A New Algebraic Approach for String Reconstruction from Substring Compositions

Utkarsh Gupta and Hessam Mahdavifar
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109, USA
Emails: utkarshg@umich.edu, hessam@umich.edu

***Abstract*—We consider the problem of binary string reconstruction from the multiset of its substring compositions, i.e., referred to as the substring composition multiset, first introduced and studied by Acharya *et al.* We introduce a new algorithm for the problem of string reconstruction from its substring composition multiset which relies on the algebraic properties of the equivalent bivariate polynomial formulation of the problem. We then characterize specific algebraic conditions for the binary string to be reconstructed that guarantee the algorithm does not require any backtracking through the reconstruction, and, consequently, the time complexity is bounded polynomially. More specifically, in the case of no backtracking, our algorithm has a time complexity of $O(n^2)$ in practice, compared to the algorithm by Acharya et al., which has a time complexity of $O(n^2 \log n)$, where n is the length of the binary string. Furthermore, it is shown that larger sets of binary strings are uniquely reconstructable by the new algorithm and without the need for backtracking leading to codebooks of *reconstruction codes* that are larger, by a linear factor in size, compared to the previously known construction by Pattabiraman *et al.*, while having $O(n^2)$ practical reconstruction complexity.**

I. Introduction

The previous decade has seen a generation of vast amounts of data [1]. However, traditional digital data storage technologies are approaching their fundamental density limits and would not be able to keep up with the need for increasing memory needs. This has led to a search for storage paradigms that offer storage densities at the nanoscale. Several molecular paradigms with significantly higher storage densities have been proposed recently [2]–[10]. DNA is one such promising data storage medium, but it is prone to a diverse type of errors, and has several scalability constraints including an expensive synthesis and sequencing process. As an alternative, synthetic polymers are emerging as the next-generation data storage medium. They offer high storage density at low cost and low readout latency. In such polymers, monomer units of different masses, which represent the two bits 0 and 1, are assembled into user-determined readable sequences. A common family of technological methods for reading amino-acid sequences (and other biomolecules) is mass spectrometry [11]. In tandem mass spectrometry (MS/MS), the given sample is ionized and randomly broken into substrings. The resulting mixture is analyzed to yield the weights of the substrings generated which are then used to reconstruct the recorded string. This problem of recovering a polymer from its fragmented ions during mass spectrometry is modeled into the problem of reconstructing a string from the multiset of its *substring compositions*. In this paper, we introduce a new algorithm to reconstruct a binary string from the multiset of its substring compositions.

This work was supported by the National Science Foundation under grants CCF–1763348, CCF–1909771, and CCF–1941633.

The problem was first introduced in [12] and [13]. The main results from [13] assert that binary strings of length $\leqslant 7$, one less than a prime, and one less than twice a prime are uniquely reconstructable, from their *substring composition multiset*, up to reversal. The authors of [13] also introduced a backtracking algorithm for reconstructing a binary string from its *substring composition multiset*. Later, the works of [14] and [15] viewed the problem from a coding theoretic perspective and focused on the problem of designing coding schemes of uniquely reconstructable strings capable of correcting a single mass error and multiple mass errors, respectively.

Two modelling assumptions are used in [13] and subsequently in [14], [15]: a) One can uniquely infer the composition (number of monomers of each type) of a polymer from its mass; and b) The masses of all the substrings of a polynomial are observed with identical frequencies. For our line of work, we again rely on these assumptions.

The algorithm we introduce in this paper, unlike the algorithm in [13], which uses the combinatorial properties of the composition multiset to reconstruct the binary strings, takes a new approach by relying on the algebraic properties of the equivalent bivariate polynomial formulation [13] of the problem. We thereby improve the in-practice time complexity of the reconstruction process. However, in general, a drawback of such algorithms is that they need backtracking which can lead to reconstruction complexity that grows exponentially with the length n, in a worst case sense. We then characterize specific algebraic properties that guarantee no backtracking is needed in our proposed algorithm. Although asymptotically, in the case of no backtracking, our algorithm has a reconstruction complexity of $O(n^2 \log n)$ which is the same as the algorithm of [13]; in practice our algorithm has a time complexity of $O(n^2)$, which is further improved as our algorithm naturally allows parallel implementation. Moreover, the *no backtraking condition* of our algorithm is more general than that of [13]. We also improve the time complexity in the case of backtracking. These results are specifically discussed in Remark 7, Remark 8, and Remark 10. Our work extends the growing list of recent work in string reconstruction problems [16]–[22].

In Section IV, we leverage the reconstruction code designs of [14], by expanding codebooks of different sizes in certain specified ways followed by taking a union of them, in order to arrive at a new codebook. The size of the new codebook is shown to be linearly larger than the reconstruction code introduced in [14]. Furthermore, it is shown that both the codes of [14], and the new code are reconstructable by our proposed reconstruction algorithm with no backtracking.

