A New Algebraic Approach for String Reconstruction from Substring Compositions

Utkarsh Gupta, *Student Member, IEEE*, and Hessam Mahdavifar, *Member, IEEE*

Abstract*—In this paper, we propose a new algorithm for the problem of string reconstruction from its substring composition multiset. Motivated by applications in polymer-based data storage for recovering strings from tandem mass-spectrometry sequencing, the proposed algorithm leverages the equivalent polynomial formulation of the problem which facilitates efficient parallel implementation. The computational complexity of the proposed reconstruction algorithm is upper bounded by $6.5n^2$ finite field operations, where the field size is upper bounded by $10n$, implying that the computational complexity is upper bounded by $6.5n^2(3.22+\log n)$ binary operations. Furthermore, it allows parallelization leading to $O(n\log n)$ reconstruction latency. We characterize sufficient conditions for a length n binary string that guarantee the string's reconstruction time complexity to be bounded polynomially. Moreover, the sufficient conditions on binary strings that guarantee reconstruction in polynomial time are more general than the conditions for the algorithm by Acharya *et al*. This is used to construct new codebooks of *reconstruction codes* that have efficient encoding procedures, and are larger, by at least a linear factor in size, compared to the previously best known construction by Pattabiraman *et al.

I. Introduction

Recent years have seen an explosion in the amount of data created globally [1]. The volume of data generated, consumed, copied, and stored is projected to reach more than 180 zettabytes by 2025. In 2020, the total amount of data generated and consumed was 64.2 zettabytes [2]. However, traditional digital data storage technologies such as SSDs, hard drives, and magnetic tapes are approaching their fundamental density limits and would not be able to keep up with the increasing memory needs [3]. Several molecular paradigms with significantly higher storage densities have been proposed recently [4]–[15]. Molecules with a structure consisting of different smaller molecules (monomers) joined together in sequences are called polymers. If different types of molecules denote different letters from an alphabet, then a polymer with a linear arrangement of these molecules, i.e., a polymer string, can be treated as a sequence of letters. DNA is one promising data storage medium which has stimulated significant interest in the data storage research community. However, DNA has several scalability constraints including an expensive synthesis and sequencing process which prevent large-scale commercialization. Furthermore, DNA is prone to diverse types of errors such as mutations within strands, or loss of strands due to breakage or degradation that could lead to potential decoding errors or even complete loss of information [8].

This has led researchers to search for alternatives in other synthetic polymers. For example, synthetic proteins (which are polymers of amino acids) are emerging as a potential alternative with data being stored using peptide sequences for the first time in 2021 [4]. Compared to DNA and other types of polymers, proteins offer several advantages for data storage, including higher stability of some proteins than DNA [16], and availability of a larger set of possible monomers (20 amino acids are observed in natural proteins). In synthetic polymer strings, 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 bio-polymers) is mass spectrometry [17]. Mass spectrometers take a large number of identical polymer strings, randomly break the polymer into substrings, and analyze the resulting mixture. The mass sequencing spectrum obtained gives us the mass/charge ratio and the abundance of different ions when the polymer is broken. This information is then modeled into the mass and frequency of each contiguous molecular substring. The process of recovering a molecular string from its mass sequencing spectrum is modeled into the problem of reconstructing a string from the multiset of the compositions of its contiguous substrings.

The class of problems of reconstructing a string from substring information falls under the general framework of string reconstruction problems. Due to their relevance in designing codebooks for molecular storage frameworks, the list of recent work in string reconstruction has grown rapidly [18]–[27]. In particular, a *composition multiset* of a binary string refers to the multiset of tuples of number of 0s and 1s in each contiguous substring of the given string. The problem of string reconstruction from its substring compositions was first introduced in [28] and [26]. The main results from [26] 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 [26] also introduced a backtracking algorithm for reconstructing a binary string from its *substring composition multiset*, and provide sufficient conditions for reconstructability of a binary string using the proposed algorithm in [26] without the need for backtracking (Lemma 5). In the case of no backtracking, this algorithm has a time complexity of $O(n^2\log n)$. Also, given the nature of the algorithm in [26], parallelization is not possible and, hence, the latency is also $O(n^2\log n)$. Note that in the case of backtracking, there is no guarantee that the time complexity will remain bounded polynomially with n. Relying

This paper was presented in part at the 2022 IEEE International Symposium on Information Theory [DOI: 10.1109/ISIT50566.2022.9834531]. This work was supported in part by the National Science Foundation under grant CCF–2415440, and by the Center for Ubiquitous Connectivity (CUbiC), sponsored by Semiconductor Research Corporation (SRC) and Defense Advanced Research Projects Agency (DARPA) under the JUMP 2.0 program.

U. Gupta and H. Mahdavifar are with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115 (email: gupta.utk@northeastern.edu and h.mahdavifar@northeastern.edu).

on this reconstruction algorithm, the works of [29], [30] and [31] viewed the problem from a coding theoretic perspective. They proposed coding schemes that are capable of correcting a single mass error and multiple mass errors, respectively, and can be reconstructed by the reconstruction algorithm without backtracking. The problem formulation in [26], and subsequently in [29], relies on the two following assumptions: 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 polymer are observed with identical frequencies. In this work, we also rely on these assumptions. In the context of combinatorics, the problem is closely related to the turnpike problem, also known as the partial digest problem, where the locations of n highway exits need to be recovered from the multiset of their $\binom{n}{2}$ interexit distances. In [26], the authors showed that the problem of string reconstruction from its *composition multiset* can be reduced to an instance of the turnpike problem.

In this paper, we propose a new algorithm to reconstruct the set of binary strings with a given multiset of substring compositions. The proposed algorithm relies on on the algebraic properties of the equivalent bivariate polynomial formulation [26] of the problem. The algorithm finds the coefficients of the corresponding polynomial in a manner that reconstructs the binary string from both ends progressing towards the center. However, in general, a drawback of such algorithms is that they may need backtracking which can lead to reconstruction complexity that grows exponentially with the length n, in a worst case scenario. Therefore, we provide algebraic conditions on binary strings that are sufficient to guarantee unique reconstruction by the proposed algorithm without backtracking, that requires at most $6.5n^2$ finite field operations. The size of the underlying finite field is upper bounded by $10n$. As a result, the reconstruction computational complexity is upper bounded by $6.5n^2(3.22 + \log n)$ binary operations. Moreover, the algorithm naturally allows parallel implementation and has an $O(n \log n)$ reconstruction latency. Latency, in this context, is defined as the total elapsed time from the start of the algorithm's execution to the completion of its output, taking into account the parallelization of computational tasks. Furthermore, the *no backtracking condition* of our algorithm is more general than that of the algorithm in [26]. This in particular implies that the *reconstruction code* introduced in [29] is reconstructable by our reconstruction algorithm without backtracking. In Section IV, properties of one-dimensional random walks are leveraged to explicitly characterize the set of binary strings that can be reconstructed by the algorithm in [26] without backtracking. In particular, we define this *reconstruction code* to be $S(n)$ and show a bijection between $S(n)$ and 1-dimensional *positive* n-step walks starting from the origin. Using this bijection we propose efficient encoding and decoding procedures for $S(n)$, and show an equivalence between $S(n)$ and the reconstruction code $S_R(n)$ introduced in [29]. We further extend this codebook to propose a new reconstruction code $T(n)$ by expanding codebooks of different sizes in certain specified ways followed by taking a union of them. The size of $T(n)$ is shown to be linearly larger than $S(n)$, and equivalently $S_R(n)$. Furthermore, it is shown that both, the codebook $S(n)$ (and equivalently $S_R(n)$), and the codebook $T(n)$ are reconstructable by the proposed reconstruction algorithm with the exact number of required operations characterized. Finally, exploiting the more general sufficient conditions, we slightly modify the proposed algorithm, to introduce larger codebooks $T_1(n)$, and $T_2(n)$. The inclusion relation between these different codebooks is presented in Figure 1. A comparison of the rates and redundancies of the different coding schemes is presented in Figure 3 and 4. The rest of this paper is organized as follows. We describe the problem setting, preliminaries, and relevant previous work in Section II. Then, we describe the new reconstruction algorithm in Section III. In Section IV, we present the new reconstruction code. Finally, we discuss concluding remarks and future research directions in Section V.

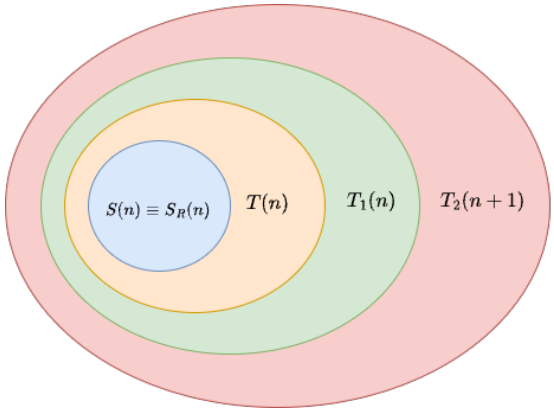

Fig. 1: Inclusion relation between different proposed codes and the the previous known code $S_R(n)$.

II. Preliminaries

In this section, we begin by introducing some notations and definitions, and then formally describe the problem of string reconstruction from substring compositions. Subsequently, we discuss the equivalent polynomial characterization of the problem, introduced by Acharya et al. [26]. The relevant results of [26] and [29] are then summarized with certain observations which will be used in the Reconstruction Algorithm. We then recall certain well known results from the theory of random walks which we use to design the proposed Reconstruction Code.

A. Problem Formulation

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

Example 1. *If* $\mathbf{s} = 1001$*, then* $C(\mathbf{s}) = \{0^1, 0^1, 1^1, 1^1, 0^1 1^1, 0^1 1^1, 0^2, 0^2 1^1, 0^2 1^1, 0^2 1^2\}$.

