Polynomial-Time Trace Reconstruction in the Low Deletion Rate Regime

Xi Chen
Columbia University, New York, NY, USA
http://www.cs.columbia.edu/~xichen
xichen@cs.columbia.edu

Anindya De
University of Pennsylvania, Philadelphia, PA, USA
https://www.seas.upenn.edu/~anindyad/
anindyad@cis.upenn.edu

Chin Ho Lee
Columbia University, New York, NY, USA
https://www.cs.columbia.edu/~chlee/
c.h.lee@columbia.edu

Rocco A. Servedio
Columbia University, New York, NY, USA
http://www.cs.columbia.edu/~rocco
rocco@cs.columbia.edu

Sandip Sinha
Columbia University, New York, NY, USA
https://sites.google.com/view/sandips
sandip@cs.columbia.edu

Abstract

In the *trace reconstruction problem*, an unknown source string $x \in \{0,1\}^n$ is transmitted through a probabilistic *deletion channel* which independently deletes each bit with some fixed probability δ and concatenates the surviving bits, resulting in a *trace* of x. The problem is to reconstruct x given access to independent traces. Trace reconstruction of arbitrary (worst-case) strings is a challenging problem, with the current state of the art for poly(n)-time algorithms being the 2004 algorithm of Batu et al. [2]. This algorithm can reconstruct an arbitrary source string $x \in \{0,1\}^n$ in poly(n) time provided that the deletion rate δ satisfies $\delta \leq n^{-(1/2+\varepsilon)}$ for some $\varepsilon > 0$.

In this work we improve on the result of [2] by giving a poly(n)-time algorithm for trace reconstruction for any deletion rate $\delta \leq n^{-(1/3+\varepsilon)}$. Our algorithm works by alternating an alignment-based procedure, which we show effectively reconstructs portions of the source string that are not "highly repetitive", with a novel procedure that efficiently determines the length of highly repetitive subwords of the source string.

2012 ACM Subject Classification Mathematics of computing → Probabilistic inference problems

Keywords and phrases trace reconstruction

Digital Object Identifier 10.4230/LIPIcs.ITCS.2021.20

Related Version Full Version: https://arxiv.org/abs/2012.02844

Funding *Xi Chen*: Supported by NSF grants CCF-1703925 and IIS-1838154.
Anindya De: Supported by NSF grants CCF-1926872 and CCF-1910534.
Chin Ho Lee: Supported by a grant from the Croucher Foundation and by the Simons Collaboration on Algorithms and Geometry.
Rocco A. Servedio: Supported by NSF grants CCF-1814873, IIS-1838154, CCF-1563155, and by the Simons Collaboration on Algorithms and Geometry.
Sandip Sinha: Supported by NSF grants CCF-1714818, CCF-1822809, IIS-1838154, CCF-1617955, CCF-1740833, and by the Simons Collaboration on Algorithms and Geometry.

1 Introduction

The *trace reconstruction* problem was proposed almost twenty years ago in works of [10, 11, 2], though some earlier variants of the problem were already considered in the 1970s [9]. This problem deals with the *deletion channel*, which works as follows: when an n-bit string (the source string) is passed through a deletion channel of rate δ, each coordinate is independently deleted with probability δ. The surviving $n' \leq n$ coordinates are concatenated to form the output of the channel, which is referred to as a *trace* of the original source string; we write "$\boldsymbol{z} \sim \mathrm{Del}_\delta(x)$" to indicate that $\boldsymbol{z}$ is a trace generated from source string x according to this probabilistic process. As discussed in [13], this channel provides an elegant formalization for the theoretical study of problems involving synchronization errors.

In the *trace reconstruction problem*, independent traces are generated from an unknown and arbitrary source string $x \in \{0,1\}^n$, and the task of the algorithm is to reconstruct (with high probability) x from its traces. The trace reconstruction problem is motivated by applications in several domains, including sensor networks and biology [13, 1, 16, 15]. It is also attractive because it is a clean and natural "first problem" which already seems to capture much of the difficulty of dealing with the deletion channel.

The problem of trace reconstruction for an arbitrary (worst-case) source string x has proved to be quite challenging.[1] [2] gave an algorithm that runs in $\mathrm{poly}(n)$ time, uses $\mathrm{poly}(n)$ traces, and with high probability reconstructs an arbitrary source string $x \in \{0,1\}^n$ provided that the deletion rate δ is at most $n^{-(1/2+\varepsilon)}$ for some constant $\varepsilon > 0$. Unfortunately, the trace reconstruction problem seems to quickly become intractable at higher deletion rates. Holenstein et al. [8] gave an algorithm that runs in time $\exp(O(n^{1/2}))$ and uses $\exp(O(n^{1/2}))$ traces for any deletion rate δ that is bounded away from 1 by a constant, and this result was subsequently improved in simultaneous and independent works by [4, 14], both of which gave algorithms with time and sample complexity $\exp(O(n^{1/3}))$. On the lower bounds side, for $\delta = \Theta(1)$ successively stronger lower bounds on the required sample complexity were given by [12] and [5], with the current state of the art being a $\tilde{\Omega}(n^{3/2})$ lower bound due to Chase [3].

The low deletion rate regime. The positive result of [4] actually gives an algorithm that is faster than $\exp(O(n^{1/3}))$ if the deletion rate is sufficiently low: [4] shows that for $O(\log^3 n)/n \leq \delta \leq 1/2$, their algorithm runs in time $\exp(O(\delta n)^{1/3})$. Consequently, for the specific deletion rate $\delta = n^{-(1/2+\varepsilon)}$, the [4] algorithm runs in time essentially $\exp(O(n^{1/6}))$, and [4] shows that no faster running time or better sample complexity is possible for any "mean-based" algorithm, a class of algorithms which includes those of [4, 14, 8].

Algorithmic approaches other than mean-based algorithms can provably do better at low deletion rates. This is witnessed by the algorithm of Batu et al. [2] which, as described earlier, runs in $\mathrm{poly}(n)$ time and uses $\mathrm{poly}(n)$ samples at deletion rate $\delta = n^{-(1/2+\varepsilon)}$. The main algorithmic component of [2] is a "*Bitwise Majority Alignment*" (`BMA` for short) procedure, which is further augmented with a simple procedure to determine the length of long "runs" (subwords of x of the form 0^ℓ or 1^ℓ with $\ell \geq \sqrt{n}$). Roughly speaking, the `BMA` algorithm maintains a pointer in each trace and increments those pointers in successive time steps, attempting to always keep almost all of the pointers correctly aligned together. The analysis of [2] shows that the `BMA` algorithm succeeds if the source string x does not contain any

[1] We note that the average-case problem, in which the reconstruction algorithm is only required to succeed for a $1 - o(1)$ fraction of all possible source strings in $\{0,1\}^n$, is much more tractable, with the current state of the art [6, 7] being an algorithm that uses $\exp(O(\log^{1/3} n))$ traces and runs in $\mathrm{poly}(n)$ time for any deletion rate δ that is bounded away from 1.

long runs, but a challenge for the BMA algorithm is that the pointers in different traces will inevitably become misaligned if x does contain a long run 0^ℓ or 1^ℓ; this is why the [2] algorithm must interleave BMA with a procedure to handle long runs separately. Intuitively, deletion rate $\delta = n^{-1/2}$ is a barrier for the [2] analysis because if $\delta = \omega(n^{-1/2})$, then each trace is likely to have multiple locations where more than one consecutive bit of x is "dropped," which is problematic for the analysis of BMA given in [2].

To summarize: given this state of the art from prior work, it is clear that alignment-based approaches can outperform mean-based algorithms at low deletion rates, but it is not clear whether, or how far, alignment-based approaches can be extended beyond the [2] results. Further incentive for studying the low deletion rate regime comes from potential applications in areas such as computer networks, where it may be natural to model deletions as occurring at relatively low rates. These considerations motivate the results of the present paper, which we now describe.

1.1 This work: An improved algorithm for the low deletion rate regime

The main result of this paper is an efficient algorithm that can handle significantly higher deletion rates than the [2] algorithm. We prove the following:

▶ **Theorem 1** (Efficient trace reconstruction at deletion rate $\delta \geq n^{-(1/3+\varepsilon)}$). *Fix any constant $\varepsilon > 0$ and let $\delta = n^{-(1/3+\varepsilon)}$. There is an algorithm* Reconstruct *that uses $O(n^{4/3})$ independent traces drawn from $\mathrm{Del}_\delta(x)$ (where $x \in \{0,1\}^n$ is arbitrary and unknown to* Reconstruct*), runs in $O(n^{7/3})$ time, and outputs the unknown source string x with probability at least $9/10$.*

Note that any deletion rate $\delta < n^{-(1/3+\varepsilon)}$ can of course be handled, given Theorem 1, by simply deleting additional bits to reduce to the $\delta = n^{-(1/3+\varepsilon)}$ case. Note further that any desired success probability $1 - \kappa$ can easily be achieved from Theorem 1 by running Reconstruct $O(\log(1/\kappa))$ times and then taking a majority vote.

At a high level, the Reconstruct algorithm works by interleaving two different subroutines.

- The first subroutine is (essentially) the BMA algorithm, for which we provide an improved analysis, showing that BMA successfully reconstructs any string that does not contain a long subword (of length at least $M = 2m + 1$ with $m = n^{1/3}$) that is a prefix of s^∞ for some short (constant-length) bitstring s. We refer to long and "highly-repetitive" subwords of x of this form as "s-*deserts*" of x; see Definition 2 for a detailed definition.
- The second subroutine is a new algorithm which we show efficiently determines the length of an s-desert in the source string x.

Thus, two novel aspects of this work that go beyond [2] are (i) our improved analysis of BMA, and (ii) our new procedure for efficiently measuring deserts (the analogous component of the [2] algorithm could only measure runs, which correspond to s-deserts with $|s| = 1$).

We believe that it may be possible to further extend the kind of "hybrid" approach that we employ in this paper to obtain efficient trace reconstruction algorithms that can handle even larger deletion rates δ. However, there are some significant technical challenges that would need to be overcome in order to do so. We describe some of these challenges at the end of the next section, which gives a more detailed overview of our approach.

2 Overview of our approach

As alluded to in the introduction, at a high level our algorithm carries out a careful interleaving of two procedures, which we call BMA and FindEnd. In this section we first give a high-level overview of the procedure BMA as well as our improved analysis. Then we give a high-level overview of FindEnd, and finally we explain how these two procedures are interleaved. We close with a brief discussion of possibilities and barriers to further progress.

2.1 Overview of BMA

The procedure BMA is exactly the same as the bitwise majority alignment algorithm of [2]; our new contribution regarding BMA is in giving a more general analysis. To explain the high level idea, let us fix the deletion rate $\delta = n^{-(1/3+\varepsilon)}$ and a constant $C = \lceil 100/\varepsilon \rceil$. Let

$$m = n^{1/3} \quad \text{and} \quad M = 2m + 1.$$

The BMA procedure operates on a sample of some $N = O(\log n)$ traces $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N$ drawn independently from $\mathrm{Del}_\delta(x)$ before the procedure begins its execution. Note that for any $i \in [N]$ and any position ℓ in the trace $\boldsymbol{y}^i$, there is a position $f_i(\ell)$ satisfying $\ell \le f_i(\ell) \le n-1$ in the target string $x = (x_0, \ldots, x_{n-1})$ that maps to ℓ under the deletion process.[2] The high level idea of BMA is to maintain pointers $\mathrm{current}_1, \ldots, \mathrm{current}_N$, with $\mathrm{current}_i$ pointing to a position in $\boldsymbol{y}^i$, such that *most of them are correctly aligned* – i.e., at the beginning of each time step t, $t = 0, 1, \ldots$, as we try to determine x_t, we have $f_i(\mathrm{current}_i) = t$ for most $i \in [N]$. Note that if this alignment guarantee were to hold for more than half of the traces for all $t = 0, 1, \ldots, n-1$, then we could reconstruct the unknown string x by taking a majority vote of $\boldsymbol{y}^i_{\mathrm{current}_i}$ in each time step. Indeed we show that this happens with high probability over the randomness of $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N$ when x does not contain an s-desert for any string $s \in \{0,1\}^{\le C}$ (i.e. a subword of length at least $M = 2m+1$ that is a prefix of s^∞). In contrast, the analysis of [2] requires the deletion rate to be $n^{-(1/2+\varepsilon)}$ but works as long as x does not contain a run of 0's or 1's (or s-deserts with $|s| = 1$ in our notation) of length at least $\sqrt{n}$.

To explain BMA in more detail, let us initialize $t = 0$ and pointers $\mathrm{current}_1(0), \ldots, \mathrm{current}_N(0)$ to position 0. (Note that most pointers are correctly aligned as desired given that $\delta = n^{-(1/3+\varepsilon)}$ and thus, x_0 is not deleted in most traces and $f_i(\mathrm{current}_i(0)) = f_i(0) = 0$ for most i.) The way the pointers are updated is as follows: At each time step t, we let w_t be the majority element of the N-element multiset $\{\boldsymbol{y}^i_{\mathrm{current}_i(t)}\}_{i\in[N]}$. For those traces $\boldsymbol{y}^i$ with $\boldsymbol{y}^i_{\mathrm{current}_i(t)} = w_t$ (i.e., the bit of $\boldsymbol{y}^i$ at the current pointer is the majority bit), we move the pointer to the right by 1, i.e. we set $\mathrm{current}_i(t+1) \leftarrow \mathrm{current}_i(t)+1$; otherwise the pointer stays the same, i.e., we set $\mathrm{current}_i(t+1) \leftarrow \mathrm{current}_i(t)$. Next we increment t and start the next round, repeating until $t = n$ when BMA outputs the string $(w_0, \ldots, w_{n-1})$.

For intuition we observe that if most of the pointers were aligned at the beginning of time step t (i.e., $f_i(\mathrm{current}_i(t)) = t$ for most $i \in [N]$), then $w_t = x_t$ is indeed the next bit in x. Moreover, if $\mathrm{current}_i(t)$ is aligned and $w_t = x_t$, then moving $\mathrm{current}_i$ to the right by 1 is justified by noting that most likely x_{t+1} is not deleted in $\boldsymbol{y}^i$ (with probability $1-\delta$), and when this happens $f_i(\mathrm{current}_i(t+1)) = t+1$ so $\mathrm{current}_i$ remains aligned at the beginning of the next time step.

In more detail, our analysis shows that when x does not contain an s-desert for any $s \in \{0,1\}^{\le C}$ BMA maintains the following invariants at the beginning of time step $t = 0, 1, \ldots, n$:

1. At time t, BMA has reconstructed $x_0, \ldots, x_{t-1}$ correctly as $w_0, \ldots, w_{t-1}$.
2. For every trace $\boldsymbol{y}^i$, $i \in [N]$, it holds that $f_i(\mathrm{current}_i(t)) \ge t$.
3. Finally, $\sum_{i\in[N]} \big(f_i(\mathrm{current}_i(t)) - t\big) \le 2N/C$.

The intuitive meaning behind conditions (2) and (3) is as follows: while (2) says that the "original position" of $\mathrm{current}_i(t)$ never falls behind t, condition (3) ensures that on average, the original positions of these pointers do not surpass t by too much. In fact, since C is a large constant, most of the pointers are perfectly aligned, i.e., they satisfy $f_i(\mathrm{current}_i(t)) = t$.

[2] It will be convenient for us to index a binary string $x \in \{0,1\}^\ell$ using $[0 : \ell-1]$ as $x = (x_0, \ldots, x_{\ell-1})$.

We now discuss how the invariants (1), (2) and (3) are maintained. First, we observe that invariant (1) for time step $t+1$, i.e., $w_t = x_t$, follows immediately from (3) at time step t. Invariant (2) for time step $t+1$ follows almost immediately from (2) at t and $w_t = x_t$. (If $f_i(\text{current}_i(t)) > t$, then $f_i(\text{current}_i(t+1)) \geq f_i(\text{current}_i(t)) \geq t+1$ given that both f_i and current_i are nondecreasing; if $f_i(\text{current}_i(t)) = t$ is aligned at time step t, then $w_t = x_t$ implies $\text{current}_i(t+1) = \text{current}_i(t) + 1$ and thus, $f_i(\text{current}_i(t+1)) \geq t+1$.) The main challenge is to show that invariant (3) is maintained. While this is not true for a general string x, we show that this holds with high probability (over $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N \sim \text{Del}_\delta(x)$) for any string x which does not have an s-desert for any $s \in \{0,1\}^{\leq C}$. (We note here that the value of m is selected so as to satisfy $m\delta \ll 1$; on the other hand, when we discuss the `FindEnd` procedure below, we will see that we also require m to satisfy $m \gg \sqrt{\delta n}$.)

In a nutshell, the main proof idea for (3) is to exploit the fact that when we draw $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N \sim \text{Del}_\delta(x)$, with high probability they satisfy two properties: (i) for every $\boldsymbol{y}^i$ and every subword of roughly $C^2 m$ consecutive positions in the original string x, no more than C positions within the subword are deleted in the generation of $\boldsymbol{y}^i$; (ii) for every subword of roughly m consecutive positions in x, the number of $\boldsymbol{y}^i$ that have at least one deletion in the subword is no more than N/C^3. These two properties can be shown using straightforward probabilistic arguments by taking advantage of the aforementioned $m\delta \ll 1$. Using these two properties, a detailed (non-probabilistic) argument shows that `BMA` can reconstruct the string x with high probability if x contains no s-desert.

The above discussion sketches our argument that if the target string x does not have an s-desert, then `BMA` correctly reconstructs x. More generally, our arguments show that if x does have an s-desert, then `BMA` correctly reconstructs the prefix of x up to the position when an s-desert shows up: Let r be the first position in x that is "*deep in an s-desert*"; this is the first position in x such that $x_{[r-m:r+m]}$[3], the length-M subword of x centered at r, is an s-desert. Then `BMA` correctly reconstructs the prefix of x up to position $r+m$. Having reached such a position, it is natural to now ask – "how do we determine the *end* of this desert?". This naturally leads us to the next procedure `FindEnd`.

2.2 Overview of `FindEnd`

Suppose that x has an s-desert with $|s| = k \leq C$, so `BMA` reconstructs the length-$(r+m+1)$ prefix of x, where r is the first position that is "deep in the s-desert" (note that it is easy to determine the position r from the output of `BMA`). The algorithm `FindEnd` takes as input the prefix $x_{[0:r+m]}$ of x and the location r, and its task is to compute the end of the s-desert: the first position $\mathsf{end} \geq r+m$ such that $x_{\mathsf{end}+1} \neq x_{\mathsf{end}-k+1}$. The `FindEnd` algorithm is rather involved but at a high level it consists of two stages: an initial *coarse estimation* of the end of the desert followed by *alignments* of traces from $\text{Del}_\delta(x)$ with the end of the desert (using the coarse estimate).

Coarse estimation: The goal of the coarse estimation stage is to identify an integer $\widehat{\beta}$ that is close to $(1-\delta)\mathsf{end}$: $|\widehat{\beta} - (1-\delta)\mathsf{end}| \leq 2\sigma$, where $\sigma := \tilde{O}(\sqrt{\delta n}) \ll m$ is basically how far an entry x_i of x can deviate from its expected location $(1-\delta)i$ in a typical trace $\boldsymbol{y} \sim \text{Del}_\delta(x)$. Intuitively, $\widehat{\beta}$ is an estimation of the location of x_{end} in a trace $\boldsymbol{y} \sim \text{Del}_\delta(x)$ that contains it, i.e., when x_{end} is not deleted in $\boldsymbol{y}$. To do this, we draw $\alpha = O(1/\varepsilon)$ many traces $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^\alpha \sim \text{Del}_\delta(x)$. Roughly speaking, we split each trace $\boldsymbol{y}^i$ into overlapping intervals of

[3] For a string $x \in [0:n-1]$ integers $0 \leq a < b \leq n-1$, we write $x_{[a:b]}$ to denote the *subword* $(x_a, x_{a+1}, \ldots, x_b)$.

length 4σ. The first interval starts at $(1-\delta)r$ and each successive interval shifts to the right by σ (so it overlaps with the previous interval by 3σ). Since $m \gg \sigma = \tilde{O}(\sqrt{\delta n})$ (which is one of the bottlenecks that requires $\delta \ll n^{-1/3}$), the s-desert is unlikely to end before $(1-\delta)r$ in a trace $\boldsymbol{y} \sim \mathrm{Del}_\delta(x)$ and must end in one of constantly many intervals with high probability, by the choice of σ. To identify one such interval, we make the following observation. Let Cyc_s be the set of all k-bit strings that can be obtained as cyclic shifts of s. Given end as the end of the s-desert that starts at x_{r-m}, every k-bit subword of $x_{[r-m:\mathsf{end}]}$ is in Cyc_s but $(x_{\mathsf{end}-k+2}, \ldots, x_{\mathsf{end}+1}) \notin \mathrm{Cyc}_s$, and these $k \leq C$ bits will most likely remain in a trace given the low deletion rate. This motivates us to look for the leftmost interval I^* such that in at least half of $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^\alpha$, it holds that $\boldsymbol{y}^i_{I^*}$ contains a k-bit subword not in Cyc_s. We show that with high probability, setting $\widehat{\beta}$ to be the right endpoint of I^* gives us a coarse estimate of $(1-\delta)\mathsf{end}$ up to an accuracy of $\pm 2\sigma$.