II. Preliminaries

A. Problem Formulation

Let $s = s_1 s_2 \dots s_n$ be a binary string of length $n \geqslant 2$. We will denote the contiguous substring $s_i s_{i+1} \dots s_j$ of s by s_i^j where $1 \leqslant i \leqslant j \leqslant n$. We will say that a substring s_i^j has the composition $1^w 0^z$ where w and z denote the number of 1s and 0s in the substring respectively. The composition multiset $C(s)$ of a sequence s is the multiset of compositions of all contiguous substrings of s. For example, if $s = 1001$, then $C(s) = \{0^1, 0^1, 1^1, 1^1, 0^1 1^1, 0^1 1^1, 0^2, 0^2 1^1, 0^2 1^1, 0^2 1^2\}$.

A set of binary strings of fixed length is called a *reconstruction code* if the composition multisets corresponding to the strings are distinct [14]. Note that a string s, and its reverse string $s^r = s_n s_{n-1} \dots s_1$ share the same composition multiset and therefore cannot simultaneously belong to a reconstruction code. We restrict the analysis of reconstruction codes to the subsets of strings of length n beginning with 1 and ending at 0. This restriction only adds a constant redundancy to the code while ensuring that a string and its reversal are not simultaneously part of the code.

Definition 1. *For a binary string s of length n and weight d, we define a non-negative integer string $A(s) = a_0 a_1 \cdots a_d$ where a_i is the number of zeros between the i^{th} and $(i+1)^{th}$ 1 in s. That is*

$$s = \underbrace{00\cdots0}_{a_0} 1 \underbrace{00\cdots0}_{a_1} 1 \underbrace{00\cdots0}_{a_2} 1 \cdots 1 \underbrace{00\cdots0}_{a_d}.$$

Note that $A(s) \to s$ is a bijection between non-negative integer strings of length $d+1$, weight (sum of values) $n-d$, and binary strings of length n, weight d.

We will also use the following notations in the subsequent sections: for a string s and the corresponding integer string $A(s) = a_0 a_1 \dots a_d$, we use $A_i^j(s)$ to denote the substring $a_i a_{i+1} \dots a_j$ of $A(s)$ and $g_i^j(s)$ to denote the sum $a_i + a_{i+1} \dots + a_j$, where $0 \leqslant i \leqslant j \leqslant d$. Whenever clear from the context, *we omit the argument s*. Observe that for any string s with weight d, $g_0^d = n - d$. For instance, if $s = 10011010$, then $A(s) = 02011$ and $g_1^3 = 3$.

B. Previous Work

In this section, we first review the results of [13] that describe the equivalent polynomial formulation of binary strings and their composition multisets. This formulation is central to the design of our Reconstruction Algorithm which we present in the next section. We use several notations and terminologies from [15] whose authors rely on this polynomial formulation to design reconstruction codes capable of correcting multiple errors. Thereafter, we revisit the design of a reconstruction code introduced in [14].

Definition 2. *For a binary string $s = s_1 s_2 \dots s_n$, a bivariate polynomial $P_s(x,y)$ of degree n is defined such that $P_s(x,y) = \sum_{i=0}^n (P_s(x,y))_i$, where $(P_s(x,y))_0 = 1$ and $(P_s(x,y))_i$ is defined recursively as*

$$(P_s(x,y))_i = \begin{cases} y\,(P_s(x,y))_{i-1} & \text{if } s_i = 0, \\ x\,(P_s(x,y))_{i-1} & \text{if } s_i = 1. \end{cases} \tag{1}$$

$P_s(x,y)$ contains exactly one term of total degree j where $0 \leqslant j \leqslant n$ and the coefficient of each term is 1. The term of the polynomial with degree j is of the form $x^w y^z$ where the substring s_1^j of s has composition $1^w 0^z$. For example, if we consider the string $s = 1001$, then we get $P_s(x,y) = 1 + x + xy + xy^2 + x^2y^2$.

Similar to the bivariate polynomial for a binary string, we describe a bivariate polynomial $S_s(x,y)$ corresponding to the composition multiset $C(s)$ of a binary string s. We associate each element $1^l 0^m$ of the multiset with the monomial $x^l y^m$. This is equivalent to saying that an x corresponds to a 1 and a y corresponds to a 0 in every monomial of $S_s(x,y)$. As an example, for $s = 1001$, $C(s) = \{0^1, 0^1, 1^1, 1^1, 0^1 1^1, 0^1 1^1, 0^2, 0^2 1^1, 0^2 1^1, 0^2 1^2\}$ and $S_s(x,y) = 2x + 2y + 2xy + y^2 + 2x^2y + x^2y^2$.