Definition 1. *For a binary string* $\mathbf{s}$ *of length* n *and weight* d*, let* a_i *be the number of zeros between the* i^{th} *and* $(i+1)^{th}$ 1*. Define* $A(\mathbf{s})$ *as the integer string* $a_0 a_1 \dots a_d$*.*

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

Proof: Consider the mapping that maps the non-negative integer strings of length $d+1$ and weight $n-d$ to binary strings of length n by constructing the corresponding binary string from an $A(\mathbf{s})$ as evident in Definition 1.That is

$$\mathbf{s} = \underbrace{00\dots0}_{a_0}1\,\underbrace{00\dots0}_{a_1}1\,\underbrace{00\dots0}_{a_2}1\dots1\,\underbrace{00\dots0}_{a_d}.$$

Now consider two such distinct non-negative integer strings $a = a_0 \dots a_d$ and $b = b_0 \dots b_d$. If the first position they differ in is i, that is $a_i \neq b_i$ and $a_j = b_j$ for $0 \leqslant j \leqslant i-1$, then the corresponding binary strings differ in the positions of their i^{th} 1s. Therefore, each such non-negative integer string corresponds to a unique binary string; implying that the mapping is injective. It is easy to see that both sets have the same size $\binom{n}{d}$, therefore implying the bijection. ∎

We will also use the following notations in our subsequent proofs: for a string $\mathbf{s}$ and the corresponding integer string $A(\mathbf{s}) = a_0 a_1 \dots a_d$, we use $A_i^j(\mathbf{s})$ to denote the substring $a_i a_{i+1} \dots a_j$ of $A(\mathbf{s})$ and $g_i^j(\mathbf{s})$ 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* $\mathbf{s}$. Observe that for any string $\mathbf{s}$ with weight d, $g_0^d = n - d$.

Example 2. *If* $\mathbf{s} = 10011010$*, then* $A(\mathbf{s}) = 02011$ *and* $g_1^3 = 3$*.*

Definition 2 ([29])**.** *A set of binary strings of fixed length is called a reconstruction code if the composition multisets corresponding to the strings are distinct.*

Note that a string $\mathbf{s}$, and its reverse string ($\mathbf{s}^* = s_n \dots s_1$) share the same composition multiset and therefore cannot simultaneously belong to a reconstruction code.

Remark 1. *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.*

In this paper, the following two problems are addressed (1) Does there exist an efficient algorithm to reconstruct a binary string given its composition multiset?, and (2) Do there exist reconstruction codes of small redundancy and consequently, large rate that can be efficiently encoded and decoded, and can be reconstructed from their composition multiset efficiently? In Section III, we propose a new backtracking algorithm that reconstructs a string $\mathbf{s}$ by recovering the integer string $A(\mathbf{s})$ from the corresponding composition multiset $C(\mathbf{s})$. We will use the bijection in Lemma 1 to design our reconstruction algorithm, and subsequently in Section IV give different families of reconstruction codes that satisfy the aforementioned properties.

B. Prior Work

In this section, we first review the results of [26] 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. Thereafter, to construct our reconstruction code, we review some elementary results from random walks, and revisit the design of the reconstruction code introduced in [29].

Definition 3. *For a binary string* $\mathbf{s} = s_1 s_2 \dots s_n$*, a bivariate polynomial* $P_{\mathbf{s}}(x,y)$ *of degree* n *is defined such that* $P_{\mathbf{s}}(x,y) = \sum_{i=0}^{n} (P_{\mathbf{s}}(x,y))_i$*, where* $(P_{\mathbf{s}}(x,y))_0 = 1$ *and* $(P_{\mathbf{s}}(x,y))_i$ *is defined recursively as*

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

$P_{\mathbf{s}}(x,y)$ 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 $\mathbf{s}_1^j$ of $\mathbf{s}$ has composition $1^w 0^z$. Similar to the bivariate polynomial for a binary string, we describe a bivariate polynomial $S_{\mathbf{s}}(x,y)$ corresponding to every composition multiset. 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 $S_{\mathbf{s}}(x,y)$.

Example 3. *If* $\mathbf{s} = 1001$*, then,* $C(\mathbf{s}) = \{0^1, 0^1, 1^1, 1^1, 0^1 1^1, 0^1 1^1, 0^2, 0^2 1^1, 0^2 1^1, 0^2 1^2\}$*,* $P_{\mathbf{s}}(x,y) = 1 + x + xy + xy^2 + x^2 y^2$*, and* $S_{\mathbf{s}}(x,y) = 2x + 2y + 2xy + y^2 + 2x^2 y + x^2 y^2$*.*

We use the following identity from [26]:

$$P_{\mathbf{s}}(x,y)\, P_{\mathbf{s}}\left(\frac{1}{x}, \frac{1}{y}\right) = (n+1) + S_{\mathbf{s}}(x,y) + S_{\mathbf{s}}\left(\frac{1}{x}, \frac{1}{y}\right). \tag{2}$$

Definition 4. *For a polynomial* $f(x,y)$*, let* $f^*(x,y)$ *be the polynomial (also known as reciprocal polynomial) defined as:*

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

It is easy to see that $f^*(x,y)$ is indeed a polynomial.

Remark 2. *If* $P_{\mathbf{s}}(x,y)$ *is the bivariate polynomial for the string* $\mathbf{s}$*, then* $P^*_{\mathbf{s}}(x,y) = P_{\mathbf{s}^*}(x,y)$*; that is* $P^*_{\mathbf{s}}(x,y)$ *is the bivariate polynomial corresponding to the reverse string* $\mathbf{s}^* = s_n s_{n-1} \dots s_1$*.*

Definition 5. *For a binary string* $\mathbf{s}$ *of length* n*, and the corresponding polynomial* $P_{\mathbf{s}}(x,y)$*, we define a polynomial* $F_{\mathbf{s}}(x,y)$ *as:*

$$F_{\mathbf{s}}(x,y) \stackrel{\text{def}}{=} P_{\mathbf{s}}(x,y) P^*_{\mathbf{s}}(x,y). \tag{4}$$

Rewriting (2), and using the definition in (4),

$$F_{\mathbf{s}}(x,y) = x^{\deg_x(P_{\mathbf{s}})} y^{\deg_y(P_{\mathbf{s}})} (n + 1 + S_{\mathbf{s}}(x,y)) + S^*_{\mathbf{s}}(x,y). \tag{5}$$

Corollary 2. *The polynomial* $F_{\mathbf{s}}(x,y)$ *can be evaluated directly from the composition multiset.*

Lemma 3 ([26], Lemma 8)**.** *For a binary string* $\mathbf{s}$*, the polynomial* $F_{\mathbf{s}}(x, y)$ *uniquely determines the composition multiset.*

Corollary 4. *There is a bijection between the composition multiset* $C(\mathbf{s})$ *and the polynomial* $F_{\mathbf{s}}(x, y)$*.*

Now, we discuss the preliminaries required for the design of reconstruction code introduced in Section IV. Lemma 5 gives sufficient conditions for a binary string to be uniquely reconstructed in polynomial time complexity by the reconstruction algorithm in [26]. Recall from Remark 1, that we restrict the analysis of reconstruction codes to the subsets of strings of length n beginning with 1 and ending at 0.

Definition 6. *If a binary string* $\mathbf{s}$ *of length* n*, is such that for all prefix-suffix pairs of length* $1 \leqslant j \leqslant n$*, one has* $wt(\mathbf{s}_1^j) \neq wt(\mathbf{s}_{n+1-j}^n)$*, then* $\mathbf{s}$ *will be called an imbalanced string.*

Remark 3. *For an imbalanced string* $\mathbf{s}$*, note that*

$$\left|\left|wt(\mathbf{s}_1^{j+1}) - wt(\mathbf{s}_{n-j}^n)\right| - \left|wt(\mathbf{s}_1^j) - wt(\mathbf{s}_{n+1-j}^n)\right|\right| \leqslant 1.$$

Since $\left|wt(\mathbf{s}_1^j) - wt(\mathbf{s}_{n+1-j}^n)\right| \neq 0$ *for any* $1 \leqslant j \leqslant \lceil n/2 \rceil$*, the sign of* $wt(\mathbf{s}_1^j) - wt(\mathbf{s}_{n+1-j}^n)$ *does not change with* j*. Therefore, for imbalanced strings of length* n *that begin with* 1 *and end with* 0*,* $wt(\mathbf{s}_1^j) > wt(\mathbf{s}_{n+1-j}^n)$ *for all* $1 \leqslant j < n$*.*

Lemma 5 ([26], Lemma 37)**.** *An imbalanced string* $\mathbf{s}$ *of length* n *is uniquely reconstructable in* $O(n^2 \log n)$ *time by the reconstruction algorithm of [26].*

In Section IV, we show a bijection between *imbalanced* strings of length n that begin with 1 and end with 0, and positive n-step walks. Using this bijection, we explicitly characterize the set of binary strings reconstructable by the algorithm in [26].

Definition 7. *A* 1*-dimensional positive* n*-step walk is defined as an assignment of* n *variables* $X_i \in \{-1, 1\}$ *for* $1 \leqslant i \leqslant n$*, such that* $S_k = \sum_{i=1}^{k} X_i$ *is positive for* $1 \leqslant k \leqslant n$*.*

Lemma 6 ([32], Lemma 3.1)**.** *The number of* 1*-dimensional positive* n*-step walks is* $\binom{n-1}{\lfloor \frac{n-1}{2} \rfloor}$*.*

The reconstruction code in [29] uses Catalan-Bertrand strings to construct a codebook. This codebook consists of strings with the property that any of their prefixes contains strictly more 0s than 1s, referred to as Catalan-Bertrand strings. The codebook is designed in such a way that for any given codeword and any same-length prefix-suffix substring pair of that codeword, the two substrings have distinct weights i.e. all strings belonging to the codebook are imbalanced, as defined in Definition 6.