In addition to obtaining $\widehat{\beta}$, the coarse estimation stage recovers the following 8σ-bit subword of x: $(x_{\mathsf{end}-k+2}, \ldots, x_{\mathsf{end}-k+8\sigma+1})$, which we will refer to as the *tail string* of the s-desert and denote by $\mathsf{tail} \in \{0,1\}^{8\sigma}$. To this end, we draw another $\alpha = O(1/\varepsilon)$ fresh traces $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^\alpha$ and examine the subword of each $\boldsymbol{y}^i$ of length 6σ centered at location $\widehat{\beta}$. Each $\boldsymbol{y}^i$ looks for the first k-bit subword in this interval that is not in Cyc_s and votes for its 8σ-bit subword that starts at this non-cyclic shift as its candidate for tail. We show that with high probability, the string with the highest votes is exactly tail. (We note that both parts of this coarse estimation procedure require that with high probability, any fixed interval of length $O(\sigma)$ in x does not get any deletions in a random trace, i.e., $\sigma\delta \ll 1$. This follows from the two constraints $m\delta \ll 1$ and $m \gg \sigma$.)

Alignments: Suppose the first stage succeeds in computing $\widehat{\beta}$ and $\mathsf{tail} \in \{0,1\}^{8\sigma}$. The second stage is based on a procedure called `Align` which satisfies two crucial criteria. These criteria are as follows: if `Align` is given an input trace $\boldsymbol{y} \sim \mathrm{Del}_\delta(x)$, then (a) with *fairly high* probability (by which we mean $1 - n^{-\Theta(\varepsilon)}$ for the rest of the overview) it returns a location ℓ in $\boldsymbol{y}$ such that $\boldsymbol{y}_\ell$ corresponds to x_{end} in x, and moreover (b) the expectation of ℓ (over the randomness of $\boldsymbol{y}$) is a "sharp estimate" of $(1-\delta)\mathsf{end}$ that is accurate up to an additive ± 0.1 error.[4] To pin down the exact end of the s-desert, `FindEnd` simply draws $\tilde{O}(n^{2/3-\varepsilon})$ many traces, runs `Align` on each of them and computes the average $\widehat{\ell}$ of the locations it returns. It is easy to show that rounding $\widehat{\ell}/(1-\delta)$ to the nearest integer gives end with high probability.

The case when $k = |s| = 1$ (so the desert is a long subword consisting either of all 0's or of all 1's) is significantly easier (and was implicitly handled in [2]), so in the following discussion we focus on the case when $k = |s| \geq 2$ and the desert has a more challenging structure. For this case our `Align` procedure uses a new idea, which is that of a "*signature*." A *signature* is a subword of x, denoted sig, of length between $2k$ and 8σ that starts at the same location $x_{\mathsf{end}-k+2}$ as tail (so sig is contained in tail, since $|\mathsf{tail}| = 8\sigma$) and either ends at a location d which is the smallest integer $d \in [\mathsf{end}+k+1 : \mathsf{end}+8\sigma-k+1]$ such that the k-bit subword that ends at d is not in Cyc_s, or has length 8σ if no such d exists (in this case sig is the same as tail). We remind the reader that the first k-bits of tail, and hence also of sig, is a string not in Cyc_s, and the same is true of the last k bits of sig if its length is less than 8σ.

[4] We note that item (b) is not an immediate consequence of item (a). In more detail, the failure probability of (a) is roughly $1/n^{\Theta(\varepsilon)}$, but if when `Align` fails in (a) it returns a location that is inaccurate by $\gg n^{\Theta(\varepsilon)}$ positions, then (b) would not follow from (a). Indeed significantly more effort is required in our analysis to ensure (b).

Given a trace $\boldsymbol{y} \sim \mathrm{Del}_\delta(x)$, `Align` (roughly speaking) attempts to locate the image of x_{end} in $\boldsymbol{y}$ by locating the image of sig within an interval in $\boldsymbol{y}$ of length $O(\sigma)$ around $\widehat{\beta}$. In a bit more detail, it checks whether the restriction of $\boldsymbol{y}$ to a certain interval J around $\widehat{\beta}$ is of the form $w \circ \mathsf{sig} \circ v$, such that the first k bits of sig is the leftmost k-bit subword of $\boldsymbol{y}_J$ that is not in Cyc_s. If $\boldsymbol{y}$ does not satisfy this condition then `Align` discards that trace and outputs nil. We note that if the only goal of `Align` were to locate a position ℓ in $\boldsymbol{y}$ such that with fairly high probability $\boldsymbol{y}_\ell$ corresponds to x_{end} (i.e. item (a) above), then in all other cases (i.e. whenever $\boldsymbol{y}$ does satisfy the above condition) `Align` could return the index of the $(k-1)$-th bit of sig in $\boldsymbol{y}_J$. (By doing this, `Align` always returns the correct position whenever the subword of x of length $O(\sigma)$ centered at end has no deletion in $\boldsymbol{y}$ and x_{end} deviates from its expected location in a trace by at most σ in $\boldsymbol{y}$, which happens with probability $O(\sigma\delta) = n^{-\Theta(\varepsilon)}$.) However, it turns out that `Align` must proceed in a slightly (but crucially) different way in order to additionally satisfy item (b) above (i.e., have the expected value of its output locations provide an accurate "sharp estimate" of $(1-\delta)\mathsf{end}$). The actual execution of `Align` is that in the case when $\boldsymbol{y}_J$ does satisfy the above condition, `Align` returns the index of the $(k-1)$-th bit of sig in $\boldsymbol{y}_J$ with high probability and with some small remaining probability (the precise value of which depends on the location of sig within $\boldsymbol{y}_J$), `Align` opts to still output nil. A detailed analysis, which we provide in Section 6.2.2, shows that this `Align` procedure satisfies both criteria (a) and (b) described above.

2.3 The overall algorithm

The overall algorithm works by alternately running `BMA` and `FindEnd`. It starts with `BMA`, which draws $N = O(\log n)$ traces of x and returns the first position r in x that is deep in a desert as well as the prefix $w = x_{[0:r+m]}$ of the target string x. Then the algorithm runs `FindEnd` to compute end, the right end of the desert. Note that the execution of `BMA` will misalign some small fraction of the traces it uses, but these errors do not affect `FindEnd` as `FindEnd` is run using fresh traces.

With end from `FindEnd`, the algorithm has now reconstructed the prefix $x_{[0:\mathsf{end}]}$ by extending $x_{[0:r+m]}$. Next the algorithm runs `BMA` again on N traces that are, ideally, drawn from $x_{[\mathsf{end}+1:n-1]}$, in order to reconstruct the next segment of x until a new desert shows up (at which point the algorithm repeats). These traces are obtained by running the `Align` procedure used by `FindEnd` on N fresh traces $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N$ of x. Let ℓ_i be the output of `Align` running on $\boldsymbol{y}^i$. As noted in (a) earlier, all but a small fraction of ℓ_i's are such that the desert ends at $\boldsymbol{y}^i_{\ell_i}$ in $\boldsymbol{y}^i$. We then run `BMA` on $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^N$, where $\boldsymbol{z}^i$ is the suffix of $\boldsymbol{y}^i$ starting at $\ell_i + 1$ for each i. Even though $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^N$ are *not* exactly N fresh traces of $x_{[\mathsf{end}+1:n-1]}$ (since a small and arbitrary fraction of $\boldsymbol{y}^i$ might be misaligned), `BMA` is able to succeed because of a crucial *robustness* property. This property is that the correctness guarantee of `BMA` holds even when a small and "adversarially" picked constant fraction of the N traces given to it are misaligned; intuitively, `BMA` enjoys this robustness because it works in each time step by taking a majority vote over its input traces, so as long as a substantial majority of the traces are correctly aligned, even a small constant fraction of adversarial traces cannot affect its correctness. The algorithm continues alternating between `BMA` and `FindEnd`, and is thereby able to reconstruct the entire target string x.

3 Preliminaries

Notation. Given a positive integer n, we write $[n]$ to denote $\{1, \ldots, n\}$. Given two integers $a \le b$ we write $[a : b]$ to denote $\{a, \ldots, b\}$. We write ln to denote natural logarithm and log to denote logarithm to the base 2. We denote the set of non-negative integers by $\mathbb{Z}_{\ge 0}$. We write "$a = b \pm c$" to indicate that $b - c \le a \le b + c$.

Subword. It will be convenient for us to index a binary string $x \in \{0,1\}^n$ using $[0 : n-1]$ as $x = (x_0, \ldots, x_{n-1})$. Given such a string $x \in \{0,1\}^n$ and integers $0 \le a \le b \le n-1$, we write $x_{[a:b]}$ to denote the *subword* $(x_a, x_{a+1}, \ldots, x_b)$ of x. An *ℓ-subword* of x is a subword of x of length ℓ, given by $(x_a, x_{a+1}, \ldots, x_{a+\ell-1})$ for some $a \in [0 : n-\ell]$.

Distributions. When we use bold font such as $\mathbf{D}, \boldsymbol{y}, \boldsymbol{z}$, etc., it is to emphasize that the entity in question is a random variable. We write "$\boldsymbol{x} \sim \mathcal{D}$" to indicate that random variable $\boldsymbol{x}$ is distributed according to distribution $\mathcal{D}$.

Deletion channel and traces. Throughout this paper the parameter $\delta : 0 < \delta < 1$ denotes the *deletion probability*. Given a string $x \in \{0,1\}^n$, we write $\mathrm{Del}_\delta(x)$ to denote the distribution of the string that results from passing x through the δ-deletion channel (so the distribution $\mathrm{Del}_\delta(x)$ is supported on $\{0,1\}^{\le n}$), and we refer to a string in the support of $\mathrm{Del}_\delta(x)$ as a *trace* of x. Recall that a random trace $\boldsymbol{y} \sim \mathrm{Del}_\delta(x)$ is obtained by independently deleting each bit of x with probability δ and concatenating the surviving bits. [5]

A notational convention. In several places we use sans serif font for names such as tail (which is a subword of the target string x), end (which is a location in the target string x), and so on. To aid the reader, whenever we use this font the corresponding entity is an "x-entity," i.e. a location, subword, etc. that is associated with the source string x rather than with a trace of x.