We will use the following identity from [13]:

$$P_s(x,y)\, P_s\left(\frac{1}{x}, \frac{1}{y}\right) = (n+1) + S_s(x,y) + S_s\left(\frac{1}{x}, \frac{1}{y}\right). \tag{2}$$

Definition 3. *For a polynomial $f(x,y)$, let $f^*(x,y)$ be the polynomial (also known as reciprocal polynomial) defined as:*

$$f^*(x,y) \overset{\text{def}}{=} x^{deg_x(f)} y^{deg_y(f)} f\left(\frac{1}{x}, \frac{1}{y}\right). \tag{3}$$

Definition 4. *For a binary string s of length n, and the corresponding polynomial $P_s(x,y)$, we define a polynomial $F_s(x,y)$ as:*

$$F_s(x,y) \overset{\text{def}}{=} P_s(x,y) P_s^*(x,y). \tag{4}$$

Rewriting equation (2), and using the definition in equation (4), we obtain

$$F_s(x,y) = x^{d_x} y^{d_y} \left(n + 1 + S_s(x,y)\right) + S_s^*(x,y); \tag{5}$$

where d_x and d_y denote the degrees of x and y in the polynomial $P_s(x,y)$.

Remark 1. *This result shows that that the polynomial $F_s(x,y)$ and $S_s(x,y)$ have a unique correspondence between them. Specifically, the polynomial $F_s(x,y)$ uniquely determines the polynomial $S_s(x,y)$ or equivalently, the composition multiset.*

Now we look at the reconstruction code introduced in [14]. The code design uses Catalan-type strings to construct a codebook in which, for a given codeword and any same length prefix-suffix substring pair of this string, the two substrings have different weights.

Definition 5 ([14]). *For reconstruction code $S_R(n)$ of even length (n even):*

$$\begin{aligned} S_R(n) \overset{\text{def}}{=} \{ & s \in \{0,1\}^n, \text{ such that } s_1 = 0,\ s_n = 1, \\ & \exists\, I \in \{2,3,\dots,n-1\}, \text{ such that} \\ & \quad \text{for all } i \in I, s_i \neq s_{n+1-i} \\ & \quad \text{for all } i \notin I, s_i = s_{n+1-i} \\ & \quad s_{[n/2] \cap I} \text{ is a Catalan Type String} \}. \end{aligned}$$

For reconstruction code $S_R(n)$ of odd length (n odd):

$$S_R(n) \overset{\text{def}}{=} \{ s_1^{(n-1)/2} 0 s_{(n+1)/2}^{n-1}, s_1^{(n-1)/2} 1 s_{(n+1)/2}^{n-1}, \text{ where } s \in S_R(n-1) \}.$$

Using this codebook $S_R(n)$, in Section IV, we design a new reconstruction code which is uniquely reconstructable by the Reconstruction Algorithm. We will use the following property of $S_R(n)$ in our construction and to show that the reconstruction code we introduce is linearly larger than $S_R(n)$.

Theorem 1 ([14], Lemma 1)**.** *For a string $s \in S_R(n)$, for all prefix-suffix pairs of length $1 \leqslant j \leqslant n-1$, one has $wt(s_1^j) \neq wt(s_{n+1-j}^n)$.*

III. Reconstruction Algorithm

As discussed in Section II-A, we only work with binary strings beginning with 1 and ending with 0. In other words, only strings $s = s_1 \ldots s_n$ with $s_1 = 1, s_n = 0$ are considered. In this section, we introduce a new reconstruction algorithm to recover such strings from a given composition multiset. Given a composition multiset, our reconstruction algorithm successively reconstructs $A(s) = a_0 \ldots a_d$, starting from both ends and then progressing towards the center, i.e., a_0 and a_d are covered first, followed by a_1 and a_{d-1}, etc.; and then backtracks when there is an error. The algorithm takes as input the polynomial $F(x, y)$ which can be derived from the given composition multiset (Remark 1). The algorithm will return the set of strings which have the given composition multiset. We will use the fact that for a string s with the given composition multiset, we must have $F_s(x, y) = F(x, y)$. Remark 1 guarantees that the set of strings recovered in this way indeed have the desired composition multiset.

Before the algorithm is discussed, we first show how certain parameters of a string s with the given composition multiset can be readily recovered from the polynomial $F(x, y)$. These parameters will be subsequently used as inputs to the algorithm.

For a string $s = s_1 \ldots s_n$ with $s_1 = 1, s_n = 0$, the corresponding non-negative integer string $A(s)$ (Definition 1) is such that $a_0 = 0$ and $a_d \geqslant 1$. Using Definitions 2 and 3,

$$P_s(x, 1) = 1 + (a_1 + 1)\, x + \cdots + (a_d + 1)\, x^d, \tag{6}$$

$$P_s^*(x, 1) = (a_d + 1) + (a_{d-1} + 1)\, x + \cdots + x^d. \tag{7}$$

Since a string s with the given composition multiset must have $F_s(x, y) = F(x, y)$, $F(x, 1) = P_s(x, 1) P_s^*(x, 1)$. Therefore, using equations (6) and (7), the weight of the string s and a_d (where $A(s) = a_0 \ldots a_d$) can be recovered as follows:

$$wt(s) = d = \frac{\deg F(x, 1)}{2}, \tag{8}$$

$$F(0, 1) = a_d + 1. \tag{9}$$

The algorithm will utilize the polynomial formulation of the problem by mapping them to elements of a polynomial ring by considering the coefficients as elements of a sufficiently large finite field, i.e., $\mathbb{F}_q$ with q being a prime number greater than n. Let $\lambda \in \mathbb{F}_q$ be a fixed primitive element of this field.