Definition 8 ([29])**.** *For reconstruction code* $S_R(n)$ *of even length (*n *even):*

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

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

$$\begin{aligned} S_R(n) \stackrel{\text{def}}{=} \{&\mathbf{s}_1^{(n-1)/2} 0 \mathbf{s}_{(n+1)/2}^{n-1},\ \mathbf{s}_1^{(n-1)/2} 1 \mathbf{s}_{(n+1)/2}^{n-1},\\ &\textit{where } \mathbf{s} \in S_R(n-1)\}. \end{aligned}$$

The authors in [29] extend this coding scheme to correct single and multiple mass errors. These code extensions relied only upon the fact that all strings in $S_R(n)$ are *imbalanced* strings. In [19], the authors show an equivalence between the set of imbalanced strings beginning with 0, and ending with 1, and the codebook $S_R(n)$. In Section IV, we show a bijection mapping between all imbalanced strings beginning with 1 and ending with 0 and positive n-step walks, thereby explicitly characterizing the size of the code $S_R(n)$, and describing an efficient encoding and decoding procedure for this codebook.

Lemma 7 ([19], Lemma IV.2)**.** $S_R(n)$ *is the set of all imbalanced binary strings of length* n *beginning with* 0*, and ending in* 1*.*

Finally, we give well known bounds on the central binomial coefficient which we will use to show the rate of our reconstruction code.

Proposition 8 ([33])**.** *The central binomial coefficient may be bounded as:*

$$\frac{4^n}{\sqrt{\pi(n+1/2)}} \leqslant \binom{2n}{n} \leqslant \frac{4^n}{\sqrt{\pi n}},\quad \forall\ n \geqslant 1. \tag{6}$$

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 $\mathbf{s} = s_1 \ldots s_n$ 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 $A(\mathbf{s}) = a_0 \ldots a_d$, starting from both ends and progressing towards the center. In other words, a_0 and a_d are recovered first, followed by a_1 and a_{d-1}, etc.; and the algorithm backtracks when there is an error in recovering a pair. The algorithm takes as input the polynomial $F(x, y)$ (Definition 5). Note that the polynomial $F(x, y)$ can be derived from $S(x, y)$ (Corollary 2) which in turn is equivalent to the corresponding composition multiset. The algorithm will return the set of strings which have the given composition multiset. We will use the fact that for a string $\mathbf{s}$ with the given composition multiset, we must have $F_{\mathbf{s}}(x, y) = F(x, y)$. Then Lemma 3 guarantees that strings recovered in this way indeed have the desired composition multiset. In Proposition 10, we outline the conditions in the strings that result in no backtracking throughout the reconstruction process using the proposed algorithm. Before the algorithm is discussed, we first show how certain parameters of a string $\mathbf{s}$ with the given composition multiset can be readily recovered from the polynomial $F(x, y)$. These parameters are shared by all the strings that share this composition multiset, and will be subsequently used as inputs to the algorithm.

For a string $\mathbf{s} = s_1 \ldots s_n$ with $s_1 = 1, s_n = 0$, the corresponding non-negative integer string $A(\mathbf{s})$ (Definition 1) is

such that $a_0 = 0$ and $a_d \geqslant 1$. Using definitions 3 and 4,

$$P_{\mathbf{s}}(x,1) = 1 + (a_1+1)\,x + \cdots + (a_d+1)\,x^d, \quad (7)$$

$$P^*_{\mathbf{s}}(x,1) = (a_d+1) + \cdots + (a_1+1)\,x^{d-1} + x^d. \quad (8)$$

Since a string $\mathbf{s}$ with the given composition multi-set must have $F_{\mathbf{s}}(x,y) = F(x,y)$, from Definition 5: $F(x,1) = F_{\mathbf{s}}(x,1) = P_{\mathbf{s}}(x,1)P^*_{\mathbf{s}}(x,1)$. Therefore, using (7) and (8), the weight of the string $\mathbf{s}$ and a_d (where $A(\mathbf{s}) = a_0 \ldots a_d$) can be recovered from $F(x,y)$ as follows:

$$wt(\mathbf{s}) = d = \frac{\deg F(x,1)}{2}, \text{ and } a_d = F(0,1) - 1. \quad (9)$$

The algorithm will utilize the polynomial formulation of the problem by mapping binary strings to elements of a polynomial ring by considering the coefficients of the polynomials $P_{\mathbf{s}}(x,y)$ and $F_{\mathbf{s}}(x,y)$ as elements of a sufficiently large finite field, i.e., $\mathbb{F}_q$ with q being a prime number greater than $5n+1$ (By Bertrand's postulate, there is a prime q with $5n+1 < q < 10n$). We will discuss several properties of the polynomials $P_{\mathbf{s}}(x,y)$ and $P^*_{\mathbf{s}}(x,y)$ (which lie in the ring $\mathbb{F}_q[x,y]$) which we use in the algorithm. Recall that for strings beginning with 1 and ending at 0, we have $a_0 = 0$.

Definition 9. *Given integers $a_0, \ldots, a_j$ and $a_d, \ldots, a_{d-j}$ in $\mathbb{N} \cup \{0\}$; define the polynomials $\alpha_j(y)$ and $\beta_j(y)$ as follows:*

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

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

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

Note that the equations (10) and (11) can be rewritten as

$$(y-1)\cdot\alpha_j(y) = y^{1+g_0^j} - y^{g_0^{j-1}}, \quad (12)$$

$$(y-1)\cdot\beta_j(y) = y^{1+g_{d-j}^d} - y^{g_{d-j+1}^d}, \quad (13)$$

where $g_0^{-1} = 0 = g_{d+1}^d$. For a string $\mathbf{s}$ of length n, and weight d, let the corresponding integer string be $A(s) = a_0 \ldots a_d$ (Definition 1). Then, using Definitions 3 and 4, for $0 \leqslant j \leqslant d$, $\alpha_j(y)$ and $\beta_j(y)$ are the coefficients of x^j in $P_{\mathbf{s}}(x,y)$ and $P^*_{\mathbf{s}}(x,y)$, respectively. In particular,

$$P_{\mathbf{s}}(x,y) = \sum_{j=0}^{d} \alpha_j(y)x^j, \text{ and } P^*_{\mathbf{s}}(x,y) = \sum_{j=0}^{d} \beta_j(y)x^j. \quad (14)$$

Remark 4. *$\alpha_j(\gamma)$ and $\beta_j(\gamma)$ correspond to the coefficients of x^j in $P_{\mathbf{s}}(x,\gamma)$ and $P^*_{\mathbf{s}}(x,\gamma)$, 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 $P_{\mathbf{s}}(x,1)$ ((7)) and $P^*_{\mathbf{s}}(x,1)$ ((8)), respectively.*

Remark 5. *For a string $\mathbf{s}$ of length n, and weight d, let the corresponding integer string be $A(\mathbf{s}) = a_0 a_1 \ldots a_d$ (Definition 1). Let $r_{\mathbf{s},j}(y)$ denotes the coefficient of x^j in the polynomial $F_{\mathbf{s}}(x,y)$. Then using $F_s(x,y) = P_s(x,y)P^*_s(x,y)$,*

$$r_{\mathbf{s},j}(y) = \sum_{k=0}^{j} \alpha_{j-k}(y)\beta_k(y). \quad (15)$$

Using (12) *and* (13)*, this equation can be re-written as*

$$(y-1)^2 \cdot r_{\mathbf{s},j}(y) = -\sum_{k=1}^{j-1} \Big(y^{2+g_0^{j-k}+g_{d-k}^d} + y^{g_0^{j-k-1}+g_{d-k+1}^d} - y^{1+g_0^{j-k}+g_{d-k+1}^d} - y^{1+g_0^{j-k-1}+g_{d-k}^d} \Big). \quad (16)$$

where g_k^l denotes the sum $a_k + a_{k+1} \ldots + a_l$ (defined in Section II-A) and $g_0^{-1} = 0 = g_{d+1}^d$.

The reconstruction algorithm will find a_j and a_{d-j} together at step j. Note that in (10), $\alpha_i(y)$ 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 $\mathbf{s}$, if by the end of step $j-1$, the algorithm recovers the pairs $(a_1, a_{d-1}), (a_2, a_{d-2}), \ldots, (a_{j-1}, a_{d-j+1})$; the polynomials $\alpha_0(y), \ldots, \alpha_{j-1}(y)$ and $\beta_0(y), \ldots, \beta_{j-1}(y)$ are well defined.

Definition 10. *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:*

$$f_j(y) \stackrel{\text{def}}{=} r_j(y) - \sum_{k=1}^{j-1} \alpha_k(y)\beta_{j-k}(y). \quad (17)$$

By the end of step $j-1$, since we know the polynomials $\alpha_0(y), \ldots, \alpha_{j-1}(y)$ and $\beta_0(y), \ldots, \beta_{j-1}(y)$, we can compute $f_j(y)$. At step j, the algorithm wants to find the pair (a_j, a_{d-j}). If the pairs $(a_1, a_{d-1}), \ldots, (a_{j-1}, a_{d-j+1})$ are identified correctly, then for the correct pair (a_j, a_{d-j}), the coefficient of x^j in $F_{\mathbf{s}}(x,y) \in \mathbb{F}_q[x,y]$ is $r_{\mathbf{s},j} = \sum_{i=0}^{j} \alpha_i(y)\beta_{j-i}(y)$ (Remark 5). Since we aim to identify strings $\mathbf{s}$ for which $F_{\mathbf{s}}(x,y) = F(x,y)$, we need to find (a_j, a_{d-j}) such that $\sum_{i=0}^{j} \alpha_i(y)\beta_{j-i}(y) = r_j(y)$. As discussed before, by the end of step $j-1$, we already know $\alpha_0(y), \ldots, \alpha_{j-1}(y)$ and $\beta_0(y), \ldots, \beta_{j-1}(y)$; therefore, a correct pair (a_j, a_{d-j}) must satisfy

$$\begin{aligned} f_j(y) &= \alpha_0(y)\beta_j(y) + \alpha_j(y)\beta_0(y) \\ &= \beta_j(y) + \alpha_j(y)\beta_0(y). \end{aligned} \quad (18)$$