4 The main algorithm

In this section we describe the main algorithm `Reconstruct`. We begin by giving a precise definition of the notion of an *s-desert*. To do this, here and throughout the paper we fix

$$C := \lceil 100/\varepsilon \rceil, \quad \text{and we recall that} \quad m = n^{1/3} \quad \text{and} \quad M = 2m+1.$$

▶ **Definition 2.** *For $s \in \{0,1\}^{\le C}$, a binary string $z \in \{0,1\}^*$ is said to be an* s-desert *if z is a prefix of s^∞ and $|z| \ge M$. A string is said to be a* desert *if it is an s-desert for some $s \in \{0,1\}^{\le C}$. Given a string $x \in \{0,1\}^n$, we say that a location $i \in [0 : n-1]$ is* deep in a desert *if the length-M subword $x_{[i-m:i+m]}$ centered at i is a desert. We say a string x has* no desert *if no subword of x is a desert (or equivalently, no location $i \in [0 : n-1]$ is deep in a desert in x); otherwise we say that it has at least one desert.*

4.1 The preprocessing step

Before stating the main algorithm, we first describe a simple preprocessing step.

▶ **Lemma 3.** *There is a randomized algorithm* `Preprocess` *which satisfies the following with probability $1 - n^{-\omega(1)}$ (over its internal randomness):*

1. *It outputs a string $v \in \{0,1\}^{n/2}$.*
2. *For any unknown string $x \in \{0,1\}^n$, given access to a sample from $\mathrm{Del}_\delta(x)$, it can output a sample from $\mathrm{Del}_\delta(z)$, where $z = x \circ v$, in linear time.*
3. *For any $s \in \{0,1\}^{\le C}$, the string v does not have a s-desert. Consequently, any desert in the string $z = x \circ v$ ends at least $n/2 - (2m+1)$ bits before the end of z.*

[5] For simplicity in this work we assume that the deletion probability δ is known to the reconstruction algorithm. We note that it is possible to obtain a high-accuracy estimate of δ simply by measuring the average length of traces received from the deletion channel.

Algorithm 1 Algorithm `Reconstruct` for $\delta = n^{-(1/3+\varepsilon)}$.

Input: Length n of an unknown $x \in \{0,1\}^n$ and access to $\mathrm{Del}_\delta(x)$ where $\delta = n^{-(1/3+\varepsilon)}$

Output: A string u, where the algorithm succeeds if $u = x$

1 Set $N := O(\log n)$
2 Draw N fresh traces $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^N$ independently from $\mathrm{Del}_\delta(x)$
3 Run $\texttt{BMA}(n, \{\boldsymbol{z}^1, \ldots, \boldsymbol{z}^N\})$ and let w be its output
4 **if** *w has no desert* **then return** w
5 **else**
6 Let r be the first location that is deep in a desert in w and let $u = w_{[0:r+m]}$
// `Main loop`
7 **for** *n/m rounds* **do**
8 Draw N fresh traces $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N$ independently from $\mathrm{Del}_\delta(x)$
9 Run $\texttt{FindEnd}(r, u, \{\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N\})$ and let b and ℓ_i, $i \in [N]$, be its output
10 Set $r = b$ and extend u to be a string of length b such that $u_{[r-m:b]}$ is a desert
11 **if** $b = n - 1$ **then** output "*FAILURE*"
12 Let $\boldsymbol{z}^i$ be the suffix of $\boldsymbol{y}^i$ starting at $\boldsymbol{y}^i_{\ell_i+1}$ for each $i \in [N]$
13 Run $\texttt{BMA}(n - b - 1, \{\boldsymbol{z}^1, \ldots, \boldsymbol{z}^N\})$ and let w be its output
14 **if** *w has no desert* **then**
15 **return** $u \circ w$
16 **else**
17 Let r^* be the first location that is deep in a desert in w and set $r \leftarrow r + r^*$ and $u \leftarrow u \circ w_{[r^*+m]}$
18 **return** u if u is of length n
19 **return** u

The algorithm chooses v to be a random string of length $n/2$. In order to obtain the original n-bit string x it suffices for us to reconstruct the $(3n/2)$-bit string z. The proof of correctness involves standard probabilistic arguments and is deferred to the full version.

For convenience of notation, we rename z as x and rename n to be the length of this string z, so we still have $x = (x_0, \ldots, x_{n-1})$. Now x is an n-bit string that has the following property: any desert in x ends at least $n/4$ bits before the right end of x. With this preprocessing accomplished, we now describe Algorithm `Reconstruct` in Algorithm 1.

4.2 The high level idea of the `Reconstruct` algorithm

At a high level the algorithm works as follows. It starts (lines 1-3) by drawing

$$N = O(\log n)$$

independent traces $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^N$ from $\mathrm{Del}_\delta(x)$ and using them to run `BMA`. An important component of our analysis is the following new result about the performance of `BMA` (note that later we require, and will give, a more robust version of the theorem below; see Theorem 6):

▶ **Theorem 4.** *Let $\delta = n^{-(1/3+\varepsilon)}$ for some fixed constant $\varepsilon > 0$. Given N traces drawn independently from $\mathrm{Del}_\delta(x)$ for some unknown string $x \in \{0,1\}^n$, `BMA` runs in $\tilde{O}(n)$ time and returns a string w of length n with the following performance guarantees:*

1. *If x has no desert, then $w = x$ with probability at least $1 - 1/n^2$;*
2. *If x has at least one desert, then w and x share the same $(r+m+1)$-bit prefix with probability at least $1 - 1/n^2$, where r is the first location that is deep in a desert of x.*

Let w be the string `BMA` returns. By Theorem 4, we have the following two cases:

1. If w has no desert, then also x has no desert and the algorithm can just return w (line 4);
2. If w has at least one desert, then writing r to denote the first location that is deep in a desert in w, it is safe to assume that $w_{[0:r+m]} = x_{[0:r+m]}$ and r is also the first location that is deep in a desert in x (line 6).

Suppose that we are in the second case with $w_{[0:r+m]} = x_{[0:r+m]}$. Then $w_{[r-m:r+m]}$ is an s-desert for some string $s \in \{0,1\}^k$ of some length $k \le C$. We let s be the shortest such string and let k be its length (so if $w_{[r-m:r+m]}$ were, for example, a subword of the form $001001001001\cdots$ of length a multiple of 12, we would take $s = 001$ and $k = 3$).

Next (lines 8-9) we run `FindEnd` to figure out where this repetition of s ends in x. We use end to denote the end of the desert, where $\mathsf{end} \ge r + m$ is the smallest integer such that $x_{\mathsf{end}+1} \neq x_{\mathsf{end}-k+1}$. By the preprocessing step we may assume that end exists and satisfies $\mathsf{end} \le 3n/4$. (We note that `FindEnd` has access to $\mathrm{Del}_\delta(x)$ to draw fresh traces by itself; we send N fresh traces $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N$ to `FindEnd` so that it can help align them to the end of the desert, which are used to run `BMA` later.) The performance guarantee for `FindEnd` is given below:

▶ **Theorem 5.** *Let $\delta = n^{-(1/3+\varepsilon)}$ for some fixed constant $\varepsilon > 0$. There is an algorithm* `FindEnd` *with the following input and output:*

- **Input:** *(i) a location $r \in [0 : 3n/4]$, (ii) a string $u \in \{0,1\}^{r+m+1}$, (iii) a multiset of strings $\{y^1, \ldots, y^N\}$ from $\{0,1\}^{\le n}$ where $N = O(\log n)$, and (iv) sample access to $\mathrm{Del}_\delta(x)$ for some unknown string $x \in \{0,1\}^n$.*
- **Output:** *An integer b, and an integer ℓ_i for each $i \in [N]$.*

The algorithm `FindEnd` *draws $\tilde{O}(n^{2/3-\varepsilon})$ many independent traces from $\mathrm{Del}_\delta(x)$, runs in $O(n^{5/3})$ time and has the following performance guarantee. Suppose that r is the first location that is deep in some desert of x; $u = x_{[0:r+m]}$; the unknown end of the desert to which x_r belongs is at most $3n/4$; and $y^1 = \boldsymbol{y}^1, \ldots, y^N = \boldsymbol{y}^N$ are independent traces drawn from $\mathrm{Del}_\delta(x)$. Then the integers b and ℓ_i that* `FindEnd` *outputs satisfy the following properties with probability at least $1 - 1/n^2$: $b = \mathsf{end}$, and $\ell_i = \mathrm{last}(\boldsymbol{y}^i)$ for at least 0.9 fraction of $i \in [N]$. Here $\mathrm{last}(y)$ for a trace y denotes the location ℓ in y such that y_ℓ corresponds to the last bit of $x_{[0:\mathsf{end}]}$ that survives in y (and we set $\mathrm{last}(y) = -1$ by default if all of $x_{[0:\mathsf{end}]}$ gets deleted in y).*

Line 9 runs `FindEnd` with fresh independent traces $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N$ drawn from $\mathrm{Del}_\delta(x)$. By Theorem 5, with high probability `FindEnd` returns the correct location $b = \mathsf{end}$, from which we can then recover $x_{[0:b]}$ as the unique extension of $w_{[0:r+m]}$ in which the pattern s keeps repeating until (and including) location b. Moreover, we have from Theorem 5 that, for at least a 9/10-fraction of all $i \in [N]$, the suffix $\boldsymbol{z}^i$ of $\boldsymbol{y}^i$ starting from $\boldsymbol{y}^i_{\ell_i+1}$ is a trace drawn from $\mathrm{Del}_\delta(x_{[b+1:n-1]})$. We further note that our preprocessing ensures $b \le 3n/4$ and thus, the algorithm does not halt on line 11.

To continue, we would like to run `BMA` again on $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^N$ (the suffixes of $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N$) to recover $x_{[b+1:n-1]}$ (or a prefix of $x_{[b+1:n-1]}$ if it contains a desert). However, observe that now we need `BMA` to be robust against some noise in its input traces because by Theorem 5, up to

$1/10$ of $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^N$ might have been obtained from an incorrect alignment of $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N$. Thus we require the following more robust performance guarantee from BMA, given by Theorem 6 below. (To state this we need a quick definition: we say two multisets of strings of the same size are η-*close* if one can be obtained from the other by substituting no more than η-fraction of its strings. One should also consider x' in the statement below as $x_{[b+1:n-1]}$ and n' as $n-b-1$.)

▶ **Theorem 6.** *Let $\delta = n^{-(1/3+\varepsilon)}$ for some fixed constant $\varepsilon > 0$. Suppose $\tilde{\boldsymbol{z}}^1, \ldots, \tilde{\boldsymbol{z}}^N$ are N independent traces drawn from $\mathrm{Del}_\delta(x')$ for some unknown string $x' \in \{0,1\}^{n'}$ with $n' \leq n$. The following holds with probability at least $1 - 1/n^2$ over the randomness of $\tilde{\boldsymbol{z}}^1, \ldots, \tilde{\boldsymbol{z}}^N \sim \mathrm{Del}_\delta(x')$:*

1. *If x' has no desert, then* BMA *running on n' and any multiset $\{z^1, \ldots, z^N\}$ that is $(1/10)$-close to $\{\tilde{\boldsymbol{z}}^1, \ldots, \tilde{\boldsymbol{z}}^N\}$ returns $w = x'$;*
2. *If x' has at least one desert, then* BMA *running on n' and any multiset $\{z^1, \ldots, z^N\}$ that is $(1/10)$-close to $\{\tilde{\boldsymbol{z}}^1, \ldots, \tilde{\boldsymbol{z}}^N\}$ returns a string w that shares the same $(r'+m+1)$-bit prefix with x', where r' is the first location that is deep in a desert in x'.*

Given Theorem 6, we can indeed successfully run BMA on $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^N$ and with high probability, it correctly recovers a prefix of $x_{[b+1:n-1]}$ up to the first point deep in the next desert (if any exists), in which case the algorithm repeats (if there is no next desert, then with high probability BMA will correctly recover the rest of x).

4.3 Correctness of Reconstruct

The case when x has no desert is handled by Theorem 6. Assuming that x has at least one desert, it follows from Theorem 6 that r, u together satisfy the following property with probability at least $1 - 1/n^2$ at the beginning of the main loop (lines 7-18): r is the first location that is deep in a desert in x and $u = x_{[0:r+m]}$. This gives the base case for the following invariant that the algorithm maintains with high probability:

> **Invariant:** *At the beginning of each loop, r is the first location deep in some desert in x and $u = x_{[0:r+m]}$.*

Assume that the invariant is met at the beginning of the current loop. Let end denote the end of the current desert (i.e., the smallest value $\mathsf{end} \geq r+m$ such that $x_{\mathsf{end}+1} \neq x_{\mathsf{end}-k+1}$; we observe that $\mathsf{end} \leq 3n/4$ always exists by the guarantee of the preprocessing step). Let $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N$ be fresh traces drawn at the beginning of this loop. For each $i \in [N]$, we write $\tilde{\boldsymbol{z}}^i$ to denote the suffix of $\boldsymbol{y}^i$ starting at $\mathrm{last}(\boldsymbol{y}^i)+1$. Given that $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N \sim \mathrm{Del}_\delta(x)$, $\tilde{\boldsymbol{z}}^1, \ldots, \tilde{\boldsymbol{z}}^N$ are indeed independent traces drawn from $\mathrm{Del}_\delta(x')$, where $x' = x_{[\mathsf{end}+1:n-1]}$. Then we note that, for the algorithm to deviate from the invariant in the current round, one of the following two events must hold for $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N$:

1. FindEnd$(r, u, \{\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N\})$ fails Theorem 5; or
2. $\{\tilde{\boldsymbol{z}}^1, \ldots, \tilde{\boldsymbol{z}}^N\}$ fails Theorem 6, i.e., there is a multiset $\{z^1, \ldots, z^N\}$ that is $(1/10)$-close to $\{\tilde{\boldsymbol{z}}^1, \ldots, \tilde{\boldsymbol{z}}^N\}$ but BMA$(n - \mathsf{end} - 1, \{z^1, \ldots, z^N\})$ violates the condition in Theorem 6.

This is because whenever FindEnd succeeds, the strings $\{\boldsymbol{z}^1, \ldots, \boldsymbol{z}^N\}$ on which we run BMA on line 13 must be $(1/10)$-close to $\{\tilde{\boldsymbol{z}}^1, \ldots, \tilde{\boldsymbol{z}}^N\}$. Theorem 5 ensures that item 1 happens with probability at most $1/n^2$; Theorem 6 ensures that item 2 happens with probability at most $1/n^2$, given that $\tilde{\boldsymbol{z}}^1, \ldots, \tilde{\boldsymbol{z}}^N$ are independent traces from $\mathrm{Del}_\delta(x')$ as required in the assumption of Theorem 6.

Algorithm 2 Algorithm BMA.

Input: A length n' and a multiset $\{z^1, \ldots, z^N\}$ of strings, each of length at most n'
Output: A string $w = (w_0, \ldots, w_{n'-1}) \in \{0,1\}^{n'}$

For each $i \in [N]$ pad each z^i to be a string u^i of length n' by adding 0's to the end
Set $t = 0$ and $\text{current}_i(t) = 0$ for each $i \in [N]$
while $t \leq n' - 1$ **do**
    Set $w_t \in \{0,1\}$ to be the majority of the N bits $u^1_{\text{current}_1(t)}, \ldots, u^N_{\text{current}_N(t)}$
    For each $i \in [N]$, set $\text{current}_i(t+1)$ to $\text{current}_i(t) + 1$ if $u^i_{\text{current}_i(t)} = w_t$; otherwise set $\text{current}_i(t+1)$ to $\text{current}_i(t)$
    Increment t.
return w.

By a union bound, the invariant holds with high probability in every round given that we only repeat for n/m rounds. Finally, observe that we only need to repeat for n/m rounds to reconstruct the entire n-bit string x, since in each round the pointer r increases by at least $2m$.

This concludes the proof of correctness of Reconstruct and the proof of Theorem 1, modulo the proofs of Theorem 6 and Theorem 5. In the rest of the paper we prove those two theorems.

5 Improved analysis of the Bitwise Majority Algorithm: Proof of Theorem 6

The bitwise majority algorithm was first described and analyzed in [2]. The analysis given in [2] established that BMA successfully reconstructs any unknown source string $x \in \{0,1\}^n$ that does not contain any "long runs" (i.e., subwords of the form $0^{n^{1/2+\varepsilon}}$ or $1^{n^{1/2+\varepsilon}}$) provided that the deletion rate δ is at most $n^{-(1/2+\varepsilon)}$. We describe the BMA algorithm in Algorithm 2. As the main result of this section we establish an improved performance guarantee for BMA. Our discussion and notation below reflects the fact that we will in general be running BMA "in the middle" of a string x for which we have already reconstructed a $(b+1)$-bit prefix of x (this is why Theorem 6 is stated in terms of a source string x' of length $n' \leq n$, which should be thought of as a suffix of x). Our goal is to prove Theorem 6.

We break the proof of Theorem 6 into two steps (Lemma 8 and Lemma 14 below). For ease of exposition, in the rest of this section if x' has at least one desert then as stated in item (2) of the theorem, we let r' be the first location that is deep in a desert in x'. If x' has no desert, then we let $r' = n' - m - 1$. Note that with this definition of r', it is guaranteed that there is no desert in $x'_{[0:r'+m-1]}$ and the goal of BMA is to return a string that shares the same $(r'+m+1)$-prefix with x.

Let $R = 9N/10$. We first prove in Lemma 8 that if a multiset of R traces $Z = \{z^1, \ldots, z^R\}$ of x' satisfies a certain sufficient "goodness" condition (see Definition 7 for details), then $\text{BMA}(n', Z)$ not only returns a string $w = (w_0, \ldots, w_{n'-1}) \in \{0,1\}^n$ that satisfies $w_{[0:r'+m]} = x'_{[0:r'+m]}$ as desired but moreover, the bitwise majority during each of the first $r'+m+1$ rounds of BMA is "robust" in the following sense: for each one of those rounds, at least $9R/10 = 81N/100$ of the R strings z^i's agree with each other. This immediately implies that when Z satisfies this condition, adding any multiset of $N/10$ strings to Z and running BMA on the resulting multiset of size N cannot affect the output of BMA during the first $r'+m+1$ rounds, so its output w still satisfies $w_{[0:r'+m]} = x'_{[0:r'+m]}$. The next lemma,

Lemma 14, shows that if $\tilde{\mathbf{Z}} = \{\tilde{\boldsymbol{z}}^1, \ldots, \tilde{\boldsymbol{z}}^N\}$ is a multiset of N traces drawn independently from $\mathrm{Del}_\delta(x')$ (as in the assumption part of Theorem 6), then with high probability *every* R-element subset of $\tilde{\mathbf{Z}}$ satisfies the sufficient condition (Definition 7) for BMA to succeed robustly. Theorem 6 follows easily by combining Lemma 8 and Lemma 14. Due to lack of space, we defer most proofs in this section to the full version.

5.1 Notation for traces

We start with some useful notation for analyzing traces of x'. When a trace $\boldsymbol{y}$ is drawn from $\mathrm{Del}_\delta(x')$ we write $\mathbf{D}$ to denote the set of *locations deleted* when x' goes through the deletion channel, i.e., $\mathbf{D}$ is obtained by including each element in $[0 : n' - 1]$ independently with probability δ, and $\boldsymbol{y}$ is set to be $x'_{[0:n'-1]\setminus\mathbf{D}}$. In the analysis of BMA when it is given as input R traces $Z = \{z^1, \ldots, z^R\}$, our analysis will sometimes refer to the set $D_i \subseteq [0 : n' - 1]$ of locations that was deleted when generating z^i.

Note that in the execution of BMA we pad each trace z^i to a string u^i of length n' by adding 0's to its end . In the rest of the section it will be convenient for us to view x' as a string of infinite length by adding infinitely many 0's to its end. We can then view each u^i as generated by first deleting the bits in $D_i \subseteq [0 : n' - 1]$ from x' and taking the n'-bit prefix of what remains. This motivates the definition of the following map $f_i : [0 : n' - 1] \to \mathbb{N}$ for each $i \in [R]$: For each $j \in [0 : n' - 1]$, $f_i(j)$ is set to be the unique integer k such that $k \notin D_i$ and $k - |D_i \cap [k - 1]| = j$. In words, $f_i(j)$ is simply the original location in x' of the j-th bit in the padded version u^i of z^i.

We specify some parameters that will be used in the rest of Section 5. Let $C = \lceil 100/\varepsilon \rceil$ (so C should be thought of as a large absolute constant) and $M = 2m + 1$ with $m = n^{1/3}$, and recall that by definition M is the shortest possible length of a desert.

5.2 BMA is robust on good sets of traces

The main result of this subsection is Lemma 8, which establishes that BMA is robustly correct in its operation on traces that satisfy a particular "goodness" condition given in Definition 7 below.

Let $Z = \{z^1, \ldots, z^R\}$ be a multiset of traces of x'. As described above we write $u^i \in \{0,1\}^{n'}$ to denote the 0-padded version of z^i, $D_i \subseteq [0 : n' - 1]$ to denote the set of locations that were deleted from x' to form z^i, and f_i to denote the map defined as above for each $i \in [R]$. We introduce the following condition for Z and then prove Lemma 8:

▶ **Definition 7.** *We say $Z = \{z^1, \ldots, z^R\}$ is* good *if the following two conditions hold:*

- **(i)** *For every $i \in [R]$ and every interval $[\mathsf{left} : \mathsf{right}] \subset [0 : n' - 1]$ of length $\mathsf{right} - \mathsf{left} + 1 = L_1 := 2C^2M$, we have $|D_i \cap [\mathsf{left} : \mathsf{right}]| \le C$.*
- **(ii)** *For every interval $[\mathsf{left} : \mathsf{right}] \subset [0 : n' - 1]$ of length $\mathsf{right} - \mathsf{left} + 1 = L_2 := M + C + 1$, the number of elements $i \in [R]$ such that $D_i \cap [\mathsf{left} : \mathsf{right}] \neq \emptyset$ is at most R/C^3.*

Intuitively, (i) says that no interval of moderate length (note that this length $2C^2M$ is polynomially less than $1/\delta$) has "too many" deletions in it in any trace, whereas (ii) says that for every interval of moderate length (again polynomially less than $1/\delta$), most of the R traces have no bit deleted within that interval.

Now we are ready to state Lemma 8:

▶ **Lemma 8.** *Let $Z = \{z^1, \ldots, z^R\}$ be a good multiset of R traces of x'. Then the string $w \in \{0,1\}^{n'}$ that $\mathtt{BMA}(n', Z)$ outputs satisfies $w_{[0:r'+m]} = x'_{[0:r'+m]}$. Moreover, during each of the first $r' + m + 1$ rounds of the execution of BMA, at least $9R/10$ of the R bits in the majority vote taken in Step 4 of BMA agree with each other.*

We start the proof of Lemma 8 by defining a map $\mathsf{distance}_i(t)$ for each z^i in Z. Recall that $\mathrm{current}_i(t)$ is the current location of the pointer into the padded trace u^i at the beginning of round t in BMA.[6] We let $\mathsf{position}_i(t) = f_i(\mathrm{current}_i(t))$, i.e. the original position in x' of the $\mathrm{current}_i(t)$-th bit of u^i. Then $\mathsf{distance}_i(t)$ is defined as $\mathsf{distance}_i(t) = \mathsf{position}_i(t) - t$, the distance between t and $\mathsf{position}_i(t)$. In Corollary 10 we will show that $\mathsf{distance}_i(t)$ is always nonnegative, and so it actually measures how many bits $\mathsf{position}_i(t)$ is ahead at round t. It may be helpful to visualize t and $\mathsf{position}_i(t)$ of a trace by writing down the source string x' with the deleted bits struck through, and having two arrows pointing to x'_t and $x'_{\mathsf{position}_i(t)}$; at the beginning of round t, the BMA algorithm tries to determine x'_t by looking at $x'_{\mathsf{position}_i(t)}$. Intuitively, having $\mathsf{distance}_i(t) = 0$ means that the i-th trace was aligned properly at round t; at the highest level, we establish Lemma 8 by showing that at least $9R/10$ of the R traces have $\mathsf{distance}_i(t) = 0$.

We state the following claim about how $\mathrm{current}_i(t), \mathsf{position}_i(t)$ and $\mathsf{distance}_i(t)$ compare to their values at the beginning of round $t-1$, assuming that the prefix $w_{[0:t-1]}$ of the output thus far matches $x'_{[0:t-1]}$.

▷ Claim 9. Let t be a positive integer such that $w_{[0:t-1]} = x'_{[0:t-1]}$. For each $i \in [R]$, we have

1. If $x'_{\mathsf{position}_i(t-1)} \neq x'_{t-1}$, then $\mathrm{current}_i(t) = \mathrm{current}_i(t-1)$, $\mathsf{position}_i(t) = \mathsf{position}_i(t-1)$ and $\mathsf{distance}_i(t) = \mathsf{distance}_i(t-1) - 1$.
2. If $x'_{\mathsf{position}_i(t-1)} = x'_{t-1}$, then $\mathrm{current}_i(t) = \mathrm{current}_i(t-1) + 1$, $\mathsf{position}_i(t) = \mathsf{position}_i(t-1) + \ell + 1$ and $\mathsf{distance}_i(t) = \mathsf{distance}_i(t-1) + \ell$, where ℓ is the nonnegative integer such that $\mathsf{position}_i(t-1)+1, \ldots, \mathsf{position}_i(t-1)+\ell \in D_i$ and $\mathsf{position}_i(t-1)+\ell+1 \notin D_i$ (or equivalently, $\ell = f_i(\mathrm{current}_i(t)) - f_i(\mathrm{current}_i(t-1)) - 1$).

We have the following useful corollary of Claim 9, which tells us that if $w_{[0:t-1]} = x'_{[0:t-1]}$ then each $\mathsf{distance}_i(t) \geq 0$ (in other words, no trace can have "gotten behind" where it should be):

▶ **Corollary 10.** *Let t be a positive integer such that $w_{[0:t-1]} = x'_{[0:t-1]}$. Then $\mathsf{distance}_i(t) \geq 0$ for all $i \in [R]$.*

We prove three preliminary lemmas before proving Lemma 8. Recall that $M = 2m+1$ is the shortest possible length of a desert. Assuming $w_{[0:t-1]} = x'_{[0:t-1]}$ for some $t > M$, the first lemma shows that if $\mathsf{distance}_i(t-M) = 0$ and no location of x' is deleted between $t-M+1$ and t, then $\mathsf{distance}_i(t)$ must stay at 0. (Note that this lemma holds for general M but we state it using $M = 2m+1$ for convenience since this is how it will be used later.) Intuitively, this says that if a length-M subword of x' experiences no deletions, then a trace that is correctly aligned at the start of the subword will stay correctly aligned throughout the subword and at the end of the subword.

▶ **Lemma 11.** *Suppose that $w_{[0:t-1]} = x'_{[0:t-1]}$ for some $t > M$. Suppose that $i \in [R]$ is such that $\mathsf{distance}_i(t-M) = 0$ and*

$$D_i \cap \big[\mathsf{position}_i(t-M)+1 : \mathsf{position}_i(t-M)+M\big] = D_i \cap \big[t-M : t\big] = \emptyset.$$

Then we have $\mathsf{distance}_i(t) = 0$.

[6] Note that whereas $\mathsf{position}_i(\cdot)$ and $\mathsf{distance}_i(\cdot)$ refer to quantities defined in terms of the source string x, $\mathrm{current}_i(\cdot)$ refers to a location in a trace string and not the source string.

In the second lemma, we assume t is such that $M < t \leq r' + m + 1$ by the choice of r. We further assume that $w_{[0:t-1]} = x'_{[0:t-1]}$ and $0 < \mathsf{distance}_i(t - M) \leq C$ for some $i \in [R]$. We show that under these assumptions, if the subword of length M in x' starting at $\mathsf{position}_i(t - M) + 1$ has no deletion, then $\mathsf{distance}_i(t) < \mathsf{distance}_i(t - M)$. Intuitively, this says that prior to a desert, if the length-M subword of x' experiences no deletions and the alignment of a trace is only modestly ahead of where it should be at the start of the subword, then the alignment will improve by the end of the subword.

▶ **Lemma 12.** *Let $M < t \leq r' + m + 1$ with $w_{[0:t-1]} = x'_{[0:t-1]}$. If $0 < \mathsf{distance}_i(t - M) \leq C$ for some $i \in [R]$ and $D_i \cap \left[\mathsf{position}_i(t - M) + 1 : \mathsf{position}_i(t - M) + M\right] = \emptyset$, then we have $\mathsf{distance}_i(t) < \mathsf{distance}_i(t - M)$.*

Finally we use the two previous lemmas to show that if $t \leq r'+m+1$ and $w_{[0:t-1]} = x'_{[0:t-1]}$, then $\mathsf{distance}_i(t)$ must lie between 0 and C. Intuitively, this says that prior to a desert, the alignment of a trace will be at worst modestly ahead of where it should be.

▶ **Lemma 13.** *Let $t \leq r' + m + 1$ and suppose that $w_{[0:t-1]} = x'_{[0:t-1]}$. Then $\mathsf{distance}_i(t) \leq C$ for all $i \in [R]$.*

Proof of Lemma 8. We prove by induction that for every positive integer $t \leq r' + m + 1$:

$$w_{[0:t-1]} = x'_{[0:t-1]} \quad \text{and} \quad \sum_{i \in [R]} \mathsf{distance}_i(t) \leq \frac{2R}{C}. \tag{1}$$

It follows that every $t \leq r'+m+1$ satisfies $|\{i \in [R] : \mathsf{distance}_i(t) = 0\}| \geq R - 2R/C \geq 9R/10$ using $C \geq 20$. The details are deferred to the full version. ◀

5.3 Traces are good with high probability

To conclude the proof of Theorem 6 it remains to prove Lemma 14, which states that with high probability a random multiset of $O(\log n)$ traces is such that every subset of 9/10 of the traces is good (recall Definition 7).

▶ **Lemma 14.** *Let $\tilde{\mathbf{Z}} = \{\tilde{\boldsymbol{z}}^1, \ldots, \tilde{\boldsymbol{z}}^N\}$ be a multiset of $N = O(\log n)$ traces drawn independently from $\mathrm{Del}_\delta(x')$. Then with probability at least $1 - 1/n^2$, every R-subset of $\tilde{\mathbf{Z}}$ is good, where $R = 9N/10$.*

6 Finding the end of a desert: Proof of Theorem 5

In this section, we describe the algorithm `FindEnd`, which is used to determine the end of a desert in x using traces from $\mathrm{Del}_\delta(x)$, and to align given traces with the end of the desert. (These aligned traces will then be used by `BMA` in the main algorithm.)

Let's recall the setting. Let $x \in \{0,1\}^n$ be the unknown string. `FindEnd` is given the first location r that is deep in some s-desert subword of x, for some string $s \in \{0,1\}^k$ with $k \leq C$. It is also given the prefix $u = x_{[0:r+m]}$ of x. We will refer to the s-desert that contains r as the *current* desert. (Note that s can be easily derived from u.) The goal of `FindEnd` is to figure out the ending location of the current desert which we denote by end:

end *is the smallest integer at least $r + m$ such that $x_{\mathsf{end}+1} \neq x_{\mathsf{end}-k+1}$.*

(Note that thanks to the preprocessing step `Preprocess`, we know that end exists and satisfies $r + m \leq \mathsf{end} \leq 3n/4$.)

