{ALIGN}(\tilde{x}^{(i)}, t, m, b_0, \ldots, b_{m-1})$
7:         Let b_m be the median of $b^{(1)}, \ldots, b^{(O(\log n))}$ ▷ Some of the outputs $(q^{(i)}, b^{(i)})$ may be **FAIL**, but we can let $b^{(i)}$ be an arbitrary real number if ALIGN failed on $\tilde{x}^{(i)}$, so that the median is well-defined.
8:     **Return** $(b_0, b_1, \ldots, b_t)$

---

never go ahead in the alignment procedure, if we find some index q in the forward alignment procedure with $g_q = f_{i_q} = m$, then the true position i_q must be at least m. Likewise, if we do the alignment procedure in reverse until we believe we have found the $(t - m)$th 1 from the back (equivalently, the $(m + 1)$th 1 from the front), the true position must be at most $m + 1$.

So, the true positions of the index found in the forward alignment procedure can only be earlier than that of the index from the backward alignment procedure, if the true positions were exactly m and $m + 1$, respectively. Thus, by comparing the indices, we can effectively verify that the positions are correct, with negligible failure probability (rather than with $1 - O(\delta)$ failure probability). This is the key towards obtaining the fine estimate of a_m, rather than just a coarse estimate that may be off by $O(\sqrt{a_m})$.

Algorithm 3 formally describes the fine alignment procedure, using $N = O(n \log n)$ traces, assuming we have already done the coarse estimation to find $b_0, b_1, \ldots, b_t$.

Algorithm 3 Fine Estimation of all gaps

1: **procedure** FINE$(t, b_0, \ldots, b_t)$
2:     Draw $N = O(n \log n)$ traces $\tilde{x}^{(1)}, \ldots, \tilde{x}^{(N)}$.
3:     **for** $m = 0$ to t **do**
4:         Initialize $b^{(1)}, b^{(2)}, \ldots, b^{(N)} \leftarrow$ **NULL**.
5:         **for** $i = 1$ to N **do**
6:             $\tilde{m} \leftarrow$ number of 1's in $\tilde{x}$.
7:             $(q_{\text{f}}, b_{\text{f}}) \leftarrow \text{ALIGN}(\tilde{x}^{(i)}, t, m, b_0, b_1 \ldots, b_t)$.
8:             $(q_{\text{b}}, b_{\text{b}}) \leftarrow \text{ALIGN}(\text{rev}(\tilde{x}^{(i)}), t, t - m, b_t, b_{t-1}, \ldots, b_0)$.
9:             **if** $q_{\text{f}} + q_{\text{b}} = \tilde{m}$ **then**
10:                 $b^{(i)} \leftarrow b_{\text{f}}$
11:             Set a_m to be $\frac{1}{1-\delta}$ times the average of all non-null $b^{(i)}$'s, rounded to the nearest integer.
12:     **Return** $(a_0, a_1, \ldots, a_t)$

Lemma 4.3. Suppose that $|b_i - (1-\delta)a_i| \leq 10\sqrt{a_i}$ for all $1 \leq 0 \leq t$. Fix indices $0 \leq m \leq t$ and $1 \leq i \leq N$, and for simplicity of notation, let $\tilde{x} := \tilde{x}^{(i)}$. Let $\tilde{m}$ be the number of 1's in $\tilde{x}$. Then, the probability that $q_{\mathsf{f}} + q_{\mathsf{b}} = \tilde{m}$, but either the forward or backward iterations finds an index in $\tilde{x}$ which does not correspond to the mth 1 or $(m+1)$th 1, respectively, from x, is at most $2n^{-10}$. Moreover, if the forward and backward iterations find indices in $\tilde{x}$ corresponding to the mth 1 and $(m+1)$th 1, respectively, then $q_{\mathsf{f}} + q_{\mathsf{b}} = \tilde{m}$. Finally, the probability of finding both corresponding indices is at least 0.98.