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

$$\begin{aligned} deg(f_j) &= \max\{deg(\beta_j), deg(\alpha_j\beta_0)\} \\ &= \max\{g_{d-j}^d, g_0^j + a_d\}. \end{aligned} \quad (19)$$

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:

$$f_j(1) = (1 + a_{d-j}) + (a_d+1)\,(a_j+1)\,. \quad (20)$$

We will use these equations to compute the possible values for the pairs (a_j, a_{d-j}). Since $1 \leqslant (a_d+1) \leqslant n < q$, we know that $q \nmid (a_d+1)$; equations (19) and (20) give us two possible values for the pair (a_j, a_{d-j}) at step j. This procedure is captured in the Reconstruction Algorithm. At the end of $\lceil d/2 \rceil$ steps, we find strings $\mathbf{s}$ for which $r_{\mathbf{s},j}(y) = r_j(y)$ for all $0 \leqslant j \leqslant \lceil d/2 \rceil$. Since, $F(x,y) = F^*(x,y)$ and $F_s(x,y) = F^*_s(x,y)$, we have $r_{\mathbf{s},j}(y) = r_j(y)$ for all $0 \leqslant j \leqslant d$. There-

fore,

$$F_{\mathbf{s}}(x,y) = \sum_{k=0}^{d} r_{\mathbf{s},j}(y)x^k = \sum_{k=0}^{d} r_j(y)x^k = F(x,y).$$

The correctness of the algorithm is guaranteed by the Lemma 3 which shows that the strings which share the same $F_{\mathbf{s}}(x,y) = F(x,y)$ 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: Set $\mathcal{S}$ of codestrings $\mathbf{s} \in \{0,1\}^n$

Function `Reconstruction` $(j = 1,\ F,\ A)$**:**

$S = \varnothing$
if $j = \lceil d/2 \rceil$ **then**
  if $d \mod 2 = 0$ **then**
    $a_j = n - d - \sum_{i \neq j} a_i$
  if *A corresponds to some binary string* $\mathbf{s}$ **then**
    $\mathcal{S} = \mathbf{s}$
  else
    $\mathcal{S} = \varnothing$
  end
  return $\mathcal{S}$
end

Compute f_j, $deg(f_j)$, and $f_j(1)$
flag = 0

$a_j = deg(f_j) - (g_0^{j-1} + a_d)$
$a_{d-j} = f_j(1) - 1 - (a_d + 1)(a_j + 1)$

if $a_j \geqslant 0$, $a_{d-j} \geqslant 0$, *and* $f_j(y) = \beta_j(y) + \alpha_j(y)\beta_0(y)$ **then**
  $A[j] = a_j, A[d-j] = a_{d-j}$
  $\mathcal{S} =$ `Reconstruction` $(j+1,\ F,\ A)$
  flag = 1
end

$a_{d-j} = deg(f_j) - g^d_{d-j+1}$
$a_j = -1 + (f_j(1) - 1 - a_{d-j})/(a_d + 1)$

if $a_j \in \mathbb{N} \cup \{0\}$, $a_{d-j} \geqslant 0$, *and* $f_j(y) = \beta_j(y) + \alpha_j(y)\beta_0(y)$ **then**
  $A[j] = a_j, A[d-j] = a_{d-j}$
  $\mathcal{S} = \mathcal{S} \cup$ `Reconstruction` $(j+1,\ F,\ A)$
  flag = 1
end

if *flag = 0* **then**
  $\mathcal{S} = \varnothing$
end
return $\mathcal{S}$

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. both choices are valid according to the algorithm; then our algorithm must choose one direction to proceed. If an error is encountered later, the algorithm comes back to the last branch (not taken yet) where both conditions were satisfied and takes the alternate path. If exactly one condition is satisfied, then our algorithm takes the corresponding path. If neither of the two conditions are satisfied, then assuming the input composition multiset to be valid, our algorithm must have taken the wrong branch in the past (when it had a choice). In such a scenario, our algorithm goes back to the last valid branch where both conditions were satisfied, and takes the alternate branch and proceeds as described. In Theorem 9 stated below, we establish the time and space complexity of the proposed reconstruction algorithm (Algorithm 1). The proof of the theorem demonstrates how these complexities can be achieved by pre-storing certain information.

Theorem 9. *Given the input $F(x,y)$, the proposed Reconstruction Algorithm (Algorithm 1), in the case of no backtracking, outputs the unique string* $\mathbf{s}$ *with $F_s(x,y) = F(x,y)$ with maximum $6.5n^2$ finite field operations, translated to maximum $6.5n^2(3.22+\log n)$ binary operations, latency $O(n \log n)$, and space complexity $O(n^2)$.*

Proof: Note that at step j of the algorithm, instead of computing $f_j(y)$ directly to find $deg(f_j)$, and verifying if $f_j(y) = \beta_j(y) + \alpha_j(y)\beta_0(y)$, we can instead do the following. First, we compute $f'_j(y) = (y-1)^2 \cdot f_j(y)$. Then $deg(f_j) = deg(f'_j) - 2$, and we will verify if $f'_j(y) = (y-1)^2 \cdot (\beta_j(y) + \alpha_j(y)\beta_0(y))$. Using (12) and (13), this is equivalent to verifying whether the following holds:

$$\begin{aligned} f'_j(y) = {} & y^{2+g^d_{d-j}} + y^{g^d_{d-j+1}} - y^{1+g^d_{d-j+1}} - y^{1+g^d_{d-j}} \\ & + y^{2+a_d+g_0^j} + y^{g_0^{j-1}} - y^{1+a_d+g_0^{j-1}} - y^{1+g_0^j}. \end{aligned} \quad (21)$$

The expression in (21) is summation of at most 8 distinct powers of y, i.e., a polynomial with at most 8 non-zero coefficients. This implies that, for $1 \leqslant i \leqslant (j-1)$, if (a_i, a_{d-i}) have been correctly identified, $f'_j(y)$ can have at most 8 non-zero coefficients. Using (12) and (13) again, $f'_j(y)$ can be computed using

$$\begin{aligned} f'_j(y) = {} & (y-1)^2 \cdot r_j(y) \\ & - \sum_{k=1}^{j-1} \Big(y^{2+g_0^{j-k}+g^d_{d-k}} + y^{g_0^{j-k-1}+g^d_{d-k+1}} \\ & \quad - y^{1+g_0^{j-k}+g^d_{d-k+1}} - y^{1+g_0^{j-k-1}+g^d_{d-k}} \Big). \end{aligned} \quad (22)$$

To perform field operations more efficiently, and to effectively use the structure of the polynomials used in the algorithm ((10), (11), and (17)), we will pre-store certain information in arrays, to be specified next. Let $\lambda \in \mathbb{F}_q$ be a primitive element of the field. Then define the following arrays:

- $field\_store[]$ of length $q-1$,
  where for $0 \leqslant i \leqslant q-2$, $field\_store[i] = \lambda^i$.
- $element\_store[]$ of length $(q-1)$,
  where for $1 \leqslant i \leqslant q-1$, $element\_store[i-1] = c$ such that $i = \lambda^c$.
- $r\_coef[,]$ of size $(d+1) \times (n-d+1)$,
  where $r\_coef[k,l] = a_{k,l}$, where, $a_{k,l}$ denotes the coefficient of y^l in $r_k(y)$ (Definition 10).
- $r\_coef\_mult[,]$ of size $(d+1) \times (n-d+3)$,
  where $r\_coef\_mult[k,l] = a'_{k,l}$, where, $a'_{k,l}$ denotes the

coefficient of y^l in $(y-1)^2 \cdot r_k(y)$.

- $r\_one\_eval[]$ of length $(d+1)$,
  where for $0 \leqslant i \leqslant d$, $r\_one\_eval[i] = r_i(1)$.
- $g\_begin[]$ of length $(d+1)$,
  Initialized with $g\_begin[0] = g_0^0 = a_0 = 0$ and the remaining values all set to -1. At step j, we will update $g\_begin[j] = g\_begin[j-1] + a_j$.
- $g\_end[]$ of length $(d+1)$,
  Initialized with $g\_end[0] = g_d^d = a_d$, and the remaining values all set to -1. At step j, we will update $g\_end[j] = g\_end[j-1] + a_{d-j}$.

At step j, we will first compute and store the coefficients of f'_j in an array $f\_dash[]$ of size $(n+3)$, initialized with the coefficients of $(y-1)^2 \cdot r_j(y)$, that is, for $0 \leqslant i \leqslant (n+2)$, $f\_dash[i] = r\_coef\_mult[j,i]$. In (22), the powers of y only involve g_0^k, and g_{d-k}^d, for $1 \leqslant k \leqslant (j-1)$. At the beginning of step j, we readily have access to these elements in the arrays $g\_begin[]$ and $g\_end[]$. Therefore, to compute f'_j, we perform the following operations for $1 \leqslant k \leqslant (j-1)$,

$$\begin{aligned}
f_dash[2 + g_0^{j-k} + g_{d-k}^d] &\leftarrow f_dash[2 + g_0^{j-k} + g_{d-k}^d] - 1,\\
f_dash[g_0^{j-k-1} + g_{d-k+1}^d] &\leftarrow f_dash[g_0^{j-k-1} + g_{d-k+1}^d] - 1,\\
f_dash[1 + g_0^{j-k} + g_{d-k+1}^d] &\leftarrow f_dash[1 + g_0^{j-k} + g_{d-k+1}^d] + 1,\\
f_dash[1 + g_0^{j-k-1} + g_{d-k}^d] &\leftarrow f_dash[1 + g_0^{j-k-1} + g_{d-k}^d] + 1.
\end{aligned}$$

That is, we do $O(d)$ field operations to update the array $f\_dash[]$ with $2(j-1)$ operations adding 1, and $2(j-1)$ operations subtracting 1. Note that this process takes $11(j-1)$ field operations, and can be parallelized to $O(1)$ latency. Thus, the coefficients of the polynomial $f'_j(y)$ are computed and stored in the matrix $f\_dash[]$ in $11(j-1)$ field addition operations, and $O(1)$ latency.

To compute the degree of the polynomial f'_j, a naive approach is to find the largest index with a non-zero array entry. This can be done in n field comparisons in a worst case sense. However, since we know that f'_j can only have at most 8 non-zero coefficients, finding the degree of the polynomial can be efficiently parallelized to $O(1)$ time. This can be done by asking if array entries at each index are non-zero, in parallel. Recording these non-zero indexes (along with the corresponding array entry) as a list of size at most 8, the largest member of index value can be computed in $O(1)$ time. Note that if the list size exceeds 8, the polynomial f'_j has been inferred incorrectly; we declare an error and backtrack. Therefore, $deg(f_j)$ can be computed with n field comparison operations and $O(1)$ latency.

To compute $f_j(1)$, recall from (17),

$$\begin{aligned}
f_j(1) &= r_j(1) - \sum_{k=1}^{j-1} \alpha_{j-k}(1)\beta_k(1)\\
&= r_j(1) - \sum_{k=1}^{j-1} (1 + a_{j-k})(1 + a_{d-k}). \qquad (23)
\end{aligned}$$

This can be computed using the $r\_one\_eval[]$, $field\_store[]$ and $element\_store[]$ arrays. This is done to avoid multiplication in the finite field, which is an asymptotically costly operation.

$$f_j(1) = r\_one\_eval[j] - \sum_{k=1}^{j-1} field\_store\Big[element\_store[a_{j-k}] + element\_store[a_{d-k}]\Big]. \qquad (24)$$

Since addition can be parallelized, $f_j(1)$ can be computed in j field additions and $O(\log n)$ latency.

For each branch of the algorithm induced by the **if** statements, we obtain the values of the pair (a_j, a_{d-j}), which we first use to update $g\_begin[j] = g_0^j$ and $g\_end[j] = g_{d-j}^d$. Note that the coefficients of the polynomial f'_j were already computed and stored in the array $f\_dash[]$. Also, recall that the polynomial f'_j can have at most 8 non-zero coefficients, whose indices are saved in a list while computing $deg(f_j)$. Using the computed value of the pair (a_j, a_{d-j}), the polynomial f'_j can be verified using (21) in 8 field additions. Now we update the arrays $g\_begin[]$ and $g\_end[]$ using 2 field addition operations.

Therefore, for each value of j, the algorithm requires $n + 12j - 3 < 13n$ field operations. Therefore, over $\lceil d/2 \rceil$ values of j, in the case of no backtracking, the algorithm will output the required string $\mathbf{s}$ in less than $(d+1)(6.5n) < 6.5n^2$ field operations, and $O(d \log n) = O(n \log n)$ latency. Since the size of the field $\mathbb{F}_q$ is $q < 10n$, each addition and comparison operation can be done with at most $\log_2(10n)$ binary operations, which means that our algorithm requires at most $6.5n^2(3.22 + \log n)$ binary operations in its implementation. This complexity is achieved by storing $(2q-1)+(d+1)(2n+7)$ field elements in advance. Hence, the required space complexity is $O(n^2)$. ∎

Remark 6. *In the case of no backtracking, the time complexity of the reconstruction algorithm proposed by Acharya et. al. in [26] is $O(n^2 \log n)$, which is same as the reconstruction complexity of our algorithm order-wise (i.e., asymptotically). Note that the algebraic nature of our reconstruction algorithm has enabled us to exactly upper bound the total number of binary operations needed, as outlined in Theorem 9. However, such an analysis is not available for the algorithm in [26].*

Remark 7. *For a binary string $\mathbf{s}$ of length n, the composition multiset $C(\mathbf{s})$ has $n(n+1)/2$ elements, which requires $O(n^2)$ space to be stored. Therefore, it is natural for reconstruction algorithms, including our algorithm as well as the reconstruction algorithm proposed by Acharya et. al. in [26], to require $O(n^2)$ space complexity.*

Example 4. *We demonstrate how to reconstruct the strings $\mathbf{s_1} = 10010110$ and $\mathbf{s_2} = 10110010$ via the process detailed above with the pseudocode provided in Reconstruction Algorithm. The strings s_1 and s_2 share the same composition multiset, and therefore $F_{\mathbf{s}_1}(x,y) = F_{\mathbf{s}_2}(x,y)$. For these strings, the corresponding integer strings are $A(\mathbf{s}_1) = 02101$ and $A(\mathbf{s}_2) = 01021$.*

We will be given the composition multiset, or equivalently, $F(x,y) = (1+y) + x(1 + 3y + 2y^2 + y^3) + \cdots$. The reconstruction algorithm should output the set of strings $\mathcal{S} = \{\mathbf{s_1} = 10010110, \mathbf{s_2} = 10110010\}$. From (9), we readily

obtain that $d = 4$, and $a_d = a_4 = 1$. The algorithm begins by finding the tuple (a_1, a_3) corresponding to $j = 1$. To find the tuple, the algorithm will need to compute certain attributes of the polynomial $f_{j=1}(y)$ (Definition 10) which is simply the coefficient of x in $F_{\mathbf{s}}(x, y)$ i.e., $(1 + 3y + 3y^2 + y^3)$. In particular, using the procedure described in Theorem 9, the algorithm first computes $f_j(y)$, and obtains $deg(f_j) = 3$, and $f_j(1) = 7$.

To check for the first branch, the algorithm, with $j = 1$, sets $a_1 = deg(f_1) - (g_0^0 + a_4) = 3 - (0+1) = 2$, and, with $d - j = 3$, $a_3 = f_1(1) - 1 - (a_4 + 1)\,(a_1 + 1) = 7 - 1 - (1+1)(2+1) = 0$. The algorithm now verifies if the computed $f_j(y)$ and the polynomial $\beta_j(y) + \alpha_j(y)\beta_0(y)$ obtained using the computed values of the pair (a_1, a_3) coincide. The process of doing this efficiently is explained in the proof of Theorem 9. This indeed is the required tuple corresponding to the string $\mathbf{s}_1$ (recall that $A(\mathbf{s}_1) = 02101$), and therefore must satisfy the required relationship. The algorithm therefore concludes $(a_1, a_3) = (2, 0)$. The sequence element a_2 can now be trivially calculated as $a_2 = d - (a_0 + a_4) - (a_1 + a_3) = (4 - (0 + 1) - (2 + 0) = 1$. Therefore, at the end of the first branch, we have $\mathcal{S} = \{10010110\}$.

Similarly, to check for the second branch, the algorithm, with $d - j = 3$, sets $a_3 = deg(f_1) - g_4^4 = 3 - 1 = 2$, and $a_1 = -1 + (f_1(1) - 1 - a_3)/(a_4 + 1) = 1 \in \mathbb{N}$. The algorithm now verifies if the computed $f_j(y)$ and the polynomial $\beta_j(y) + \alpha_j(y)\beta_0(y)$ obtained using the computed values of the pair (a_1, a_3) coincide. Since this tuple is also a valid tuple corresponding to the string s_2 (recall that $A(\mathbf{s}_2) = 01021$), after verification, must satisfy the required relationship. The algorithm now concludes that $(a_1, a_3) = (1, 2)$. The value of a_2 can now be calculated as $a_2 = d - (a_0 + a_4) - (a_1 + a_3) = (4 - (0 + 1) - (2 + 0) = 0$. Therefore, at the end of the second branch, we have $\mathcal{S} = \{10010110, 10110010\}$.

We say that a string $\mathbf{s}$ *stops* at step j if the algorithm fails to uniquely determine (a_j, a_{d-j}) at step j. As explained above, this is possible if either both or neither of the two **if** conditions are satisfied. In both cases, the algorithm had a step $j' \leqslant j$ where both of the two conditions were satisfied. Therefore, we will say a string $\mathbf{s}$ *pauses* at step j if there are two acceptable branches for the tuple (a_j, a_{d-j}). Note that in Example 4, while reconstructing the strings, both of the two possible solutions were satisfied for the tuple (a_j, a_{d-j}) corresponding to $j = 1$. Therefore, we can say that the reconstruction algorithm in Example 4 *paused* at step $j = 1$. In the following lemma, we give algebraic conditions (25) and (26), characterizing the strings that *pause* at some step j.

Proposition 10. *Let the bi-variate polynomial corresponding to a string $\mathbf{s}$ be $F_{\mathbf{s}}(x, y)$. If the reconstruction algorithm pauses at step j, then string $\mathbf{s}$ satisfies either of the following two relations:*

$$g_0^j - g_{d-j}^d = a_0 + 1 = 1, \text{ and } a_j \geqslant 1; \tag{25}$$

$$g_{d-j}^d - g_0^j = a_d + 1, \qquad \text{and } a_{d-j} \geqslant a_d + 1. \tag{26}$$

Proof: Since the reconstruction algorithm pauses at step j, both **if** statements corresponding to the two branches induced by the reconstruction algorithm must be satisfied. Therefore, there must exist a pair of tuples, which we call (a_j, a_{d-j}) and (a'_j, a'_{d-j}), such that both of them satisfy (18) for all $y \in \mathbb{F}_q$. Call the polynomials corresponding to these pairs $(\alpha_j(y), \beta_j(y))$ and $(\alpha'_j(y), \beta'_j(y))$ respectively. Let $h_j(y) = \beta_j(y) + \alpha_j(y)\beta_0(y)$, and $h'_j(y) = \beta'_j(y) + \alpha'_j(y)\beta'_0(y)$. Then these polynomials must satisfy

$$\begin{aligned} f_j(y) &= \beta_j(y) + \alpha_j(y)\beta_0(y) = h_j(y) \\ &= \beta'_j(y) + \alpha'_j(y)\beta'_0(y) = h'_j(y). \end{aligned} \tag{27}$$

Let $\lambda \in \mathbb{F}_q$ be a primitive element of the field. In particular, the following relations must be satisfied:

$$\begin{aligned} f_j(\lambda) &= \beta_j(\lambda) + \alpha_j(\lambda)\beta_0(\lambda) = h_j(\lambda) \\ &= \beta'_j(\lambda) + \alpha'_j(\lambda)\beta'_0(\lambda) = h'_j(\lambda), \end{aligned} \tag{28}$$

and

$$\begin{aligned} f_j(\lambda^{-1}) &= \beta_j(\lambda^{-1}) + \alpha_j(\lambda^{-1})\beta_0(\lambda^{-1}) = h_j(\lambda^{-1}) \\ &= \beta'_j(\lambda^{-1}) + \alpha'_j(\lambda^{-1})\beta'_0(\lambda^{-1}) = h'_j(\lambda^{-1}). \end{aligned} \tag{29}$$

By the step $j - 1$, we know g_0^{j-1} and g_{d-j+1}^d. Using (19),

$$g_{d-j+1}^d + a'_{d-j} = deg(f_j) = g_0^{j-1} + a_j + a_d. \tag{30}$$

Using (20),

$$(a_j - a'_j)(a_d + 1) = (a'_{d-j} - a_{d-j}). \tag{31}$$

From (28),

$$f_j(\lambda) \;= \lambda^{g_{d-j+1}^d} \left(\sum_{i=0}^{a_{d-j}} \lambda^i \right) + \lambda^{g_0^{j-1}} \left(\sum_{i=0}^{a_j} \lambda^i \right) \left(\sum_{i=0}^{a_d} \lambda^i \right),$$

and

$$f_j(\lambda) = \lambda^{g_{d-j+1}^d} \left(\sum_{i=0}^{a'_{d-j}} \lambda^i \right) + \lambda^{g_0^{j-1}} \left(\sum_{i=0}^{a'_j} \lambda^i \right) \left(\sum_{i=0}^{a_d} \lambda^i \right).$$

Since $\lambda \neq 1$, i.e., $\lambda - 1$ is invertible, equating the two expressions and multiplying by $(\lambda - 1)^2$,

$$\begin{aligned} &\lambda^{g_{d-j+1}^d}(\lambda - 1)(\lambda^{a_{d-j}+1} - \lambda^{a'_{d-j}+1}) \\ &= \lambda^{g_0^{j-1}}(\lambda^{a_d+1} - 1)(\lambda^{a'_j+1} - \lambda^{a_j+1}). \end{aligned} \tag{32}$$

Similarly using (29), and equating the expressions after multiplying by $(\lambda^{-1} - 1)^2$;

$$\begin{aligned} &\lambda^{-g_{d-j+1}^d}(\lambda^{-1} - 1)(\lambda^{-a_{d-j}-1} - \lambda^{-a'_{d-j}-1}) \\ &= \lambda^{-g_0^{j-1}}(\lambda^{-a_d-1} - 1)(\lambda^{-a'_j-1} - \lambda^{-a_j-1}). \end{aligned}$$

Simplifying, we get

$$\begin{aligned} &\lambda^{-g_{d-j+1}^d - a_{d-j} - a'_{d-j} - 3}(\lambda - 1)(\lambda^{a_{d-j}+1} - \lambda^{a'_{d-j}+1}) \\ &= \lambda^{-g_0^{j-1} - a_j - a'_j - a_d - 3}(\lambda^{a_d+1} - 1)(\lambda^{a'_j+1} - \lambda^{a_j+1}). \end{aligned}$$

Now using relation (32) and equating power of λ (which can be done since λ is primitive root in a field of size $q > 5n+1$),

$$2g_{d-j+1}^d + a_{d-j} + a'_{d-j} = 2g_0^{j-1} + a_j + a'_j + a_d. \tag{33}$$

Solving the four equations obtained from (30), (31), and (33);

$$(a_j, a_{d-j}) = (t + g^d_{d-j+1} + 1, t + g_0^{j-1}), \quad (34)$$

$$(a'_j, a'_{d-j}) = (t + g^d_{d-j+1}, t + g_0^{j-1} + a_d + 1), \quad (35)$$

where $t = a_{d-j} - g_0^{j-1} = deg(f_j) - (1 + a_d + g_0^{j-1} + g^d_{d-j+1})$. The tuple (a_j, a_{d-j}) in (34) corresponds to the condition (25), and the tuple (a'_j, a'_{d-j}) in (35) corresponds to the condition (26). ∎

Definition 11. *We will call the strings which satisfy condition* (25) *for some* $0 < j < d/2$ *as type-1 strings, and the strings which satisfy condition* (26) *for some* $0 < j < d/2$ *as type-2 strings.*

Remark 8. *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. Since our algorithm can only confuse a type-1 string with a type-2 string, if our algorithm knows the type of string, it can know which branch to choose thereby avoiding backtracking. In Section IV, we will use this fact to design reconstruction codes by avoiding all strings of a single type.*

Remark 9. *In Example 4, while reconstructing the strings* $\mathbf{s_1} = 10010110$ *and* $\mathbf{s_2} = 10110010$ *(which share the same composition multiset), the algorithm pauses at step* $j = 1$*. For these strings, the corresponding integer strings are* $A(\mathbf{s}_1) = 02101$ *and* $A(\mathbf{s}_2) = 01021$*. Note that, for string* $\mathbf{s_1}$*,* $g_0^1 - g_3^4 = (0+2) - (0+1) = 1$*, satisfying* (25)*. Similarly, for string* $\mathbf{s_2}$*,* $g_3^4 - g_0^1 = (2+1) - (0+1) = 2 = a_4 + 1$*, satisfying* (26)*. Therefore, string* $\mathbf{s_1}$ *is a type-1 string and string* $\mathbf{s_2}$ *is a type-2 string.*

Corollary 11. *If an imbalanced string* $\mathbf{s}$ *(Definition 6) of length* n *is such that it begins in* 1 *and ends at* 0*, then* $\mathbf{s}$ *can be uniquely reconstructed with the complexity outlined in Theorem 9.*

Proof: We will show that an *imbalanced* string cannot be a *type-1 string*. As discussed in the previous remark, telling our algorithm to always choose condition (26) in case of a *pause*, any such string can be reconstructed without backtracking and hence according to the process in Theorem 9.

Let if possible, $\mathbf{s}$ also be a *type-1 string*. Let step j be the first time the string $\mathbf{s}$ *pauses* and satisfies condition (25). If condition (25) is satisfied, then the $(j+1)^{th}$ one in $\mathbf{s}$ is at position $(g_0^j + j + 1)$, and the $(j+1)^{th}$ last one in $\mathbf{s}$ is at position $g_0^j + j$ from the end of the string. Therefore,

$$wt\left(\mathbf{s}_1^{g_0^j + j}\right) - wt\left(\mathbf{s}^d_{n - g^d_{d-j} - j}\right) = j - (j+1) = -1. \quad (36)$$

But note that $wt(\mathbf{s}_1^1) - wt(\mathbf{s}_n^n) = 1$. Consider the function $f(i) = wt(\mathbf{s}_1^i) - wt(\mathbf{s}^n_{n-i+1})$. This function is such that $f(i+1) = f(i) \pm 1$. Therefore, the function must have been zero at some point, contradicting the fact that $\mathbf{s}$ is *imbalanced*. ∎

Remark 10. *Corollary 11 implies that our algorithm uniquely reconstructs the codewords of the codebook* $S_R(n)$ *described in [29] (revisited in Section II-B) without backtracking.*

IV. Reconstruction Code

In this section, we explicitly describe the reconstruction code $S(n)$ (Definition 12) which will consist of all *imbalanced* strings (Definition 6) of length n, beginning with 1, and ending at 0. The design of our reconstruction code is such that we avoid all strings satisfying condition (25) in our codebook. This will ensure that in case of a *pause*, the reconstruction algorithm will know which branch to take. For a string to not be uniquely reconstructable, it must *pause* at some step; therefore, avoiding *pauses* ensures that the string is uniquely reconstructed from its composition multiset. Note that Lemma 7 implies that the reconstruction code $S_R(n)$ (Definition 8) is the reverse of the reconstruction code $S(n)$. We show a bijection between $S(n)$ and positive n-step walks (Definition 7) thereby explicitly describing the code size and propose efficient procedures for mapping information message into this code and then retrieving them. The bounds on the redundancy are provided in Corollary 13. Corollary 11 ensures that the elements of $S(n)$ are uniquely reconstructable by our Reconstruction Algorithm with $O(n^2 \log n)$ complexity and $O(n \log n)$ latency (Theorem 9). Recall that the elements of this codebook $S(n)$ are also reconstructable by the algorithm in [26] without backtracking (Lemma 5). The relevant background for this section is discussed in Section II-B.

Later, we extend $S(n)$ 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 $T(n)$ (Definition 14). This codebook $T(n)$ contains $S(n)$, but also has strings that are not *imbalanced*. The more general sufficient conditions for reconstruction in polynomial time of our algorithm (Proposition 10) ensure that elements of the codebook $T(n)$ can be reconstructed with the same complexity, i.e., in at most $6.5n^2$ field operations (according to Theorem 9). Finally, using the ideas discussed in Remark 8, we propose codebooks $T_1(n)$ (Definition 15), and $T_2(n)$ (Definition 17), through which we give computational bounds on the size of reconstruction codebooks uniquely reconstructable by the reconstruction algorithm with the same complexity as $S(n)$.

Definition 12. *Define* $S(n)$ *to be the set of all imbalanced binary strings of length* n *beginning with* 1*, and ending at* 0*; that is for all prefix-suffix pairs of length* $1 \leqslant j < n$*, one has* $wt(\mathbf{s}_1^j) > wt(\mathbf{s}^n_{n+1-j})$.

