RECONSTRUCTING STRINGS FROM RANDOM TRACES

SOURCE CODING (EE5390)

March 28, 2021

SHAIK MASTAN VALI
K SRIKANTH

INDIAN INSTITUTE OF TECHNOLOGY, HYDERABAD

Table of contents

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 show that for random strings t, we can reconstruct the original string (w.h.p.) for q = $\mathbb{O}(1/\log n)$ using only $\mathbb{O}(\log n)$ 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 = $\mathbb{O}(1/n^{1/2+\epsilon})$ using $\mathbb{O}(1)$ samples. In this case, using $\mathbb{O}(n\log n)$ samples, we can reconstruct the original string exactly.

⇒ 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.

Introduction

$\Rightarrow$ Let $t = t_1\, t_2\, ...t_n$ be a string over an alphabet $\sum$. Suppose we are given a collection of random sub-sequences (traces) of t where each random subsequence is obtained independently as follows:

- $\rightarrow$ For each i, the symbol t_i is deleted independently with probability q.
- $\rightarrow$ The surviving symbols are concatenated to produce the subsequence.

$\Rightarrow$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.”

$\Rightarrow$In the literature, various error correcting codes for the deletion channel are studied . Such codes allow one to reconstruct the transmitted string (from a single observation) when the transmitted string is actually a codeword.

$\Rightarrow$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.

Introduction(Cont....)

⇒We also would like to note that another class of error correcting codes called erasure codes 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)

→ (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 sub-patterns from a set of sequences known to be biologically related.

Introduction(Cont....)

⇒ 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.

⇒ Here, 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 , 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 $\sum$, one can consider $|\sum|$ different mappings from $\sum$ to {0,1}, each of which maps exactly one letter in $\sum$ to 1, solve the induced inference problems on 0,1-sequences and from these solutions reconstruct the solution for the original problem.

Techniques and Results

$\Rightarrow$ 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.

$\Rightarrow$ 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 = $O(\frac{1}{logn})$, Bitwise Majority Alignment can reconstruct the transmitted string with high probability using O(logn) received strings.

$\Rightarrow$Our second main result focuses on arbitrary sequences and we show that for deletion probability as large as $\Omega(\frac{1}{n^{1/2+\epsilon}})$, we can recover a very close approximation (a string with identical structure) of the original string by examining only $O(\frac{1}{\epsilon})$ samples.

$\Rightarrow$ The central idea underlying our techniques is to show that Bitwise Majority Alignment has selfcorrecting 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.

Preliminaries

$\Rightarrow$ We consider the following problem. A binary string t of length n is transmitted m times over a deletion channel; that is, the j^{th} transmission results in a binary string r_j that is created from t by deleting each bit independently with probability q.

$\Rightarrow$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 $\delta > 0$.

Def-1: (Run) A run of 1's (also called a 1-run) in a string t is a maximal sub-string of consecutive 1's. A run of 0's is defined analogously. We denote the i^{th} run of string t by L_i and the length of L_i by l_i.

Def-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}.

Def-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-2}, are used to determine the bits of L_i. If at the L^{th} step, the alignment of r_j is 0 runs ahead, then we say that alignment of r_j is good at this step.

The Bitwise Majority Alignment Algorithm

$\Rightarrow$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

Bit wise 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 $R[j, c[j]] = b$.

Reconstructing Random Strings

Lemma 6.1. A random binary string of length n satisfies the following properties with probability at least $1 - (2/n)$:

- No run has length greater than 2logn;
- 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 $l_{i'} < l_{i'+2h}$

$\Rightarrow$Subsequently, consider only transmitted strings t that satisfy the conditions of Lemma 3.1. The analysis of the algorithm hinges on the following observation:

$\Rightarrow$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}.

$\Rightarrow$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.

Reconstructing Random Strings(Conti..)

⇒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.

Reconstructing Random Strings(Conti..)

⇒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 $\beta = e^{-2k/d}$

Reconstructing Random Strings(Conti..)

Lemma 6.2 The random walk R dominates the real random walk of the alignment process.

Lemma 6.3. 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 6.4 Consider $m = \Theta(\log n)$ instances of the random walk R. With high probability, for each of the first n steps $3\,\mathrm{m}/4$ of the instances are at state 0 .

Lemma 6.5 If at least $m' = \Theta(\log n)$ strings are good at the start of each segment then with high probability at least $8\,\mathrm{m}'/9$ strings are good at the beginning of every run.

Lemma 6.6 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.

Theorem 6.1 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.

Reconstructing Arbitrary Strings

$\Rightarrow$ 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($\mathrm{e.g.}$, 010101 . . .).

$\Rightarrow$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.

Lemma 7.1. A collection of *m* received strings generated by deleting with probability *q*, with high probability satisfies

Lemma 7.2 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.

Lemma 7.3 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 Li is a long run.

Lemma 7.4 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 $t' = L'_1L'_2L'_3 \ldots L'_k$ such that $\ell'_i = \ell_i$ whenever L_i is a short run and $\ell'_i = \ell_i + o\,(\ell_i)$ otherwise.

Recovering Lengths of Long Runs:

$\Rightarrow$ 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.

$\Rightarrow$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 7.5 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 $k > 0$,

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

$\Rightarrow$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 j th 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 j th repetition of the algorithm.

$\Rightarrow$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

Recovering Lengths of Long Runs(Conti....):

$\Rightarrow$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.

$\Rightarrow$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 $O\left(\sqrt{l_i q}\log n\right)$. 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.

$\Rightarrow$The remaining X_j 's either have no noise or have noise at most $O\left(\sqrt{l_i q}\log n\right)$ (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.

$\Rightarrow$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.

Recovering Lengths of Long Runs(Conti....):

$\Rightarrow$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,

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

$\Rightarrow$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 $\pm O\left(\sqrt{l_iq}\log n\right)$ Hence, the total noise contribution in the summation of X_j 's is $O\left(Nq^{3/2}\sqrt{l_i}\log^2 n\right)$, and thus the average noise contribution is $O\left(\log^2 n/n^{1/4}\right)$.

$\Rightarrow$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.

Lower bound:

$\Rightarrow$ A 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.

$\Rightarrow$ 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.

$\Rightarrow$ 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.

$\Rightarrow$ 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 $O(1/\sqrt{nq(1-q)})$.

$\Rightarrow$ Hence, distinguishing between t_0 and t_1 is the same as distinguishing an $\epsilon-$ biased coin from a fair coin with $\epsilon = O(1/\sqrt{nq(1-q)})$.

$\Rightarrow$ It is well known that this requires $\Omega\left(\epsilon^{-2}\right) = \Omega(nq(1-q))$ coin flips. Hence, $\Omega(nq(1-q))$ samples are required for the exact reconstruction of the lengths of the runs.

The End


भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad