Reconstructing Strings from Random Traces

Tuğkan Batu* Sampath Kannan† Sanjeev Khanna‡ Andrew McGregor§

Abstract

We are given a collection of m random subsequences (traces) of a string t of length n where each trace is obtained by deleting each bit in the string with probability q. Our goal is to exactly reconstruct the string t from these observed traces. We initiate here a study of deletion rates for which we can successfully reconstruct the original string using a small number of samples. We investigate a simple reconstruction algorithm called Bitwise Majority Alignment that uses majority voting (with suitable shifts) to determine each bit of the original string. We show that for random strings t, we can reconstruct the original string (w.h.p.) for $q = O(1/\log n)$ using only $O(\log n)$ samples. For arbitrary strings t, we show that a simple modification of Bitwise Majority Alignment reconstructs a string that has identical structure to the original string (w.h.p.) for $q = O(1/n^{1/2+\epsilon})$ using $O(1)$ samples. In this case, using $O(n \log n)$ samples, we can reconstruct the original string exactly. Our setting can be viewed as the study of an idealized biological evolutionary process where the only possible mutations are random deletions. Our goal is to understand at what mutation rates, a small number of observed samples can be correctly aligned to reconstruct the parent string.

In the process of establishing these results, we show that Bitwise Majority Alignment has an interesting self-correcting property whereby local distortions in the traces do not generate errors in the reconstruction and eventually get corrected.

1 Introduction

Let $t = t_1 t_2 \ldots t_n$ be a string over an alphabet Σ. Suppose we are given a collection of random subsequences (traces) of t where each random subsequence is obtained independently as follows: For each i, the symbol t_i is deleted independently with probability q. The surviving symbols are concatenated to produce the subsequence. How many observations do we need to reconstruct t with high probability?

A *deletion channel*, which can model the generation process above, is a communication channel that drops each symbol in a transmission independently with probability q. We use the terminology of a deletion channel and talk about t as the "transmitted string" and each random subsequence as the "received string." In the literature, various error correcting codes for the deletion channel are studied (cf., [3, 4, 5, 7]). Such codes allow one to reconstruct the transmitted string (from a single observation) when the transmitted string is actually a *codeword*. So, a decoding algorithm for such an error correcting code can be viewed as an algorithm to solve the problem stated above for a particular (and small) subset of all possible strings. We also would like to note that another class of error correcting codes called erasure codes [1] is resilient against dropped packets during the transmission. But in this model, one can take considerable advantage of the fact that we know the location of bits that were deleted. We would like to emphasize that in the problem that we study differs from these in two important respects: (i) we wish to successfully reconstruct any transmitted string (and not only codewords), and (ii) we have no information about the locations of deletions.

The central motivation for our problem comes from computational biology, in particular, the multiple sequence alignment problem. In a typical biological scenario, we observe related DNA or protein sequences from different organisms. These sequences are the product of a random process of evolution that inserts, deletes, and substitutes characters in the sequences. The multiple sequence alignment problem is commonly used to deduce conserved subpatterns from a set of sequences known to be biologically related [2]. In particular, one would like to deduce the common ancestor of these related sequences. In reality, each of the observed sequences is not produced independently by this evolution process. Sequences from organisms that are evolutionarily very closely related undergo common evolution

*Dept. of CIS, University of Pennsylvania, Philadelphia, PA. Email: batu@cis.upenn.edu. This work was supported by ARO DAAD 19-01-1047 and NSF CCR01-05337.

†Dept. of CIS, University of Pennsylvania, Philadelphia, PA. Email: kannan@cis.upenn.edu. This work was supported by NSF CCR98-20885 and NSF CCR01-05337.

‡Dept. of CIS, University of Pennsylvania, Philadelphia, PA. Email: sanjeev@cis.upenn.edu. Supported in part by an Alfred P. Sloan Research Fellowship and by an NSF Career Award CCR-0093117.

§Dept. of CIS, University of Pennsylvania, Philadelphia, PA. Email: andrewm@cis.upenn.edu.

(identical changes) for the most part and only diverge and undergo independent evolution for a small period of time.

The multiple sequence alignment problem is one of the most important problems in computational biology and is known to be NP-hard. An *alignment* of k strings is obtained by inserting spaces into (or at either end of) each string so that the resulting strings have same length, say, l. Then, the strings are put in an array with k rows of l columns each. Typically, a score is assigned to an alignment to measure its quality. Different scoring schemes are proposed in the literature. In one common family of schemes, the score of an alignment is taken to be the sum of the scores of the columns; the score of a column is defined as some function of the symbols in the column. In standard versions, this function has a high value when all the symbols in the column agree and its value drops off as there is greater and greater variation in the column. The objective is to find an alignment with the maximum score. Note that in the case of related sequences, it is not clear how these scoring schemes serve the purpose of discovering the common ancestor, from which each sequence is generated. In fact, it is easy to construct examples where the optimum alignment will not produce the common ancestor.

In this paper, we initiate a study of a string reconstruction problem in an idealized evolutionary model where the only possible mutations are restricted to random deletions. Our goal is to understand for what parameter values (evolutionary rates and number of sequences), enough information is retained such that we can *completely recover* the original string by suitably "aligning" the observed samples.

For the rest of the paper, we assume that the alphabet is $\{0,1\}$, because sequences from a larger alphabet can be inferred more easily. Specifically, if the actual transmitted string comes from an alphabet Σ, one can consider $|\Sigma|$ different mappings from Σ to $\{0,1\}$, each of which maps exactly one letter in Σ to 1, solve the induced inference problems on $\{0,1\}$-sequences and from these solutions reconstruct the solution for the original problem.

1.1 Our Techniques and Results: We investigate a natural alignment algorithm, called, Bitwise Majority Alignment. The idea behind this algorithm is to recover each bit of the transmitted string by simply considering a majority vote from the received strings. As the algorithm progresses, while recovering the bit i in the transmitted string, it may be positioned at completely different positions in the received strings. Our first main result is that for all but a vanishingly small fraction of length-n strings, if each symbol is deleted with probability $q = O(\frac{1}{\log n})$, Bitwise Majority Alignment can reconstruct the transmitted string with high probability using $O(\log n)$ received strings. The bound on the number of received strings needed is essentially tight since $\Omega(\frac{\log n}{\log\log n})$ transmissions are necessary to merely ensure that every bit in the transmitted string is successfully received in at least one of the strings. Our second main result focuses on arbitrary sequences and we show that for deletion probability as large as $\Omega(1/n^{1/2+\epsilon})$, we can recover a very close approximation (a string with identical structure) of the original string by examining only $O(1/\epsilon)$ samples. The algorithm used for reconstruction is a slightly modified version of Bitwise Majority Alignment—the modification is critical for handling long runs of 1's or 0's in the transmitted string. By using $O(n \log n)$ samples, we can exactly recover the original string with high probability. Notice that $n^{\Omega(1)}$ samples are necessary for exact construction. These results are in strong contrast to the work of Levenshtein [6] which shows that in an adversarial model of deletions, $n^{\Omega(d)}$ distinct subsequences are necessary to reconstruct t when each received string has d arbitrary deletions.

The central idea underlying our techniques is to show that Bitwise Majority Alignment has self-correcting behavior whereby even though locally, some of the received strings vote incorrectly, these votes do not overwhelm the correct majority and moreover, the majority vote helps put these received strings back on track. The proof of this recovery property requires looking far ahead into the structure of the transmitted string and establishing that the errors do not accumulate in the meantime. It is an interesting phenomenon that even though Bitwise Majority Alignment operates on a local view of received strings, its correctness relies on global properties of the received strings.

2 Preliminaries

We consider the following problem. A binary string t of length n is transmitted m times over a deletion channel; that is, the jth transmission results in a binary string r_j that is created from t by deleting each bit independently with probability q. We seek an algorithm to correctly reconstruct t from the received strings $r_1, \ldots, r_m$ with probability at least $1-\delta$, for a given error probability $\delta > 0$.

DEFINITION 2.1. (RUN) *A* run *of 1's (also called a* 1-run*) in a string t is a maximal substring of consecutive 1's. A run of 0's is defined analogously. We denote the ith run of string t by L_i and the length of L_i by l_i.*

Note that a string t is a concatenation of runs, alternating between runs of 1's and runs of 0's. After a transmission of t, the total number of runs in the received string may be less than the total number of runs in t due to deletions. In the event that a run is completely deleted, the preceding run and the following run, which are of the same kind, get concatenated together.

DEFINITION 2.2. *We say that runs L_i and L_{i+2} are* merged *during transmission if all the bits in run L_{i+1} are deleted during transmission to combine L_i and L_{i+2}.*

2.1 The Bitwise Majority Alignment Algorithm: In this section, we describe our basic reconstruction algorithm. The input to the algorithm is a $m \times n$ array R where each row corresponds to one of the received strings. Since received strings may be (much) shorter in length than the transmitted string, we assume that each received string is padded at the end with special characters so that the length of each string is exactly n. Throughout the execution of the algorithm, we maintain a pointer $c[j]$ for each received string r_j that points to the leftmost bit in r_j that has not been processed so far. Our algorithm scans the received strings simultaneously from left to right. For each position i, the algorithm determines t_i as the next bit of the majority of the received strings. Then the pointers for the received strings that voted with the majority are incremented by one, while other pointers remain unchanged. Thus as the reconstruction progresses, the algorithms is looking at different bit positions in the received strings. Given array R of the received strings, string t is reconstructed as follows.