Definition 6. *Given $a_1, \ldots, a_j$ and $a_d, \ldots, a_{d-j}$ in $\mathbb{N} \cup \{0\}$; define the polynomials $\alpha_j(y)$ and $\beta_j(y)$ as follows:*

$$\alpha_j(y) = y^{g_0^{j-1}} + y^{1+g_0^{j-1}} + \cdots + y^{g_0^j}, \tag{10}$$

$$\beta_j(y) = y^{g_{d-j+1}^d} + y^{1+g_{d-j+1}^d} + \cdots + y^{g_{d-j}^d}; \tag{11}$$

where g_k^l denotes the sum $a_k + a_{k+1} \ldots + a_l$ (defined in Section II-A).

Then, using Definitions 2 and 3, $\alpha_j(\lambda)$ and $\beta_j(\lambda)$ denote the coefficient of x^j in $P_s(x, \lambda)$ and $P_s^*(x, \lambda)$, respectively. In particular,

$$\alpha_0(\lambda) = 1, \text{ and } \alpha_d(\lambda) = \lambda^{n-d-a_d} + \cdots + \lambda^{n-d}; \tag{12}$$

$$\beta_d(\lambda) = \lambda^{n-d}, \text{ and } \beta_0(\lambda) = 1 + \cdots + \lambda^{a_d}. \tag{13}$$

Remark 2. *$\alpha_j(\gamma)$ and $\beta_j(\gamma)$ correspond to the coefficients of x^j in $P_s(x, \gamma)$ and $P_s^*(x, \gamma)$, respectively, for all $\gamma \in \mathbb{F}_q$. For instance, putting $\gamma = 1$ gives $\alpha_j(1) = a_j + 1$ and $\beta_j(1) = a_{d-j} + 1$ which are the coefficients of x^j in the polynomials $P_s(x, 1)$ (equation (6)) and $P_s^*(x, 1)$ (equation (7)), respectively.*

The reconstruction algorithm will find a_j and a_{d-j} together at step j. Note that $\alpha_i(y)$ is defined using g_0^i and g_0^{i-1}, and therefore, can be obtained by knowing the elements $a_1, \ldots, a_i$. Similarly, $\beta_i(y)$ can be obtained from $a_d, \ldots, a_{d-i}$. Hence, for a string s, by the end of step $j-1$ of the algorithm, the polynomials $\alpha_0(y), \ldots, \alpha_{j-1}(y)$ and $\beta_0(y), \ldots, \beta_{j-1}(y)$ are well defined.

Definition 7. *Let $r_j(y)$ denote the coefficient of x^j in $F(x, y)$. Then $r_j(y)$ can be treated as a polynomial in y. At the end of step $j-1$, for polynomials $\alpha_0(y), \ldots, \alpha_{j-1}(y)$ and $\beta_0(y), \ldots, \beta_{j-1}(y)$, define the polynomial $f_j(y)$ as follows:*

$$f_j(y) \overset{\text{def}}{=} r_j(y) - \sum_{k=1}^{j-1} \alpha_k(y) \beta_{j-k}(y). \tag{14}$$

At step j, the algorithm finds a_j and a_{d-j}. If $a_1, \ldots, a_{j-1}$ and $a_d, \ldots, a_{d-j+1}$ are identified correctly, then for the correct pair (a_j, a_{d-j}), the coefficient of x^j in $F_s(x, y) \in \mathbb{F}_q[x]$ is $\sum_{i=0}^j \alpha_i(y) \beta_{j-i}(y)$. Since $F_s(x, y) = F(x, y)$, we must have $\sum_{i=0}^j \alpha_i(y) \beta_{j-i}(y) = r_j(y)$. Since we already know $\alpha_0(y), \ldots, \alpha_{j-1}(y)$ and $\beta_0(y), \ldots, \beta_{j-1}(y)$ by step $j-1$; a correct (a_j, a_{d-j}) must satisfy

$$f_j(y) = \alpha_0(y) \beta_j(y) + \alpha_j(y) \beta_0(y). \tag{15}$$

By noting that the degrees of both sides should be equal, we have

$$\begin{aligned} deg(f_j) &= \max\{deg(\alpha_0 \beta_j), deg(\alpha_j \beta_0)\} \\ &= \max\{g_{d-j}^d, g_0^j + a_d\}. \end{aligned} \tag{16}$$

Furthermore, observe that $\alpha_i(1) = \beta_{d-i}(1) = 1 + a_i$ and hence, $f_j(1) = \beta_j(1) + (a_d + 1)\, \alpha_j(1)$. From this we obtain:

$$f_j(1) = (1 + a_{d-j}) + (a_d + 1)\, (a_j + 1)\,. \tag{17}$$

Since any correct pair (a_j, a_{d-j}) must satisfy equations (16) and (17), we will use these to compute the possible values for the pairs (a_j, a_{d-j}).