Proof. TOPROVE 8 □

We are now ready to prove Theorem 1.1. Indeed, given the accuracy of the crude estimation procedure, it suffices to check that for each m, we compute a_m correctly, with at least $1 - n^{-5}$ probability.

Theorem 4.4 (Fine Estimation). Assume that t, the number of ones in x, is computed correctly, and for all $0 \leq m \leq t$, $|b_m - (1-\delta)a_m| \leq 10\sqrt{a_m}$.

Then, for any fixed $m : 0 \leq m \leq t$, with at least $1 - n^{-5}$ probability, we compute the gap a_m correctly.

Proof. TOPROVE 9 □

References

[BCF+19] Frank Ban, Xi Chen, Adam Freilich, Rocco A. Servedio, and Sandip Sinha. Beyond trace reconstruction: Population recovery from the deletion channel. In *Foundations of Computer Science (FOCS)*, pages 745–768, 2019.

[BCSS19] Frank Ban, Xi Chen, Rocco A. Servedio, and Sandip Sinha. Efficient average-case population recovery in the presence of insertions and deletions. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*, pages 44:1–44:18, 2019.

[BKKM04] Tugkan Batu, Sampath Kannan, Sanjeev Khanna, and Andrew McGregor. Reconstructing strings from random traces. In *Symposium on Discrete Algorithms (SODA)*, pages 910–918, 2004.

[BLS20] Joshua Brakensiek, Ray Li, and Bruce Spang. Coded trace reconstruction in a constant number of traces. In *Foundations of Computer Science (FOCS)*, 2020.

[CDK21] Diptarka Chakraborty, Debarati Das, and Robert Krauthgamer. Approximate trace reconstruction via median string (in average-case). In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 213 of *LIPIcs*, pages 11:1–11:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[CDL+21a] Xi Chen, Anindya De, Chin Ho Lee, Rocco A. Servedio, and Sandip Sinha. Polynomial-time trace reconstruction in the low deletion rate regime. In *Innovations in Theoretical Computer Science (ITCS)*, 2021.

[CDL+21b] Xi Chen, Anindya De, Chin Ho Lee, Rocco A. Servedio, and Sandip Sinha. Polynomial-time trace reconstruction in the smoothed complexity model. In *Symposium on Discrete Algorithms (SODA)*, 2021.

[CDL+22] Xi Chen, Anindya De, Chin Ho Lee, Rocco A. Servedio, and Sandip Sinha. Near-optimal average-case approximate trace reconstruction from few traces. In *Symposium on Discrete Algorithms (SODA)*, 2022.

[CDL+23] Xi Chen, Anindya De, Chin Ho Lee, Rocco A. Servedio, and Sandip Sinha. Approximate trace reconstruction from a single trace. In *Symposium on Discrete Algorithms (SODA)*, 2023.

[CGMR20] Mahdi Cheraghchi, Ryan Gabrys, Olgica Milenkovic, and João Ribeiro. Coded trace reconstruction. *IEEE Trans. Inf. Theory*, 66(10):6084–6103, 2020.

[Cha21a] Zachary Chase. New lower bounds for trace reconstruction. *Ann. Inst. H. Poincaré Probab. Statist.*, 57(2), 2021.

[Cha21b] Zachary Chase. Separating words and trace reconstruction. In *Symposium on Theory of Computing (STOC)*, 2021.

[CP21] Zachary Chase and Yuval Peres. Approximate trace reconstruction of random strings from a constant number of traces. *CoRR*, abs/2107.06454, 2021.

[DOS19] Anindya De, Ryan O'Donnell, and Rocco A. Servedio. Optimal mean-based algorithms for trace reconstruction. *Annals of Applied Probability*, 29(2):851–874, 2019.

[DRR19] Sami Davies, Miklos Racz, and Cyrus Rashtchian. Reconstructing trees from traces. In *Conference On Learning Theory (COLT)*, pages 961–978, 2019.