In addition to computing end, `FindEnd` is also given a multiset of $N = O(\log n)$ traces $y^1, \ldots, y^N$ and needs to return a location ℓ_i for each y^i such that most of them are correctly aligned to the end of the desert. Formally, we write $\mathrm{last}(y)$ for a trace y to denote the location ℓ in y such that y_ℓ corresponds to the last bit of $x_{[0:\mathsf{end}]}$ that survives in y; we set $\mathrm{last}(y) = -1$ by default if all of $x_{[0:\mathsf{end}]}$ gets deleted. The second goal of `FindEnd` is to output $\ell_i = \mathrm{last}(y^i)$ for almost all y^i when they are drawn independently from $\mathrm{Del}_\delta(x)$.

We present the algorithm `FindEnd` in Algorithm 3, where

$$\sigma := \lceil \sqrt{\delta n} \cdot \log n \rceil.$$

(Intuitively, σ provides a high-probability upper bound on how far a bit of x can deviate from its expected position in a trace $\boldsymbol{y} \sim \mathrm{Del}_\delta(x)$.) `FindEnd` consists of the following two main procedures:

1. We will refer to the 8σ-bit string
$$\mathsf{tail} := x_{\mathsf{end}-k+2}\ x_{\mathsf{end}-k+3}\ \cdots\ x_{\mathsf{end}+8\sigma-k+1}$$
around the end x_{end} of the current desert as its *tail* string and denote it by $\mathsf{tail} \in \{0,1\}^{8\sigma}$. (Note that $\mathsf{end} + 8\sigma - k + 1 < n$ given that $\mathsf{end} \le 3n/4$.) The first procedure, `Coarse-Estimate`, will provide with high probability a *coarse estimate* $\widehat{\beta}$ (see Lemma 15) of the expected location $(1-\delta)\mathsf{end}$ of the right end of the current desert in a trace of x. This procedure is described in Section 6.1.
2. With $\widehat{\beta}$ and $\mathsf{tail} \in \{0,1\}^{8\sigma}$ in hand, the second procedure `Align` can help align a given trace with the right end of the current desert. Informally, running on a trace $\boldsymbol{y} \sim \mathrm{Del}_\delta(x)$, `Align` returns a position ℓ such that with high probability over the randomness of $\boldsymbol{y} \sim \mathrm{Del}_\delta(x)$, it holds that $\ell = \mathrm{last}(\boldsymbol{y})$. The performance guarantee of `Align` is given in Lemma 16. It may sometimes (with a small probability) return nil, meaning that it fails to align the given trace. This procedure is described in Section 6.2.

The algorithm `FindEnd` starts by running `Coarse-Estimate` to obtain a coarse estimate $\widehat{\beta}$ of $(1-\delta)\mathsf{end}$ and the tail string (line 1). It then (line 2) runs `Align` on the given N traces y^i to obtain ℓ_i for each $i \in [N]$. The second property of `FindEnd` in Theorem 5 about ℓ_i's follows directly from the performance guarantee of `Align`. To obtain a sharp estimate of end, `FindEnd` draws another set of $\tilde{O}(n^{2/3-\varepsilon})$ traces $\boldsymbol{z}^i$ (line 3). It runs `Align` on each of them and uses the average of its outputs (discarding traces for which `Align` returns nil) to estimate $(1-\delta)\mathsf{end}$ (lines 4-6). (It is clear that this average would be accurate to within $\pm o(1)$ if `Align` always successfully aligned its input trace with the right end of the current desert; the actual performance guarantee of `Align` is weaker than this, but a careful analysis enables us to show that it is good enough for our purposes.)

6.1 The `Coarse-Estimate` procedure

Recall that $\sigma = \lceil \sqrt{\delta n} \cdot \log n \rceil$. Given r, u as specified earlier and sample access to $\mathrm{Del}_\delta(x)$, the goal of `Coarse-Estimate` is to obtain an integer $\widehat{\beta}$ such that $|\widehat{\beta} - (1-\delta)\mathsf{end}| \le 2\sigma$. We will refer to such an estimate as a *coarse estimate* of $(1-\delta)\mathsf{end}$. In addition, `Coarse-Estimate` returns a string t that with high probability is exactly the tail string $\mathsf{tail} \in \{0,1\}^{8\sigma}$. This is done by drawing only $O(1/\varepsilon)$ many traces.

▶ **Lemma 15.** *Let $\delta = n^{-(1/3+\varepsilon)}$ with a fixed constant $\varepsilon > 0$. There is an algorithm* `Coarse-Estimate` *which takes the same two inputs r and u as in* `FindEnd` *and sample access to $\mathrm{Del}_\delta(x)$ for some unknown string $x \in \{0,1\}^n$, and returns an integer $\widehat{\beta} \in [0 : n-1]$ and a string $t \in \{0,1\}^{8\sigma}$. It draws $O(1/\varepsilon)$ traces from $\mathrm{Del}_\delta(x)$, runs in time $O(n)$ and has the following performance guarantee. Suppose that r and u satisfy the same conditions as in Theorem 5 with respect to x. Then with probability at least $1 - 1/n^3$, we have that $t = \mathsf{tail}$ and $\widehat{\beta}$ satisfies $|\widehat{\beta} - (1-\delta)\mathsf{end}| \le 2\sigma$.*

Algorithm 3 Algorithm `FindEnd`.

Input: $r \in [0:3n/4]$, $u \in \{0,1\}^{r+m+1}$, a multiset $\{y^1, \dots, y^N\}$ of N strings from $\{0,1\}^{\le n}$ where $N = O(\log n)$, and sample access to $\mathrm{Del}_\delta(x)$ for some string $x \in \{0,1\}^n$.

Output: An integer $b \ge r+m$ and an integer ℓ_i for each $i \in [N]$.

1 Run `Coarse-Estimate`(r,u), which returns an integer $\widehat{\beta}$ and a string $t \in \{0,1\}^{8\sigma}$.

2 For each $i \in [N]$, run `Align`$(\widehat{\beta}, t, y^i)$. If `Align` returns nil, set $\ell_i = -1$; otherwise let ℓ_i be the integer `Align` returns.

3 Draw $\gamma = O(n^{2/3-\varepsilon}\log^3 n)$ traces $\boldsymbol{z}^1, \dots, \boldsymbol{z}^\gamma$ from $\mathrm{Del}_\delta(x)$.

4 For each $i \in [\gamma]$, run `Align`$(\widehat{\beta}, t, \boldsymbol{z}^i)$ and let h_i be its output.

5 Let β be the average of h_i's that are not nil, and let b be the integer nearest to $\beta/(1-\delta)$.

6 Return b, and ℓ_i for each $i \in [N]$.

Proof. We start with the coarse estimate $\widehat{\beta}$. Let $\widehat{r} = \lceil (1-\delta)r \rceil$ and consider the following collection of overlapping intervals of positions in a trace of x:

$$\mathcal{I} := \Big\{ \big[\widehat{r} + j\sigma : \widehat{r} + (j+4)\sigma\big] : j \in \mathbb{Z}_{\ge 0} \Big\}.$$

Note that each interval I contains $4\sigma+1$ positions. `Coarse-Estimate` draws $\alpha = O(1/\varepsilon)$ traces $\boldsymbol{y}^1, \dots, \boldsymbol{y}^\alpha$ from $\mathrm{Del}_\delta(x)$ and finds the leftmost interval $I^* \in \mathcal{I}$ such that at least half of $\boldsymbol{y}^i$'s satisfy the following property: $\boldsymbol{y}^i_{I^*}$ contains a k-bit subword that is not a cyclic shift of s. The algorithm then sets $\widehat{\beta}$ to be the right endpoint of I^*.

Finally, `Coarse-Estimate` recovers the tail string as follows. Let J' be the interval $[\widehat{\beta} - 3\sigma : \widehat{\beta} + 3\sigma]$. It draws another sequence of $\alpha = O(1/\varepsilon)$ fresh traces $\boldsymbol{y}^1, \dots, \boldsymbol{y}^\alpha$ from $\mathrm{Del}_\delta(x)$. For each $\boldsymbol{y}^i$ it looks for the leftmost non-cyclic shift of s in $\boldsymbol{y}^i_{J'}$. When such a non-cyclic shift exists, say $\boldsymbol{y}^i_\tau \cdots \boldsymbol{y}^i_{\tau+k-1}$, $\boldsymbol{y}^i$ *votes* for the 8σ-bit string $\boldsymbol{y}^i_\tau \cdots \boldsymbol{y}^i_{\tau+8\sigma-1}$ as its candidate for the tail string. It then returns the 8σ-bit string with the most votes.

Clearly, the running time of `Coarse-Estimate` is $O(n)$ as the procedure consists of a linear scan over $O(1/\varepsilon)$ traces. The proof of correctness is deferred to the full version. ◀

6.2 The `Align` procedure

We start with the performance guarantee of `Align`:

▶ **Lemma 16.** *Let $\delta = n^{-(1/3+\varepsilon)}$ for some fixed constant $\varepsilon > 0$. There is an algorithm* `Align` *running in time $O(n)$ with the following input and output:*

- **Input:** *a number $\widehat{\beta} \in [0:n-1]$, and strings $t \in \{0,1\}^{8\sigma}, y \in \{0,1\}^{\le n}$.*
- **Output:** *an integer $\ell \in [0:n-1]$, or* nil.

It has the following performance guarantee. Suppose x, u, r and end *satisfy the hypothesis in Theorem 5, $\widehat{\beta}$ and t satisfy the conclusion of Lemma 15, and $y = \boldsymbol{y} \sim \mathrm{Del}_\delta(x)$ is a random trace. Then*

1. *Whenever* `Align` *returns an integer ℓ, we have $|\ell - (1-\delta)\mathsf{end}| \le O(\sigma)$.*
2. *With probability at least $1 - \tilde{O}(n^{-3\varepsilon/2})$,* `Align` *returns exactly* last$(\boldsymbol{y})$*; and*
3. *Conditioned on* `Align` *not returning* nil*, the expectation of what* `Align` *returns is $(1-\delta)\mathsf{end} \pm o(1)$.*

6.2.1 Setup for the proof of Lemma 16

For the special case when $k = |s| = 1$ (so the desert subword is of the form 0^a or 1^a for some $a \geq M = 2n^{1/3} + 1$), the description of `Align` is relatively simple.