Remark 3. *Note that equations (16) and (17) can have at most two solutions and thus the pair (a_j, a_{d-j}) at step j can take at most two possible values.*

If $a_1, \ldots, a_{j-1}$ and $a_d, \ldots, a_{d-j+1}$ are identified correctly, then the polynomials $\alpha_j(y)$ and $\beta_j(y)$ obtained from a correct pair (a_j, a_{d-j}) must satisfy equation (15). In particular, in Proposition 2, we show that verifying the equation (15) for

$y = \lambda$ and $y = \lambda^{-1}$, where λ is a primitive root of $\mathbb{F}_q$, is enough to say that equation (15) holds for all y.

Note that the algorithm gives us the set of strings with $F_s(x, y) = F(x, y)$. And Remark 1 shows that such strings indeed share the same composition multiset.

Algorithm 1 Reconstruction Algorithm

Input : Polynomial $F(x, y)$, array A of size d initialized with $A[d] = a_d$ and $A[i] = 0$ for all $0 \leqslant i \leqslant d-1$,
Output: Codestrings $s \in \{0,1\}^n$

```
Function Reconstruction(j, F, M):
    if $j = d/2$ then
        if M corresponds to some binary string s then
            S = s
        else
            S = empty set
        return $S$
    Compute $deg(f_j)$ and $f_j(1)$
    $a_j = deg(f_j) - (g_0^{j-1} + a_d)$
    $a_{d-j} = f_j(1) - 1 - (a_d + 1)(a_j + 1)$
    if $a_j \geqslant 0$, $a_{d-j} \geqslant 0$, $\alpha_0(\lambda)\beta_j(\lambda) + \alpha_j(\lambda)\beta_0(\lambda) = f_j(\lambda)$,
      and $\alpha_0(\lambda^{-1})\beta_j(\lambda^{-1}) + \alpha_j(\lambda^{-1})\beta_0(\lambda^{-1}) = f_j(\lambda^{-1})$ then
        $M[j] = a_j, M[d-j] = a_{d-j}$
        $S$ = Reconstruction($j+1$, F, M)
    $a_{d-j} = deg(f_j) - g_{d-j+1}^d$
    $a_j = (f_j(1) - 1 - a_{d-j})/(a_d + 1)$
    if $a_j \in \mathbb{N} \cup \{0\}$, $a_{d-j} \geqslant 0$, $\alpha_0(\lambda)\beta_j(\lambda) + \alpha_j(\lambda)\beta_0(\lambda) =$
      $f_j(\lambda)$, and $\alpha_0(\lambda^{-1})\beta_j(\lambda^{-1}) + \alpha_j(\lambda^{-1})\beta_0(\lambda^{-1}) =$
      $f_j(\lambda^{-1})$ then
        $M[j] = a_j, M[d-j] = a_{d-j}$
        $S = S \cup$ Reconstruction($j+1$, F, M)
    return $S$
```

Remark 4. *From equations* (10), *and* (11), *we see that* $\alpha_k(\lambda)$ *and* $\beta_l(\lambda)$ *are of the form* $\frac{\lambda^a(\lambda^b-1)}{\lambda-1} = \lambda^{a+b'}$, *where* $\frac{\lambda^b-1}{\lambda-1} = \lambda^{b'}$. *Assuming addition happens in* $O(\log n)$ *time, pre-storing* b' *corresponding to* b; $\alpha_k(\lambda)\beta_{j-k}(\lambda)$ *can be evaluated in* $O(\log n)$ *as a power of* λ *and consequently,* $\sum_{k=1}^{j-1}\alpha_k(\lambda)\beta_{j-k}(\lambda)$ *can be calculated in* $O(j\log n)$ *time and* $O(n)$ *space. If the coefficient* $a_{k,l}$ *of* $x^k y^l$ *of the polynomial* $F(x, y)$ *are stored in a matrix, then* $a_{j,l}\lambda^l$ *can be calculated in* $O(\log n)$ *time, and the* n *row values can be summed in* $O(n\log n)$ *time. Thus* $f_j(\lambda)$ *can be calculated in* $O(n\log n)$ *time.*

Remark 5. *For practically relevant values of* n, *addition can be considered an* $O(1)$ *process. For example, on a* 32*-bit system, two* 32 *bit numbers can be added in one cycle, and therefore for* $\log n < 32$, *addition can be assumed to be an* $O(1)$ *process, and therefore for practical values of* n, *calculating* $f_j(\lambda)$ *is an* $O(n)$ *process.*

Remark 6. *Since the degree and the coefficients of the polynomial* $f_j(y)$ *(Definition 7) are always non-negative integers less than* n, $deg(f_j) = \lfloor log_{n+1}(f_j(n+1)) \rfloor$.

Remark 7. *Backtracking is possible even when the string is uniquely reconstructable. Assuming no back-tracking, time complexity of the algorithm is* $O(dn\log n) = O(n^2\log n)$. *This is same as the time complexity of the backtracking algorithm proposed by Acharya et. al. in [13]. For practical values of* n, *our algorithm shows an improved complexity of* $O(dn) = O(n^2)$. *Furthermore, in the case of no backtracking, the reconstruction algorithm can be implemented over* $O(n\log n)$ *latency by executing additions in parallel while calculating* $f_j(\lambda)$ *etc.*

From Remark 3, we know that the reconstruction algorithm has at most two valid choices for the pair (a_j, a_{d-j}) at step j, and therefore can have at most two branches at any step. If both the conditions are satisfied i.e. the algorithm branches; then our algorithm must choose one direction to proceed. If the algorithm has no valid choices at a particular step, the algorithm comes back to the last branch (not taken yet) where both conditions were satisfied and takes the alternate path. In this case, if neither of the two conditions are satisfied, then assuming the input composition multiset doesn't have any errors, our algorithm must have taken some wrong branch in the past (when it had a choice). Therefore, for neither of the two conditions to be satisfied, the algorithm must have branched somewhere in the past, i.e. it there must have been some step j in the past where both conditions were satisfied.

We say that a string s *pauses* at step j if the algorithm has two valid choices for the pair (a_j, a_{d-j}) of the string $A(s)$ at step j. As explained above, if a string at some step has no valid choices for the pair (a_j, a_{d-j}), it must have paused at some step $i < j$. Therefore, characterizing algebraic conditions for strings to avoid *pausing* at any step will guarantee that the algorithm doesn't need to backtrack. This motivates the design of our Reconstruction Code (Definition 12). Furthermore, avoiding backtracking ensures that the string is uniquely reconstructed from its composition multiset. Proposition 2 gives algebraic conditions which characterize the strings that branch at some given step j. The detailed proofs can be found in the extended version of our paper [23].

Proposition 2. *Let the bi-variate polynomial corresponding to a string* s *be* $F_s(x, y)$. *Then the reconstruction algorithm pauses at step* j *if and only if the string* s *satisfies either of the following two relations:*

$$g_0^j - g_{d-j}^d = a_0 + 1 = 1 \quad \text{and } a_j \geqslant 1 \tag{18}$$
$$g_{d-j}^d - g_0^j = a_d + 1 \quad \text{and } a_{d-j} \geqslant a_d + 1 \tag{19}$$

Moreover, when the reconstruction algorithm pauses at step j, *it has exactly two choices for the tuple* (a_j, a_{d-j}).

Proof sketch: The algorithm pauses at step j, if there exist two pairs of 2-tuples, say (a_j, a_{d-j}) and (a'_j, a'_{d-j}) such that both of them satisfy equation (15) for $y = \lambda$ and $y = \lambda^{-1}$. Using equations (16), (17), and the corresponding two expressions for the polynomial evaluations $f_j(\lambda)$ and $f_j(\lambda^{-1})$; we get four equations in four variables which give the required result. ■

Corollary 3. *If a binary string* s *of length* n, *which begins with*

1 *and ends with* 0*, is such that for all prefix-suffix pairs of length* $1 \leqslant j \leqslant n$*, one has* $wt(s_1^j) \neq wt(s_{n+1-j}^n)$*, then* s *is uniquely reconstructable by this reconstruction algorithm.*

Remark 8. *Corollary 3 implies that our algorithm uniquely reconstructs the codewords of the reconstruction code described in [14] (revisited in Section II-B) without backtracking. In Remark 7 we showed the reconstruction algorithm presented in this paper has a worst-case time complexity of* $O(n^2)$ *when there is no backtracking compared to the reconstruction algorithm in [13] which has a time complexity of* $O(n^2 \log n)$*.*

Definition 8. *We will call the strings which satisfy condition* (18) *for some* $0 < j < d/2$ *as type-1 strings, and the strings which satisfy condition* (19) *for some* $0 < j < d/2$ *as type-2 strings.*

Remark 9. *A string can be a type-1 string, a type-2 string, both a type-1 and a type-2 string, or be of neither type.*

Remark 10. *If we define* l_s *as defined in [13], that is*

$$l_s \overset{\text{def}}{=} \left|\{i < n/2 : w(s_1^i) = w(s_{n+1-i}) \text{ and } s_{i+1} \neq s_{n-i}\}\right|;$$

by proof of Corollary 3, each time the string s *pauses at some step* j*, we have* $j \in I$*. Furthermore, for every* $j \in I$*, the string is a type-1 string. Therefore the number of branches in case of backtracking in our algorithm is less than or equal to the number of branches of the backtracking algorithm in [13]. Thus our algorithm is able to find* s *before depth* $l_s + 1$ *and therefore the time complexity of our algorithm is* $O(2^{l_s} n^2)$ *compared to the algorithm in [13] whose time complexity is* $O(2^{l_s} n^2 \log n)$*.*