Bitwise Majority Alignment(R)
  Let $c[j] = 1$ for all $j = 1, \ldots, m$
  For $i = 1$ to n
    Let b be the majority over all j of $R[j, c[j]]$
    $t[i] \leftarrow b$.
    Increment $c[j]$ for each j such that $R[j, c[j]] = b$.

During this alignment process, suppose the algorithm is currently reconstructing a run L_i of t. At this point, the counters in various received strings may be pointing to bits from the earlier or later runs in t. For example, if an entire run is deleted during the transmission, the counter may be pointing to the next run. The next definition classifies these "misalignments" according to how far a received string actually is from the run being constructed.

DEFINITION 2.3. *We say that received string r_j (or the alignment for r_j) is h* ahead *if, while determining the bits in run L_i of t, the algorithm processed all the bits of r_j coming from run L_{i+2h}. Analogously, we say that the received string r_j is h* behind *when bits from an earlier run, namely, L_{i-2h}, are used to determine the bits of L_i. If at the ith step, the alignment of r_j is 0 runs ahead, then we say that alignment of r_j is* good *at this step.*

3 Reconstructing Random Strings

In this section, we show that for $q = \frac{1}{d \log n}$ for some suitably large constant d and $m = O(\log n)$, the Bitwise Majority Alignment algorithm reconstructs the transmitted string t for all but a vanishingly small fraction of t's in $\{0,1\}^n$ with high probability. We start with some simple properties of random strings that will be crucial in our analysis.

LEMMA 3.1. *A random binary string of length n satisfies the following properties with probability at least $1 - (2/n)$:*

- *No run has length greater than $2 \log n$;*
- *There exist constants k, k' with $k' \leq k$, such that if we consider any $2k \log n$ consecutive runs, say, with lengths $l_i, l_{i+1}, \ldots, l_{i-1+2k\log n}$, then, for all $h \leq k' \log n$, there exists i' such that $i \leq i' \leq i - 1 + 2k \log n$ and $l_{i'} < l_{i'+2h}$.*

Proof. The probability that the length of a given run exceeds $2 \log n$ is at most $1/n^2$. Since there are at most n runs, the probability that there exists a run of length greater than $2 \log n$ is at most $1/n$.

Consider the probability that there do not exist two runs in a given segment of $2k \log n$ runs such that they are $2h$ runs apart and the second run is of strictly greater length. Let the run lengths in the segment be $l_i, \ldots l_{i-1+2k\log n}$. This means that

$$l_{i+\Delta} \geq l_{i+2h+\Delta} \geq l_{i+4h+\Delta} \geq \cdots \tag{3.1}$$

for all offsets $\Delta = 0, 1, \ldots, 2h-1$. The probability that a run is greater than or equal in length to another is $2/3$ since the probability that the two runs are of equal length is $\sum_{i \geq 1} 2^{-2i} = 1/3$ and, given that the runs are not equal in length, there is a $1/2$ probability that the second run is longer. Hence, the probability that Inequality (3.1) holds true for all $\Delta \leq 2h - 1$ is less than $(\frac{2}{3})^{k \log n}$. Hence, by union bounds, the probability that there exists a segment such that for some $1 \leq h \leq k' \log n$, all runs $2h$ apart are such that the latter run is at most as long as the earlier run is bounded above by

$$nk' \log n \left(\frac{2}{3}\right)^{k \log n} = \frac{k' \log n}{n^{k(\log 3 - 1) - 1}}.$$

With probability $\geq 1 - 2/n$ both conditions are met.

Subsequently, consider only transmitted strings t that satisfy the conditions of Lemma 3.1. The analysis of the algorithm hinges on the following observation: If the received string r_j is h ahead before the beginning of run L_i and $l_i < l_{i+2h}$, then r_j will have more than l_i bits in the next run, provided that r_j does not lose at least $l_{i+2h} - l_i$ bits of L_{i+2h}. Thus, at the end of the run, the pointer $c[j]$ for r_j will be advanced only l_i positions, thus not using up all the bits from L_{i+2h}. As a result, the alignment of r_j will now be at most $h-1$ runs ahead. In other words, the alignment of r_j is corrected by at least one run when r_j is "processing" a run longer than l_i. For the t we are considering there exist many such pairs of runs for each $h \leq k' \log n$. So, for each segment of string t with $2k \log n$ consecutive runs, a misaligned received string will at least partially correct its alignment with high probability. On the other hand, the alignment of a received string can get still further ahead due to runs that are completely deleted during the transmission.

We model this alignment process for each received string with a random walk on a line starting from 0. At the end of each segment of $2k \log n$ runs, the random walk takes a step reflecting the change in the alignment during this segment. For example, if the received string was already 2 ahead before the segment and another run in this segment is deleted, the random walk may move from state 2 to state 3. Similarly, if an alignment is corrected as described above, the random walk may move from state 2 to state 1. We would like to show that at any step in the reconstruction, the majority of the random walks corresponding to the received strings are at state 0 with high probability. This will ensure a successful reconstruction since the majority of the received strings is processing bits from the correct run.

The transition probabilities for the random walk described above depends on the particular string t and the segment of t that is being processed. For example, if there are many runs of length 1 in the current segment, then the probability of making a step in the positive direction increases due to the higher chances of deletion of a complete run. To simplify the analysis, we instead use another random walk R that does not depend on the particular string t. This new random walk is provably more prone to drift away from state 0 than the original one. Hence, it is a pessimistic approximation to the original random walk and safe to use in the analysis. The random walk R is defined with states $[0, \infty]$. It starts at state 0 and has transition matrix P, where

$$P_{i,j} = \begin{cases} \alpha^{j-i} & \text{for } i < j \ \& \ i < k' \log n \\ \beta & \text{for } 0 < i < k' \log n \ \& \ j = i-1 \\ \frac{1}{1-\beta}\alpha^{j-i} & \text{for } i < j \ \& \ i \geq k' \log n \\ 1 - \sum_{k \neq i} P_{i,k} & \text{for } i = j \end{cases}$$

for $\alpha = (2k/d)$ and $\beta = e^{-2k/d}$

LEMMA 3.2. *The random walk R dominates the real random walk of the alignment process.*

Proof. Consider the alignment of a received string r_j at the beginning of a specific segment of t. Suppose at the start of this segment, r_j is $h \leq k' \log n$ ahead. We know that there exist runs L_i and L_{i+2h} such that $l_i < l_{i+2h}$ that can fix the misalignment of r_j by 1. If no runs get completely deleted in this segment of t and L_{i+2h} is transmitted intact, then r_j drops back to being at most $h-1$ ahead by the end of the segment as mentioned in the discussion following Lemma 3.1. The probability that L_{i+2h} is intact is at least $(1-q)^{2 \log n}$. The probability that no run in this segment gets completely deleted is at least $(1-q)^{2k \log n}$. Hence, the probability that r_j drops back to being at most $h-1$ ahead by the end of the segment is at least

$$\begin{aligned} (1-q)^{(2+2k)\log n} &= (1 - \frac{1}{d \log n})^{(2+2k)\log n} \\ &\geq (1 - \frac{1}{d \log n})^{3k \log n} \\ &\geq e^{-2k/d}. \end{aligned}$$

The received string r_j moves from being h runs ahead to being $j > h$ runs ahead only if $j-h$ runs are deleted. The probability of this happening is at most

$$\binom{2k \log n}{j-h} q^{j-h}(1-q)^{2k \log n - (j-h)} \leq \left(\frac{2k}{d}\right)^{j-h}.$$

We are interested in when the alignment random walk for a received string is in state 0 (a.k.a. *good*) since this corresponds to it voting correctly in the Bitwise Majority Alignment. To this end we use the following lemma whose proof can be found in the appendix.

LEMMA 3.3. *We can choose constants k and d such that after any $u \in [n]$ steps, the probability that random walk R is at state 0 is at least 19/20.*

LEMMA 3.4. *Consider $m = \Theta(\log n)$ instances of the random walk R. With high probability, for each of the first n steps $3m/4$ of the instances are at state 0.*

Proof. The probability that a given instance is at state 0 for each of these steps is at least 19/20 by Lemma 3.3. The lemma follows by an application of Chernoff bounds.

LEMMA 3.5. *If at least $m' = \Theta(\log n)$ strings are good at the start of each segment then with high probability at least $8m'/9$ strings are good at the beginning of every run.*

Proof. The probability that a good string turns bad during a segment is at most $2kq \log n = 2k/d$. The lemma follows by an application of Chernoff bounds.

LEMMA 3.6. *For $m = \Theta(\log n)$ received strings, with high probability, at the start of every run, at least $2/3$ fraction of the received strings are good.*

Proof. This follows from Lemma 3.3, Lemma 3.4 and Lemma 3.5.

THEOREM 3.1. *The* Bitwise Majority Alignment *algorithm correctly reconstructs the string t from $m = \Theta(\log n)$ received strings for $q = O(1/\log n)$ with high probability.*

Proof. We prove the theorem by induction on the execution of the algorithm. Assume that we have correctly determined the bits of $L_1, L_2, \ldots, L_{i-1}$, and that no received string has fallen behind in the alignment. By Lemma 3.6, the majority of the received strings are good at the start of L_i. Moreover, given that $l_i \leq 2 \log n$ it can be shown by Chernoff that the majority of the received strings have not lost any bits of L_i and are pointing to the first received bit of L_i. Hence, the Bitwise Majority Alignment will correctly deduce the bits of L_i.