- **Description of `Align` for $k = 1$:** Let $J := [\widehat{\beta} - 3\sigma : \widehat{\beta} + 3\sigma]$. `Align` outputs nil if the string y_J contains no occurrence of $\overline{b}$; if y_J does contain at least one occurrence of $\overline{b}$ then `Align` outputs the location in J of the first occurrence of $\overline{b}$.

The proof of correctness is deferred to the full version. Now consider the general case when $k \geq 2$. Let Cyc_s be the set of all k-bit strings that can be obtained as cyclic shifts of s. The key notion behind `Align` is the idea of the "*signature*." This is a subword of x of length at most 8σ that starts at the same location $x_{\mathsf{end}-k+2}$ as tail (so it is contained in tail; we remind the reader that the first k-bits of tail is a string not in Cyc_s). The signature ends at location d where d is the smallest integer $d \in [\mathsf{end} + k + 1 : \mathsf{end} + 8\sigma - k + 1]$ such that the k-bit subword that ends at d is not in Cyc_s; if no such d exists, the signature is taken to have length 8σ and is the same as tail. (Alternatively, the signature is the shortest prefix of tail that contains a k-bit subword not in Cyc_s that does not use the first k bits; and it is set to tail if every k-bit subword of tail after removing the first k bits lies in Cyc_s.)

We will write sig to denote the signature string. We observe that $2k \leq |\mathsf{sig}| \leq 8\sigma$, and that given the string tail it is algorithmically straightforward to obtain sig. Given sig, we say that a string z of length at most $15\sigma + 1$ is in the *right* form if it can be written as

$$z = w \circ \mathsf{sig} \tag{2}$$

where the leftmost k-bit subword in z that is not in Cyc_s is the first k bits of sig. The main motivation behind the definition of the signature is the following crucial lemma:

▶ **Lemma 17.** *Let $s \in \{0,1\}^k$ for some $2 \leq k \leq C$, and let z be a string of length at most $15\sigma + 1$ that is in the right form. For $\boldsymbol{y} \sim \mathrm{Del}_\delta(z)$, the probability that $|\boldsymbol{y}| < |z|$ (so at least one deletion occurs) and $\boldsymbol{y}$ is the prefix of a string in the right form is at most $O(\delta)$.*

The high-level idea is that a deletion is likely to create an additional disjoint k-bit subword in $\boldsymbol{y}$ that is not in Cyc_s, unless a deletion occurs in some $O(k)$ specific places in z or two deletions are $O(k)$ close to each other. This additional subword will help us argue that $\boldsymbol{y}$ does not have the right form. The detailed proof is deferred to the full version.

6.2.2 Proof sketch of Lemma 16 when $k \geq 2$

- **Description of `Align` for $k \geq 2$:** Given a coarse estimate $\widehat{\beta}$ (such that $|\widehat{\beta} - (1-\delta)\mathsf{end}| \leq 2\sigma$), $\mathsf{sig} \in \{0,1\}^{\leq 8\sigma}$, and a trace y, `Align` checks if the restriction of y to the interval $J := \left[\widehat{\beta} - 3\sigma : \widehat{\beta} + 12\sigma\right]$ has a prefix in the right form (see Equation (2)), i.e.,

  $$y_J = w \circ \mathsf{sig} \circ v \tag{3}$$

  so that the first k bits of sig is the leftmost k-bit subword of y_J not in Cyc_s. If y_J is not of this form `Align` returns nil. If y_J is of this form and sig ends at location $L \in [0 : 15\sigma]$ in y_J, `Align` returns the index of the $(k-1)$-th bit of sig (i.e., sig_{k-2} which intuitively should correspond to x_{end}) in y with probability

  $$p_L := (1-\delta)^{15\sigma - L}, \tag{4}$$

  and with the remaining probability returns nil. Note that

  $$(1-\delta)^{15\sigma-L} = 1 - O(\delta\sigma) = 1 - \tilde{O}(n^{3\varepsilon/2}), \quad \text{for all } L \in [0 : 15\sigma],$$

  so `Align` only returns nil with probability $o(1)$ when y_J is of the form Equation (3).

Discussion. The main subtlety in the definition of `Align` is the "discounting probability" given by Equation (4), which plays an important role in ensuring that the location returned by `Align` (conditioned on `Align` not returning nil) is sufficiently close in expectation to the correct location. The proof of correctness of `Align` is deferred to the full version.

6.3 Proof of Theorem 5

Proof of Theorem 5. The proof follows from the guarantees in Lemma 15 and Lemma 16, using standard concentration bounds. First, we have from Lemma 15 that the output $(\widehat{\beta}, t)$ of `Coarse-Estimate` satisfies $|\widehat{\beta} - (1-\delta)\mathsf{end}| \leq 2\sigma$ and $t = \mathsf{tail}$ with probability $1 - O(1)/n^3$. Assume that this holds for the rest of the proof.

By Lemma 16, with probability at least $1 - \tilde{O}(n^{-3\varepsilon/2})$ `Align` returns an integer ℓ_i (and not nil), and $\ell_i = \text{last}(\boldsymbol{y}^i)$, for each $i \in [N]$. Now, the Chernoff bound (additive form) implies that $\ell_i = \text{last}(\boldsymbol{y}^i)$ for at least 0.9 fraction of $i \in [N]$ with probability $1 - \exp(-\Omega(N)) \geq 1 - 1/n^3$, where we choose the hidden constant in $N = O(\log n)$ to be sufficiently large.

It remains to show that $b = \mathsf{end}$ with probability at least $1 - 1/n^3$. Recalling step 4 of `FindEnd`, let $G \subset [\gamma]$ be the set of indices i for which $h_i = \texttt{Align}(\widehat{\beta}, t, \boldsymbol{z}^i) \neq \text{nil}$. Using the same argument as above, we have that $|G| \geq 0.9\gamma$ with probability $1 - \exp(-\Omega(\gamma)) = 1 - \exp(-\tilde{\Omega}(n^{2/3-\varepsilon}))$. The guarantees in Lemma 16 imply that $|\,\mathbf{E}[h_i \,|\, h_i \neq \text{nil}] - (1-\delta)\mathsf{end}| \leq o(1)$ and that the random variable h_i (conditioned on its not being nil) always lies in an interval of width $O(\sigma)$ for all $i \in [G]$. Moreover, $\{h_i\}_{i \in G}$ are independent random variables.

Let $\beta = (1/|G|)\sum_{i \in G} h_i$ be the average of h_i over $i \in G$. By Hoeffding's inequality and our choice of $\gamma = O(n^{2/3-\varepsilon}\log^3 n) = O(\sigma^2 \log n)$ (with a sufficiently large hidden constant),

$$\mathbf{Pr}[|\beta - \mathbf{E}[h_i \,|\, h_i \neq \text{nil}]| \geq 0.1] \leq \exp\left(-\Omega\left(\frac{\gamma}{\sigma^2}\right)\right) \leq \exp(-\Omega(\log n)) \leq 1/n^3.$$

By triangle inequality, $|\beta - (1-\delta)\mathsf{end}| \leq 0.1$, and so $|\beta/(1-\delta) - \mathsf{end}| \leq 0.2$, with probability at least $1 - 1/n^3$. Hence, the integer b closest to $\beta/(1-\delta)$ is end, which implies `FindEnd` returns end with probability at least $1 - 1/n^2$ (by union bound over all the failure probabilities).

Finally, the runtime of `FindEnd` is dominated by the final procedure to compute b. Since each run of `Align` on a trace takes $O(n)$ and `Align` is run on $\gamma \leq n^{2/3}$ traces, `FindEnd` runs in time $O(n^{5/3})$. This concludes the proof of Theorem 5. ◀

References

1 Alexandr Andoni, Constantinos Daskalakis, Avinatan Hassidim, and Sebastien Roch. Global alignment of molecular sequences via ancestral state reconstruction. *Stochastic Processes and their Applications*, 122(12):3852–3874, 2012.

2 T. Batu, S. Kannan, S. Khanna, and A. McGregor. Reconstructing strings from random traces. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004*, pages 910–918, 2004.

3 Z. Chase. New lower bounds for trace reconstruction. *CoRR*, abs/1905.03031, 2019. `arXiv:1905.03031`.

4 Anindya De, Ryan O'Donnell, and Rocco A. Servedio. Optimal mean-based algorithms for trace reconstruction. In *Proceedings of the 49th ACM Symposium on Theory of Computing (STOC)*, pages 1047–1056, 2017.

5 N. Holden and R. Lyons. Lower bounds for trace reconstruction. *CoRR*, abs/1808.02336, 2018. `arXiv:1808.02336`.

6 Nina Holden, Robin Pemantle, and Yuval Peres. Subpolynomial trace reconstruction for random strings and arbitrary deletion probability. *CoRR*, abs/1801.04783, 2018. `arXiv:1801.04783`.

7 Nina Holden, Robin Pemantle, Yuval Peres, and Alex Zhai. Subpolynomial trace reconstruction for random strings and arbitrary deletion probability. *CoRR*, abs/1801.04783, 2020. `arXiv:1801.04783`.

8 T. Holenstein, M. Mitzenmacher, R. Panigrahy, and U. Wieder. Trace reconstruction with constant deletion probability and related results. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008*, pages 389–398, 2008.

9 V. V. Kalashnik. Reconstruction of a word from its fragments. *Computational Mathematics and Computer Science (Vychislitel'naya matematika i vychislitel'naya tekhnika), Kharkov*, 4:56–57, 1973.

10 Vladimir Levenshtein. Efficient reconstruction of sequences. *IEEE Transactions on Information Theory*, 47(1):2–22, 2001.

11 Vladimir Levenshtein. Efficient reconstruction of sequences from their subsequences or supersequences. *Journal of Combinatorial Theory Series A*, 93(2):310–332, 2001.

12 Andrew McGregor, Eric Price, and Sofya Vorotnikova. Trace reconstruction revisited. In *Proceedings of the 22nd Annual European Symposium on Algorithms*, pages 689–700, 2014.

13 Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels. *Probability Surveys*, 6:1–33, 2009.

14 Fedor Nazarov and Yuval Peres. Trace reconstruction with $\exp(o(n^{1/3}))$ samples. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1042–1046, 2017.

15 Lee Organick, Siena Dumas Ang, Yuan-Jyue Chen, Randolph Lopez, Sergey Yekhanin, Konstantin Makarychev, Miklos Z Racz, Govinda Kamath, Parikshit Gopalan, Bichlien Nguyen, et al. Random access in large-scale dna data storage. *Nature biotechnology*, 36(3):242, 2018.

16 S.M. Hossein Tabatabaei Yazdi, Ryan Gabrys, and Olgica Milenkovic. Portable and error-free DNA-based data storage. *Scientific Reports*, 7(1):5011, 2017.