IV. Reconstruction Code

In this section, we describe a new reconstruction code $T(n)$ such that the codewords are uniquely reconstructable by our Reconstruction Algorithm without backtracking. The codebook design relies on the property (Theorem 1) of the reconstruction code $S_R(n)$ introduced in [14], which showed that all same length prefix-suffix substring pairs have different weights. We review other relevant properties of this code design in Section II-B. We exploit the more general condition for reconstruction without backtracking of our algorithm (Proposition 2) and construct new codewords that may violate Theorem 1, that is have some prefix-suffix substring pairs of the same weight, and yet can be uniquely reconstructed by our algorithm without backtracking. We define the following three kinds of sets whose construction uses this $S_R(n)$. The reconstruction code $T(n)$ will be defined as the union of these sets. Note that, any codebook with all codewords beginning at 1 and ending with 0, and satisfying Theorem 1 can be extended in a similar way.

Definition 9. *The set* P_n *is defined as a set of binary strings of length* n *which begin at* 1*, end at* 0*, and is such that the reverse of a string lying in this set lies in* $S_R(n)$ *(Definition 5):*

$$P_n = \{s \in \{0,1\}^n, \text{ such that } s^R \in S_R(n)\}. \tag{20}$$

Definition 10. *The set* Q_n *is defined as a set of binary strings of length* n *which both begin and end with the substring* 1^k0*, for some* $k < n/2$*, and has the substring* $s_{k+1}^{n-k} \in S_R(n-2k)$*:*

$$Q_n = \bigcup_{k=1}^{(n-2)/2} \{s \in \{0,1\}^n, \text{ such that } s_1^{k+1} = 1^k0, s_{n-k}^n = 1^k0, \text{ and } s_{k+1}^{n-k} \in S_R(n-2k)\}. \tag{21}$$

Definition 11. *The set* R_n *is defined as a set of binary strings of length* n *which begin with the substring* 1^k0*, and end with the substring* 1^k00 *for some* $k < (n-1)/2$*, and has the substring* $s_{k+1}^{n-k-1} \in S_R(n-2k-1)$*:*

$$R_n = \bigcup_{k=1}^{(n-3)/2} \{s \in \{0,1\}^n, \text{ such that } s_1^{k+1} = 1^k0, s_{n-k-1}^n = 1^k00, \text{ and } s_{k+1}^{n-k-1} \in S_R(n-2k-1)\}. \tag{22}$$

Remark 11. *Looking at the* $(n-1)^{th}$ *bit, note that* $Q_n \cap R_n = \phi$*. Moreover, by definition of* Q_n*,* $P_n \cap Q_n = \phi$*.*

Definition 12. *The reconstruction code* $T(n)$ *is defined as:*

$$T(n) = P_n \cup Q_n \cup R_n. \tag{23}$$

We already know that strings of P_n are reconstructable without backtracking. In the following lemma, we show that Q_n and R_n have no *type-2 strings*. Therefore, if the algorithm *pauses* at some step j, we know that the string is a *type-1 string* and therefore must satisfy condition (18). Thus we can uniquely determine (a_j, a_{d-j}) and don't need to backtrack.

Proposition 4. *If a string* $s \in T(n)$ *such that* s *pauses at step* j *for some* $0 < j < d/2$*, then* s *is a type-1 string.*

Proof: We will show that $s \in Q_n$ cannot be a *type-2 string*. The proof for R_n is similar. Observe that $F(1,0) = k+1$, where $s_1^{k+1} = 1^k0$; and hence we can retrieve k and therefore s_1^k. Note that $g_0^j = 0$ and $g_{d-j}^d = 1$ for $1 \leqslant j < k$. Using Proposition 2, the algorithm does not pause at step any j for $1 \leqslant j < k$.

If possible, let $s \in Q_n$ be a *type-2 string* that *pauses* at some step $j \geqslant k$. Then the string satisfies the Condition 19. Using $a_{d-j} \geqslant a_d + 1 \geqslant 2$,

$$wt\left(s_{k+1}^{1+j+g_0^j}\right) = j + 1 - k = wt\left(s_{n+1+a_d-j-g_{d-j}^d}^{n-k+1}\right)$$

contradicting $s_{k+1}^{n-k} \in S_R(n-2k)$. ∎

Theorem 5. *Given* $n > 6$*, there exists a constant* $c = 1.025$ *such that* $|T(n)| \geqslant c|S_R(n)|$*.*

Proof sketch: Using Remark 11, we note that $|T(n)| \geqslant |P_n| + |Q_n| \geqslant |S_R(n)| + |S_R(n-2)|$. Using bounds on the size of $S_R(n)$ and well known inequalities for the central binomial coefficient, we show that $\frac{|S_R(n-2)|}{|S_R(n)|} \geqslant \frac{1}{40}$. ∎

References

[1] D. R.-J. G.-J. Rydning, "The digitization of the world from edge to core," *Framingham: International Data Corporation*, p. 16, 2018.