The result from [19], discussed in Section II-B as Lemma 7, shows that the codebook $S_R(n)$ (Definition 8) is the reverse of the codebook $S(n)$, i.e. for all $\mathbf{s} \in S(n)$, the reverse string $\mathbf{s}^* \in S_R(n)$ and vice-versa. Recall that $\mathbf{s}$ and $\mathbf{s}^*$ share their composition multisets and cannot belong to the same reconstruction code; therefore $S(n)$ and $S_R(n)$ are disjoint. In [19], the authors show that the elements of the codebook $S_R(n)$, and therefore similarly $S(n)$, are also uniquely reconstructable from the multiset of their prefix-suffix compositions. Theorem 12 gives the ingredients to explicitly encode messages in the codewords of $S(n)$ which we detail in Figure 2.

Theorem 12. *There is a bijection between* $S(n)$ *and positive* n*-step walks (Definition 7).*

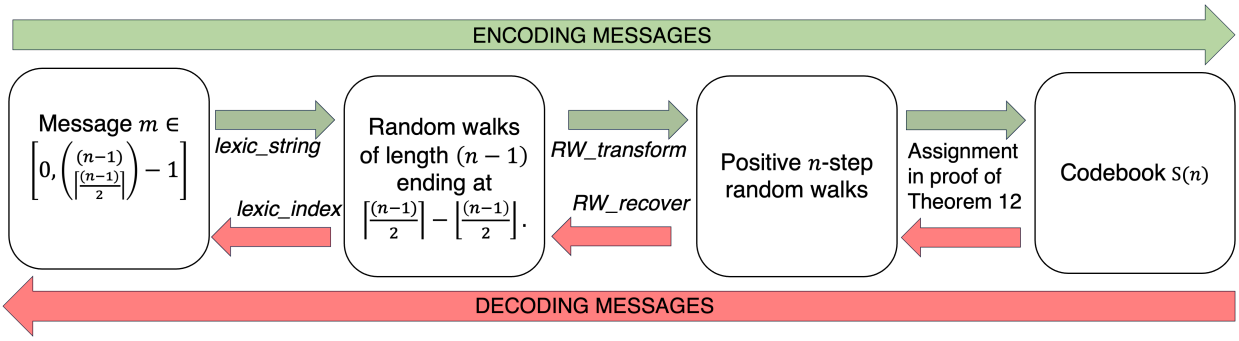

Fig. 2: The procedure of encoding and decoding messages from the codebook $S(n)$.

Proof: Given a binary string $\mathbf{s} = s_1 \dots s_n$, assign X_i's in the following way:

$$X_{2i-1} = \begin{cases} 1, & \text{if } s_i = 1, \\ -1, & \text{if } s_i = 0; \end{cases} \quad \text{and}$$

$$X_{2i} = \begin{cases} -1, & \text{if } s_{n+1-i} = 1, \\ 1, & \text{if } s_{n+1-i} = 0. \end{cases}$$

This assignment is uniquely invertible. That is, for each such $\mathbf{s}$, there is a unique assignment of variables X_j's and vice versa. This ensures that this assignment is injective i.e., different strings are mapped to different random walks. Now note that $S_{2k} = \sum_{i=1}^{2k} X_i = 2(wt(\mathbf{s}_1^k) - wt(\mathbf{s}_{n-k+1}^n))$. As mentioned in Remark 3, if $\mathbf{s}$ is an *imbalanced* string beginning with 1 and ending at 0, then $wt(\mathbf{s}_1^k) - wt(\mathbf{s}_{n-k+1}^n) \geqslant 1$ for all $1 \leqslant k \leqslant \lceil n/2 \rceil$. Therefore, $S_{2k} \geqslant 2$ for all $1 \leqslant k \leqslant \lceil n/2 \rceil$, and $S_{2k+1} \geqslant S_{2k} - 1 \geqslant 1$ for all $1 \leqslant k \leqslant \lfloor n/2 \rfloor$. Note that, for a positive n-step walk, since the k-th step is either -1 or $+1$, the parity of S_k changes after each step. This shows that the assignment of X_j's is surjective, and therefore, also bijective. ∎

The above result along with Lemma 6 and Proposition 8 gives us the following corollary.

Corollary 13. *The size of $S(n)$ is given by $\binom{n-1}{\lceil \frac{n-1}{2} \rceil} \geqslant \frac{2^{n-\frac{1}{2}}}{\sqrt{\pi n}}$. Therefore, redundancy of the reconstruction code $S(n)$ is at most $\lceil 1/2 \log n + 1/2 + 1/2 \log_2 \pi \rceil$.*

Corollary 13 characterizes the size of the codebook $S(n)$, but we still need to design efficient encoding and decoding procedures for mapping and retrieving information messages from the codebook elements. To construct an encoder $\mathcal{E} : \left[0, \binom{n-1}{\lceil \frac{n-1}{2} \rceil} - 1\right] \to S(n)$ and a decoder $\mathcal{D} : S(n) \to \left[0, \binom{n-1}{\lceil \frac{n-1}{2} \rceil} - 1\right]$, we will begin by mapping $[0, \binom{N}{\lceil \frac{n-1}{2} \rceil} - 1]$ to the set of binary strings with length $(n-1)$ and weight $\lceil \frac{n-1}{2} \rceil$. This is equivalent to lexicographically ordering binary strings of length $(n-1)$ with weight $\lceil \frac{n-1}{2} \rceil$. In [34], Kabal uses a coding trellis to give an efficient way of mapping a selection of k items from a given set of N items i.e. mapping binary strings of length N and weight k to $[0, \binom{N}{k} - 1]$. We will call *lexic_string*(N, k, i) the procedure that outputs the i^{th} string in the lexicographic ordering of binary strings of length N and weight k; and *lexic_index*$(N, k, \mathbf{s})$ the procedure that outputs the index i of the string $\mathbf{s}$ in the lexicographic ordering of binary strings of length N and weight k. Treating these binary strings as 1-dimensional random walks with $1 \to 1$, and $0 \to -1$, we have a lexicographic ordering of random walks of length $(n-1)$ ending at $S_{n-1}(\mathbf{X}) = \sum_{k=1}^{n-1} X_k = \left(\lceil \frac{n-1}{2} \rceil - \lfloor \frac{n-1}{2} \rfloor\right)$. In [32, Chapter 3], Feller explicitly describes a bijection between $(n-1)$-step random walks with $S_{n-1}(\mathbf{X}) = \left(\lceil \frac{n-1}{2} \rceil - \lfloor \frac{n-1}{2} \rfloor\right)$, and 1-dimensional *positive* n-step random walks (Definition 7). Given an $(n-1)$-step random walk $\mathbf{X} = (X_1, X_2, \dots, X_{n-1})$ with $S_{n-1}(\mathbf{X}) = \left(\lceil \frac{n-1}{2} \rceil - \lfloor \frac{n-1}{2} \rfloor\right)$, let $m \in \{1, \dots, n-1\}$ be the smallest index such that $S_m(\mathbf{X}) = \sum_{k=1}^{m} X_k \leqslant S_j(\mathbf{X}) = \sum_{k=1}^{j} X_k$ for all $j \in \{1, \dots, n-1\}$ and $j \neq m$. Then

$$\begin{aligned} \mathbf{W} &= (W_1, \dots, W_n) \\ &= (1, X_{m+1}, \dots, X_{n-1}, -X_m, \dots, -X_1) \end{aligned} \tag{37}$$

is a *positive* n-step random walk. We call this procedure of mapping $\mathbf{X} \to \mathbf{W}$ as *RW_transform(X)*. In fact, this mapping is a bijection (see proof of problem 7, [32, Chapter 3]). Note that the n-step random walk $\mathbf{W}$ ends at $\left(1 - 2S_m(\mathbf{X}) + \lceil \frac{n-1}{2} \rceil - \lfloor \frac{n-1}{2} \rfloor\right)$ which lets us recover $S_m(\mathbf{X})$ from this walk. Since m is the smallest index with the minimum $S_m(\mathbf{X})$, the largest index j of this n-step random walk with $S_j(\mathbf{W}) = \sum_{k=1}^{j} W_j = \left(1 - S_m(\mathbf{X}) + \lceil \frac{n-1}{2} \rceil - \lfloor \frac{n-1}{2} \rfloor\right) = 1 + \sum_{k=m+1}^{n-1} X_k$, is such that $j = n - m$. This lets us retrieve the $(n-1)$-step random walk as

$$\begin{aligned} \mathbf{X} &= (X_1, \dots, X_{n-1}) \\ &= (-W_n, \dots, -W_{n-m+1}, W_2, \dots, W_{n-m}) \end{aligned} \tag{38}$$

We call this procedure of mapping $\mathbf{W} \to \mathbf{X}$ as *RW_reco–ver(W)*. This mapping when merged with the assignment of variables in the proof of Theorem 12 can be adapted to give us a procedure to explicitly map, via a bijection, *imbalanced* strings beginning with 1 and ending at 0 to the selection of some $\lfloor \frac{n-1}{2} \rfloor$ objects from $(n-1)$ objects. Therefore, a combination of the procedures described in [32], and [34] can be used to describe an invertible map from $[0, \binom{n-1}{\lfloor \frac{n-1}{2} \rfloor} - 1]$ to the set of *imbalanced* strings beginning with 1, and ending at 0 i.e. $S(n)$. This is demonstrated in Figure 2.