[DRSR21] Sami Davies, Miklós Z. Rácz, Benjamin G. Schiffer, and Cyrus Rashtchian. Approximate trace reconstruction: Algorithms. In *International Symposium on Information Theory (ISIT)*, pages 2525–2530. IEEE, 2021.

[HHP18] Lisa Hartung, Nina Holden, and Yuval Peres. Trace reconstruction with varying deletion probabilities. In *Analytic Algorithmics and Combinatorics (ANALCO)*, pages 54–61, 2018.

[HL20] Nina Holden and Russell Lyons. Lower bounds for trace reconstruction. *Annals of Applied Probability*, 30(2):503–525, 2020.

[HMPW08]   Thomas Holenstein, Michael Mitzenmacher, Rina Panigrahy, and Udi Wieder. Trace reconstruction with constant deletion probability and related results. In *Symposium on Discrete Algorithms (SODA)*, pages 389–398, 2008.

[HPP18]   Nina Holden, Robin Pemantle, and Yuval Peres. Subpolynomial trace reconstruction for random strings and arbitrary deletion probability. In *Conference On Learning Theory (COLT)*, pages 1799–1840, 2018.

[KM05]   Sampath Kannan and Andrew McGregor. More on reconstructing strings from random traces: insertions and deletions. In *International Symposium on Information Theory (ISIT)*, pages 297–301, 2005.

[KMMP21]   Akshay Krishnamurthy, Arya Mazumdar, Andrew McGregor, and Soumyabrata Pal. Trace reconstruction: Generalized and parameterized. *IEEE Trans. Inf. Theory*, 67(6):3233–3250, 2021.

[Lev01a]   Vladimir I. Levenshtein. Efficient reconstruction of sequences. *IEEE Trans. Information Theory*, 47(1):2–22, 2001.

[Lev01b]   Vladimir I. Levenshtein. Efficient reconstruction of sequences from their subsequences or supersequences. *J. Comb. Theory, Ser. A*, 93(2):310–332, 2001.

[MPV14]   Andrew McGregor, Eric Price, and Sofya Vorotnikova. Trace reconstruction revisited. In *European Symposium on Algorithms (ESA)*, pages 689–700, 2014.

[MS22]   Andrew McGregor and Rik Sengupta. Graph reconstruction from random subgraphs. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 229, pages 96:1–96:18, 2022.

[MS24]   Andrew McGregor and Rik Sengupta. Graph reconstruction from noisy random subgraphs. *CoRR*, abs/2405.04261, 2024.

[Nar21]   Shyam Narayanan. Improved algorithms for population recovery from the deletion channel. In *Symposium on Discrete Algorithms (SODA)*, pages 1259–1278. SIAM, 2021.

[NP17]   Fedor Nazarov and Yuval Peres. Trace reconstruction with $\exp(\text{o}(\text{n}^{1/3}))$ samples. In *Symposium on Theory of Computing (STOC)*, pages 1042–1046, 2017.

[NR21]   Shyam Narayanan and Michael Ren. Circular trace reconstruction. In *Innovations in Theoretical Computer Science (ITCS)*, 2021.

[PZ17]   Yuval Peres and Alex Zhai. Average-case reconstruction for the deletion channel: Subpolynomially many traces suffice. In *Foundations of Computer Science (FOCS)*, pages 228–239, 2017.

[Rub23]   Ittai Rubinstein. Average-case to (shifted) worst-case reduction for the trace reconstruction problem. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 261 of *LIPIcs*, pages 102:1–102:20, 2023.

[SY23]   Alec Sun and William Yue. The trace reconstruction problem for spider graphs. *Discrete Mathematics*, 346(1):113115, 2023.

[VS08]   Krishnamurthy Viswanathan and Ram Swaminathan. Improved string reconstruction over insertion-deletion channels. In *Symposium on Discrete Algorithms (SODA)*, pages 399–408, 2008.