[2] C. C. A. Ng, W. M. Tam, H. Yin, Q. Wu, P.-K. So, M. Y.-M. Wong, F. Lau, and Z.-P. Yao, "Data storage using peptide sequences," *Nature Communications*, vol. 12, no. 1, pp. 1–10, 2021.

[3] K. Launay, J.-A. Amalian, E. Laurent, L. Oswald, A. Al Ouahabi, A. Burel, F. Dufour, C. Carapito, J.-L. Clément, J.-F. Lutz *et al.*, "Precise alkoxyamine design to enable automated tandem mass spectrometry sequencing of digital poly (phosphodiester) s," *Angewandte Chemie*, vol. 133, no. 2, pp. 930–939, 2021.

[4] K. Matange, J. M. Tuck, and A. J. Keung, "DNA stability: a central design consideration for DNA data storage systems," *Nature communications*, vol. 12, no. 1, pp. 1–9, 2021.

[5] M. G. Rutten, F. W. Vaandrager, J. A. Elemans, and R. J. Nolte, "Encoding information into polymers," *Nature Reviews Chemistry*, vol. 2, no. 11, pp. 365–381, 2018.

[6] A. Al Ouahabi, J.-A. Amalian, L. Charles, and J.-F. Lutz, "Mass spectrometry sequencing of long digital polymers facilitated by programmed inter-byte fragmentation," *Nature communications*, vol. 8, no. 1, pp. 1–8, 2017.

[7] V. Zhirnov, R. M. Zadegan, G. S. Sandhu, G. M. Church, and W. L. Hughes, "Nucleic acid memory," *Nature materials*, vol. 15, no. 4, pp. 366–370, 2016.

[8] R. N. Grass, R. Heckel, M. Puddu, D. Paunescu, and W. J. Stark, "Robust chemical preservation of digital information on DNA in silica with error-correcting codes," *Angewandte Chemie International Edition*, vol. 54, no. 8, pp. 2552–2555, 2015.

[9] S. H. T. Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic, "A rewritable, random-access DNA-based storage system," *Scientific reports*, vol. 5, no. 1, pp. 1–10, 2015.

[10] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized DNA," *Nature*, vol. 494, no. 7435, pp. 77–80, 2013.

[11] T. E. Creighton, *Proteins: structures and molecular properties*. Macmillan, 1993.

[12] J. Acharya, H. Das, O. Milenkovic, A. Orlitsky, and S. Pan, "On reconstructing a string from its substring compositions," in *2010 IEEE International Symposium on Information Theory*, 2010, pp. 1238–1242.

[13] ——, "String reconstruction from substring compositions," *SIAM Journal on Discrete Mathematics*, vol. 29, no. 3, pp. 1340–1371, 2015.

[14] S. Pattabiraman, R. Gabrys, and O. Milenkovic, "Reconstruction and error-correction codes for polymer-based data storage," in *2019 IEEE Information Theory Workshop (ITW)*. IEEE, 2019, pp. 1–5.

[15] R. Gabrys, S. Pattabiraman, and O. Milenkovic, "Mass error-correction codes for polymer-based data storage," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 25–30.

[16] S. Marcovich and E. Yaakobi, "Reconstruction of strings from their substrings spectrum," *IEEE Transactions on Information Theory*, 2021.

[17] R. Gabrys, S. Pattabiraman, and O. Milenkovic, "Reconstructing mixtures of coded strings from prefix and suffix compositions," in *2020 IEEE Information Theory Workshop (ITW)*. IEEE, 2021, pp. 1–5.

[18] M. Cheraghchi, R. Gabrys, O. Milenkovic, and J. Ribeiro, "Coded trace reconstruction," *IEEE Transactions on Information Theory*, vol. 66, no. 10, pp. 6084–6103, 2020.

[19] M. Abroshan, R. Venkataramanan, L. Dolecek, and A. G. i Fabregas, "Coding for deletion channels with multiple traces," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 1372–1376.

[20] R. Gabrys and O. Milenkovic, "Unique reconstruction of coded sequences from multiset substring spectra," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 2540–2544.

[21] H. M. Kiah, G. J. Puleo, and O. Milenkovic, "Codes for dna sequence profiles," *IEEE Transactions on Information Theory*, vol. 62, no. 6, pp. 3125–3146, 2016.

[22] A. S. Motahari, G. Bresler, and N. David, "Information theory of dna shotgun sequencing," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6273–6289, 2013.

[23] U. Gupta and H. Mahdavifar, "A new algebraic approach for string reconstruction from substring compositions," *arXiv preprint arXiv:2201.09955*, 2022.