4 Reconstructing Arbitrary Strings

In this section, we show that any n-bit string can be reconstructed with high probability by using $m = O(1/\epsilon)$ traces provided the deletion probability q is $1/n^{1/2+\epsilon}$ for any $\epsilon > 0$. The Bitwise Majority Alignment algorithm can not be directly used on arbitrary strings. In particular, consider a string that starts with a long run of 1's. Clearly, different received strings will see a different number of 1's for this run. While scanning from left to right, the first location where more than $m/2$ strings vote for a 0, could lead to splitting this run in the trace that received the maximum number of 1's. Other difficulties include recognizing when a large run absorbs a nearby small run and the merger can not be locally detected. In what follows, we show that even though locally a majority of received strings may present a misleading view to Bitwise Majority Alignment, a small modficiation of Bitwise Majority Alignment can correct and recover from these distortions to successfully reconstruct the transmitted string.

A run is called *long* if its length is at least $\sqrt{n}$ and is *short* otherwise. An *alternating sequence* in a string is a sequence of length at least two such that each run in the sequence has length 1 (e.g., $010101\ldots$). A bit is called the *first bit* of an alternating sequence if it is a run of length 1 that either follows a run of length greater than 1 or it is the first bit in the transmitted string. A *delimiting run* for an alternating sequence is the first run of length at least two that follows the alternating sequence. We start with a simple lemma about properties of the received strings. We fix $m = \lceil 6/\epsilon \rceil$ from here on. The lemma below follows from elementary probability calculations that show that the probability of violating any promise is $o(1)$.

LEMMA 4.1. *A collection of m received strings generated by deleting with probability q, with high probability satisfies the following:*

(P1) The first bit in the transmitted string is not deleted in any received string.

(P2) Bits at both ends of any long run are preserved in all strings. Moreover, at most $m/3$ bits are lost among all received strings during the first $\sqrt{n}$ bits of any long run.

(P3) For any two adjacent locations $i, i+1$ in the transmitted string, at most 1 bit is lost among all m received strings. If a received string loses the first bit of a run, then the first bit of each of the next two runs is intact in all received strings.

(P4) If the bit just before or just after a short run is lost in any of the received strings, then at most 1 bit is lost in the run itself in all received strings.

(P5) At most $m/3$ bits are lost in any window of size $\sqrt{n}$. Thus at least $2m/3$ received strings see a short run intact.

(P6) If the first bit of an alternating sequence is deleted in any received string then no more bits are lost in the alternating sequence in all received strings. Moreover, the first two bits of its delimiting run and the first bit of the run following it are then preserved in all received strings.

We first show that if long runs stay intact, then Bitwise Majority Alignment can directly reconstruct the transmitted string.

LEMMA 4.2. *If all received strings obey the properties outlined in Lemma 4.1, and all long runs are intact in each received string, then* Bitwise Majority Alignment *exactly determines the transmitted string.*

Proof. Let $L_1 L_2 L_3 \ldots L_k$ be the transmitted string with L_i being the ith run. We will use induction on phases where each phase comprises of one or more consecutive runs of the transmitted string. The first phase starts at L_1. In order to prove correctness of

Bitwise Majority Alignment, we maintain the following invariant after each phase. Suppose the current phase terminated with the run L_{i-1}. Then (i) at least $m-1$ strings point to the first received bit of L_i, and (ii) at most one string points either to the second received bit in L_i or to the first received bit in L_{i+1}.

The base case clearly holds (by $P1$). Assume inductively that we have successfully completed the first z phases and the invariant holds at the begining of the current phase. Let L_i be the first run of the phase. Assume without any loss of generality that L_i is a run of 1's.

Suppose one of the counters is pointing to a 0-run. Then L_i must be a run of length 1. Bitwise Majority Alignment will recover this run correctly and no further deletions could have happened in L_i or L_{i+1} in any other received string (by $P3$). At the end of this iteration, all counters once again point to the first received bit in L_{i+1}. We consider the current phase over and the next phase starts at L_{i+1}. From here on, all strings point to a 1-run.

Now if majority says that this is a long run, it must be true since long runs are intact by $P2$ and Bitwise Majority Alignment will correctly recover this run since the bit following it is intact in all received strings. At the end of this, all counters point to the first received bit in L_{i+1}. We again consider the current phase over and the next phase starts at L_{i+1}.

So let us assume that each counter points to a bit in L_i and L_i is a short run. By $P5$, a majority of the strings sees L_i intact and thus we know the exact length of L_i. Bitwise Majority Alignment will recover L_i correctly (by $P4$) but we need to show that at the end of this process, the counters will be suitably positioned. We consider the following cases:

- If in some received string, the length of the run is less than ℓ_i, then the only possibility is that a deletion occured in this string and furthermore, the first bit of L_{i+1} must have been successfully received in at least $m-1$ strings (by $P4$). In this case, the counters will indeed be positioned correctly and a new phase can start at L_{i+1}.

- If in some received string the length of the run is greater than ℓ_i, we know that a merger of L_i with L_{i+2} must have taken place. Moreover, by P_3 we know that L_{i+1} is a 0-run of length 1 and that the first bit of L_{i+3} must be intact in all received strings (by $P3$). It is easy to verify that Bitwise Majority Alignment will correctly recover L_i, L_{i+1}, and L_{i+2} and after L_{i+2} is recovered, the counters in each string will point to the first received bit of L_{i+3}. We will terminate this phase at L_{i+2}.

- All received strings see the same run length. In this case, either all received strings see only the bits in L_i or one of the strings lost a bit in L_i, the entire run L_{i+1}, and merged with the run L_{i+2}. This is only possible if $\ell_{i+1} = \ell_{i+2} = 1$. Thus if either ℓ_{i+1} or ℓ_{i+2} has length greater than 1, we can simply terminate the current phase here. Otherwise, we are at the begining of an alternating sequence, starting at L_{i+1}. Let L_j be the delimiting run for this alternating sequence. After Bitwise Majority Alignment recovers L_i, one of the received strings may be positioned at L_{i+3} while all other strings are positioned at the first received bit of L_{i+1} (by $P3$ and $P6$). First consider the case when no string is pointing to L_{i+3}. Then since no two succesive bits get deleted, Bitwise Majority Alignment correctly recovers the entire sequence with every string positioned at the first received bit of L_j. Otherwise, a received string points to L_{i+3} and by $P6$, no more bits are lost in L_{i+1} through L_{j-1} in any other received string. As we run Bitwise Majority Alignment, while processing the run L_{j-1}, in this received string we will see a run of opposite parity (i.e. L_j) while all other strings point to a run of length 1. Bitwise Majority Alignment will insert L_{j-1} in this string and the pointer for this string points to the second received bit in L_j. For all other strings, it points to the first bit of L_j. In either case, the pointers in all strings are positioned correctly at the end of this phase that we terminate at L_{j-1}.

The analysis above crucially relies on the long runs being intact. It is easy to see that a long run will lose many bits in each of the received strings. As a result, Bitwise Majority Alignment algorithm can get mispositioned in many received strings while processing a long run. We next show that a natural modification of Bitwise Majority Alignment can handle long runs as well. In the modified Bitwise Majority Alignment, when we arrive at a long run, we simply increment the counters to the begining of the next run. We set the length of the run to be the median length scaled by a factor of $1/(1-q)$. Otherwise, the algorithm behaves identically to Bitwise Majority Alignment. The simple observation here is that even in presence of deletions in long runs, we always recognize a long run.

LEMMA 4.3. *If at least $m-2$ strings point to the first received bit of a run L_i, then we can always determine whether or not L_i is a long run.*

Proof. We claim that if a majority of the strings have at least $\sqrt{n}$ bits in the ith run, then l_i must be long and it short otherwise. Suppose L_i is a long run. Then by P5, at least $2m/3$ strings must have received the first $\sqrt{n}$ bits of this run intact. So the majority must see a run of length at least $\sqrt{n}$. Now suppose L_i is a short run. Then by $P3$, at most one string participates in a merger Therefore, in majority of the received strings we must see a run of length less than $\sqrt{n}$.

LEMMA 4.4. *Suppose $t = L_1L_2L_3 \cdots L_k$ is the transmitted string. If all received strings obey the properties outlined in Lemma 4.1, then the modified* Bitwise Majority Alignment *reconstructs a string $t' = L_1'L_2'L_3' \cdots L_k'$ such that $\ell_i' = \ell_i$ whenever L_i is a short run and $\ell_i' = \ell_i + o(\ell_i)$ otherwise.*

Proof. The proof is similar to that of Lemma 4.2. We can essentially maintain the same inductive invariants, and using Lemma 4.3, we can recognize whenever L_i is a long run. At this point, we update the counters in each string to the first received bit in the next run. The length of L_i is estimated by scaling the median observed length by a factor of $1/(1-q)$.

4.1 Recovering Lengths of Long Runs: We now describe how to exactly determine the length of the long runs when $q = O(1/\sqrt{n})$. The main idea is to repeat the modified Bitwise Majority Alignment algorithm $\Theta(nq \log n)$ times using a new set of m received strings each time.

Let $B(n, p)$ denote the binomial distribution with parameters n and p, that is, the sum of n independent Bernoulli trials, each with success probability p. The next lemma is a variant of Chernoff bounds.

LEMMA 4.5. *Let X_i's for $i = 1, \ldots, n$ be independent Bernoulli trials each with success probability p, and $X = \sum_i X_i$. Let $\sigma = \sqrt{np(1-p)}$ be the standard deviation of X. Then, for $k > 0$,*

$$\Pr(|X - np| \geq k\sigma) \leq 2\exp(-k^2/6).$$

We now describe how we determine the length of a long run L_i. Without loss of generality, let L_i be a long run of 1's. Consider the jth repetition of the algorithm. Let $z_1, \ldots, z_m$ be the observed lengths for L_i in each received string. The median X_j of all z_i's is the estimate for l_i given by the jth repetition of the algorithm.

In a given repetition, the majority of the received strings do not lose the delimiting bits for L_i with high probability. Hence, for the majority of the repetions, X_j, is distributed according to the binomial distribution $B(l_i, 1-q)$, the sum of l_i Bernoulli trials each with success probability $(1-q)$. When the run L_{i-1} or run L_{i+1} is lost in the majority of the received strings in a given repetition of the algorithm, X_j value includes the noise added by the concatenated runs.

Let X be the median of X_j's. With high probability, X_j is within $\log n$ times the standard deviation of $B(l_i, 1-q)$, namely $O(\sqrt{l_i q}\log n)$. Hence we can eliminate all X_j's that differ from X by at least $O(\sqrt{Xq}\log n)$ since they are guaranteed to be noisy estimates. The remaining X_j's either have no noise or have noise at most $O(\sqrt{l_i q}\log n)$ (due to concatenations). Let $N = \Theta(nq \log n)$ be the number of these X_j's. The final estimate for l_i is obtained by taking the average of these X_j's.

In expectation, either run L_{i-1} or run L_{i+1} will be lost in $O(q)$ fraction of the repetitions. Hence, using Chernoff bounds, we can show that in no more than $O(q \log n)$ fraction of the repetitions, run L_{i-1} or run L_{i+1} is lost. So, $O(q \log n)$ is the fraction of the noisy estimates.

First, we prove that if there were no noisy estimates, then the sum of X_j's divided by $(1-q)N$ is within an additive $1/3$ of l_i with high probability, thus it determines l_i (by simply rounding up or down to the nearest integer value). Using Chernoff bounds,

$$\begin{aligned}
&\Pr\left(\left|\sum X_j - (1-q)Nl_i\right| > (1-q)N/3\right) \\
&= \Pr\left(\left|\sum X_j - (1-q)Nl_i\right| > \left(\frac{\sqrt{N(1-q)}}{3\sqrt{ql_i}}\right)\sqrt{Nl_iq(1-q)}\right) \\
&\leq 2\exp(-N(1-q)/54ql_i) \leq 2\exp(-O(N/qn)) \leq \frac{1}{n}
\end{aligned}$$

Now, we will figure out the contribution of the noise to this estimate. Recall that only $O(q \log n)$ fraction of the estimates can have a noise of $\pm O(\sqrt{l_i q}\log n)$. Hence, the total noise contribution in the summation of X_j's is $O(Nq^{3/2}\sqrt{l_i}\log^2 n)$, and thus the average noise contribution is $O(\log^2 n/n^{1/4})$. Since the noise in the estimate is $o(1)$, it does not change the result of the up/down rounding to the nearest integer. We thus determine l_i with high probability.

4.2 A lower bound: In this section, we outline an argument that $\Omega(nq(1-q))$ received strings are necessary for exact reconstruction of the transmitted string. Let $t_0 = 1^{n/2}0^{n/2}$ and $t_1 = 1^{(n/2)+1}0^{(n/2)-1}$ be two strings. We claim that $\Omega(nq(1-q))$ samples are needed to distinguish between t_0 and t_1 when transmitted over a deletion channel with deletion probability q. Hence, showing the optimality of our algorithm from the previous section.

Distinguishing between t_0 and t_1 boils down to distinguishing $B(n/2, 1-q)$ from $B((n/2)+1, 1-q)$ using independent samples. The density functions of $B(n/2, 1-q)$ and $B(n/2+1, 1-q)$ are such that the former dominates until (and including) $n(1-q)/2$, and the latter dominates afterwards. Also, the L_1 distance between them is $O(1/\sqrt{nq(1-q)})$. Hence, distinguishing between t_0 and t_1 is the same as distinguishing an ϵ-biased coin from a fair coin with $\epsilon = O(1/\sqrt{nq(1-q)})$. It is well known that this requires $\Omega(\epsilon^{-2}) = \Omega(nq(1-q))$ coin flips. Hence, $\Omega(nq(1-q))$ samples are required for the exact reconstruction of the lengths of the runs.

References

[1] N. Alon and J. Edmonds and M. Luby, *Linear Time Erasure Codes with Nearly Optimal Recovery*, 36th Annual Symposium on Foundations of Computer Science, pp. 512-519,1995.

[2] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, 1997.

[3] V. I. Levenshtein, *Binary codes capable of correcting deletions, insertions and reversals* (in Russian), Doklady Akademii Nauk SSSR, **163** (1965), no. 4, 845–848. Enlgish translation in Soviet Physics Dokl., **10** (1966), no. 8, 707–710.

[4] V. I. Levenshtein, *Binary codes capable of correcting spurious insertions and deletions of ones* (in Russian), Problemy Peredachi Informatsii, **1** (1965), no. 1, 12–25. Enlgish translation in Problems of Information Transmission, **1** (1965), no. 1, 8–17.

[5] V. I. Levenshtein, *On perfect codes in the deletion/insertion metric* (in Russian), Diskret. Mat. **3** (1991), no. 1, 3–20. English translation in Discrete Mathematics and Applications **2** (1992), no. 3, 241–258.

[6] V. I. Levenshtein, *Efficient reconstruction of sequences*, IEEE Trans. Inform. Theory **47** (2001), no. 1, 2–22.

[7] L. J. Shulman and D. Zuckerman, *Asymptotically good codes correction insertions, deletions and transpositions*, IEEE Trans. Inform. Theory **45** (1999), no. 7, 2552–2557.

A Proof of Lemma 3.3

Proof. Let step u' be the last visit to state 0 prior to step u. Let $v = u - u'$. We first upper bound the probability that the random walk is not at state 0 in v steps. Then, by summing on all possible values for v, we prove the lemma.

Let X be the sum of the lengths of forward moves, Y be the number of "stay at the same position" moves and Z be the number of backward moves between the steps u' and u. Clearly $X+Y+Z \geq v$. Thus, we would like to upper bound $\Pr(X > Z)$.

$$\begin{aligned} & \Pr(X \geq Z) \\ = \;& \Pr(X \geq Z \cap Y \geq Z) + \Pr(X \geq Z \cap Z \geq Y) \\ \leq \;& \Pr(Y \geq Z) + \Pr(X \geq v/3) \end{aligned}$$

Now,

$$\begin{aligned} \Pr(Y \geq Z) &= \sum_{j=0}^{v/2} \Pr(Z=j)\Pr(Y \geq Z | Z=j) \\ &\leq \sum_{j=0}^{v/2} \Pr(Z=j) \\ &\leq e^{-\beta v(1-1/(2\beta))^2/2} \end{aligned}$$

To bound $\Pr(X \geq v/3)$, we consider X as the sum of v random variables X_j with the following distribution: $X_j = i$ with probability α^i (for $i > 0$) and $X_j = 0$ with probability $\frac{1-2\alpha}{1-\alpha}$. Consider a new random variable X' which is the sum of v random variables X_j' with the following distribution: $X_j' = i$ with probability $(2\alpha)^{i-1}(1-2\alpha)$ (for $i \geq 1$) Note that

$$\Pr(X \geq v/3) \leq \Pr(X' \geq v/3 + v)$$

since $\Pr(X_j = 0) \geq \Pr\left(X_j' = 1\right)$ but $\Pr(X_j = l) \leq \Pr\left(X_j' = l+1\right)$ for $l \geq 1$.

Furthermore, note that X' is a sum of geometric random variables, that is, the negative binomial distribution. Hence, we can bound $\Pr(X \geq v/3)$ as follows:

$$\begin{aligned} & \Pr(X \geq v/3) \\ \leq \;& \Pr(X' \geq 4v/3) \\ = \;& \sum_{x \geq 4v/3} \binom{x-1}{v-1} (1-2\alpha)^v (2\alpha)^{x-v} \\ \leq \;& \left(\frac{1-2\alpha}{2\alpha}\right)^v \left(\frac{e}{v-1}\right)^{v-1} \sum_{x \geq 4v/3} x^{v-1}(2\alpha)^x \\ \leq \;& \left(\frac{1-2\alpha}{2\alpha}\right)^v \left(\frac{4ve}{3(v-1)}\right)^{v-1} 2\alpha^{4v/3} \sum_{j \geq 0} [2e\alpha]^j \\ \leq \;& (4\alpha^{1/3})^v \frac{1}{1-2e\alpha} \end{aligned}$$

Hence, we have shown that

$$\Pr(X \geq Z) \leq (4\alpha^{1/3})^v \frac{1}{1-2e\alpha} + e^{-\beta v(1-1/(2\beta))^2/2}.$$

Hence, we can conclude that for $v > k' \log n$, the probability that the random walk does not return to state 0 in v steps is negligible. By summing the expression above over all v such that $1 \leq v \leq n$, we show that the probability that a random walk is not at state 0 at step u is less than $1/20$ for suitable contants k and d.