Now, we finally extend our reconstruction code $S(n)$ by expanding codebooks of different sizes in certain specified ways followed by taking a union of them, in order to arrive at a

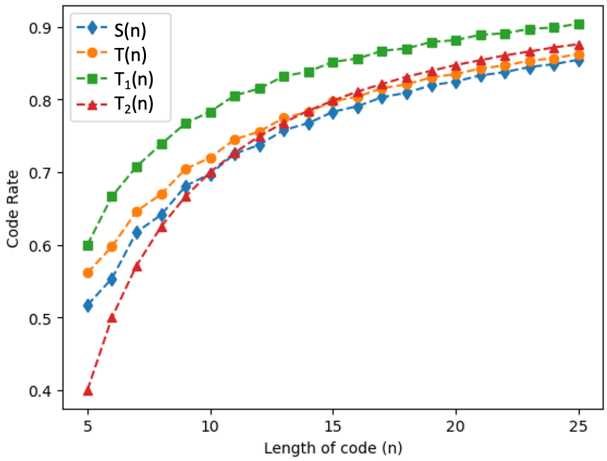

Fig. 3: Comparison of code rates

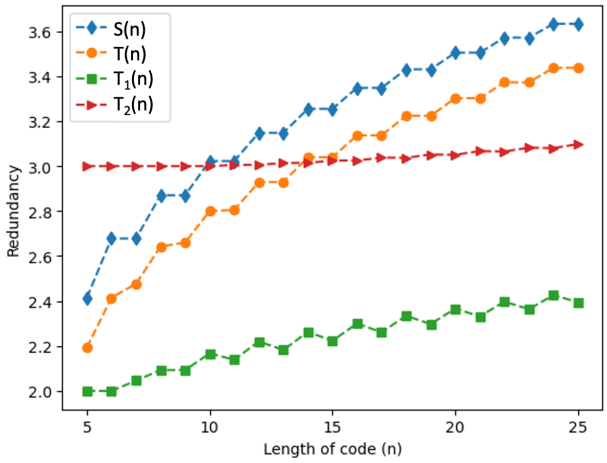

Fig. 4: Comparison of code redundancies

new codebook $T(n)$. We define the following kinds of sets whose construction uses this $S(n)$. The reconstruction code $T(n)$ will be defined as the union of these sets.

Definition 13. *Given a positive integer n, and $2 \leqslant k \leqslant \lfloor n/2 \rfloor$, define $P_{n,k}$ as the set of binary strings of length n which begin at 1, end at 0, as follows:*

$$P_{n,k} = \{\mathbf{s} \in \{0,1\}^n,\ \mathbf{t} \in \{0,1\}^{k-2}, \textit{such that } \mathbf{s}_1^k = 1\mathbf{t}0,\ \mathbf{s}_{n-k+1}^n = 1\mathbf{t}^*0, \textit{ and },\mathbf{s}_{k+1}^{n-k} \in S(n-2k)\}, \tag{39}$$

where $\mathbf{t}^*$ denotes the reverse of the string $\mathbf{t}$.

Proposition 14. *Given a binary string $\mathbf{s} \in P_{n,k}$ of length n, with $2 \leqslant k \leqslant \lfloor n/2 \rfloor$; $\mathbf{s}$ is uniquely reconstructable by our algorithm.*

Proof: We will show that any $\mathbf{s} \in P_{n,k}$ is not a *type-1 string*, and therefore the result will follow from Remark 8. This proof will be similar to the proof of Corollary 11. Consider the function $f(i) = wt(\mathbf{s}_1^i) - wt(\mathbf{s}_{n-i+1}^n)$. Note that,

$$f(i) \begin{cases} = 1, \text{ for } 1 \leqslant i \leqslant k-1; \\ = 0, \text{ for } i = k; \\ > 0, \text{ otherwise.} \end{cases}$$

The first two results follow from the construction of $P_{n,k}$ in Definition 13, and the last inequality follows from Remark 3. As seen in the (36), in the proof of Corollary 11; for every *type-1 string*, there exists a j', such that $f(j') = -1$, implying that $\mathbf{s} \in P_{n,k}$ cannot be a *type-1 string*. ∎

Definition 14. *Define* $T(n) = S(n) \bigcup \left(\bigcup_{k=1}^{\lfloor n/2 \rfloor} P_{n,k} \right)$.

Remark 11. *The extended codebook presented in the ISIT 2022 version of this paper [35] avoided type-2 strings and was shown to be larger than $S(n)$ by a linear factor $41/40$. The codebook defined here avoids type-1 strings and is shown to be larger than $S(n)$ by a linear factor of $9/8$.*

Theorem 15. *Given $\epsilon > 0$, there exists an $N \in \mathbb{N}$ such that for all integers $n > N$ we have*

$$|T(n)| \geqslant (1.125 - \epsilon)\,|S(n)|. \tag{40}$$

Proof: Let $\mathbf{s} \in P_{n,k_1} \cap P_{n,k_2}$, with $k_1 \neq k_2$. Then

$$wt(\mathbf{s}_1^{k_1}) - wt(\mathbf{s}_{n+1-k_1}^n) = 0 \neq wt(\mathbf{s}_1^{k_2}) - wt(\mathbf{s}_{n+1-k_2}^n).$$

This means that $P_{n,k_1} \cap P_{n,k_2} = \varnothing$. Now note that,

$$\frac{|T(n)|}{|S(n)|} = 1 + \sum_{k=2}^{\lfloor n/2 \rfloor} \frac{|P_{n,k}|}{|S(n)|} = 1 + \sum_{k=2}^{\lfloor n/2 \rfloor} 2^{k-2} \frac{\binom{n-1-2k}{\lfloor \frac{n-1}{2} \rfloor - k}}{\binom{n-1}{\lfloor \frac{n-1}{2} \rfloor}}.$$

Setting $n = 2n' + 1$, we see that,

$$\begin{aligned} \frac{|T(n)|}{|S(n)|} &= 1 + \sum_{k=2}^{n'} 2^{k-2} \frac{\binom{2n'-2k}{n'-k}}{\binom{2n'}{n'}} \\ &= 1 + \sum_{k=2}^{n'} 2^{k-2} \left(4^{-k} + \frac{k \cdot 4^{-k}}{2n'} + O(\frac{1}{n'^2}) \right) \\ &= 1 + \frac{2n'(2^{n'} - 3) + 3 \cdot 2^{n'}}{n' \cdot 2^{n'+4}} + O(\frac{1}{n}) \\ &\geqslant \frac{9}{8} + O(\frac{1}{n}). \end{aligned}$$

∎

As we discuss in Remark 8, our algorithm can only confuse a *type-1 string* with a *type-2 string*. We exploit the algebraic conditions in Proposition 10 to find codebooks of larger sizes that can be reconstructed by slightly modifying the proposed Reconstruction Algorithm (without changing the time complexity). In Figure 3 and 4, we compare the rates and redundancies of the reconstruction codes: $S(n)$ (Definition 12), $T(n)$ (Definition 14), $T_1(n)$ which is the codebook formed by excluding all strings of *type-1* (Definition 15), and $T_2(n)$ which is the codebook formed by excluding all strings that are both *type-1* and *type-2* and adding an indicator bit to remaining strings to denote their type (Definition 17). Given a positive integer $n > 3$, we define $T_1(n)$ to be the set of all binary strings of length n beginning with 1, and ending with 0 with no *type-1 strings*; that is all strings satisfying condition (25) for any $1 \leqslant j \leqslant d$ are removed from the set of

strings being considered.

Definition 15. *Given a positive integer $n > 3$, define $T_1(n)$ as a set of binary strings of length n which begin at 1, end at 0, as follows,*

$$T_1(n) \stackrel{\text{def}}{=} \{\, \mathbf{s} \in \{0,1\}^n \mid s_1 = 1,\ s_n = 0,\ \textit{and} \nexists\, j \in \{1,2,\ldots,\lfloor wt(\mathbf{s})/2 \rfloor\} \textit{ such that } a_j \geqslant 1 \textit{ and } g_0^j(\mathbf{s}) - g_{wt(\mathbf{s})-j}^{wt(\mathbf{s})}(\mathbf{s}) = 1 \,\}.$$

The set $T_1(n)$ contains strings which are either only *type-2*, or neither of the types. Therefore for each element in $T_1(n)$, our algorithm even in case of a *pause* knows exactly which branch to take (the branch satisfying condition (26)). Therefore, it uniquely reconstructs the string without backtracking. In the proof of Proposition 14, it is shown that all codewords of the codebook $T(n)$ are such that they are not *type-1* strings. This means that $T(n) \subseteq T_1(n)$. Extending this argument further, we define $S_{12}(n)$ to be the set of all binary strings of length n beginning with 1, and ending with 0 with no strings that are both *type-1* and *type-2*.

Definition 16. *Given a positive integer $n > 3$, define $S_{12}(n)$ as a set of binary strings of length n which begin at 1, end at 0, as follows,*

$$S_{12}(n) \stackrel{\text{def}}{=} \{\, \mathbf{s} \in \{0,1\}^n,\ s_1 = 1,\ s_n = 0, \nexists\, j \in \{1,2,\ldots,\lfloor wt(\mathbf{s})/2 \rfloor\} \textit{ such that } a_j \geqslant 1 \textit{ and } g_0^j(\mathbf{s}) - g_{wt(\mathbf{s})-j}^{wt(\mathbf{s})}(\mathbf{s}) = 1, \nexists\, k \in \{1,2,\ldots,\lfloor wt(\mathbf{s})/2 \rfloor\} \textit{ such that } a_{d-k} \geqslant a_d + 1 \textit{ and } g_{wt(\mathbf{s})-k}^{wt(\mathbf{s})} - g_0^k = a_d + 1 \,\}.$$

This means that the set $S_{12}(n)$ contains strings which are either only *type-1*, only *type-2*, or neither of the types. Note that, for our algorithm to know which branch to take, we will need to add an extra bit of redundancy, an indicator bit, to the elements of $S_{12}(n)$. This bit will indicate if the string being considered is *type-2* or not. If the added bit is 1, in case of a *pause*, our algorithm will know that the string is *type-2*, and take the branch corresponding to condition (26). If the added bit is 0, in case of a *pause*, our algorithm will know that the string is *type-1*, and take the branch corresponding to condition (25), or continue without backtracking in the case of no *pauses*. We define $T_2(n+1)$ to be the codebook of length $(n+1)$ where the codebook is formed by adding this indicator bit to the elements of $S_{12}(n)$.

Definition 17. *Given a positive integer $n > 3$, The code $T_2(n+1)$ is defined as the codebook of length $(n+1)$ where the codebook is formed by adding an indicator bit to the elements of $S_{12}(n)$.*

The set $S_{12}(n)$ by definition contains strings which are either only *type-1*, *type-2*, or neither of the types. But this means that the codebook $T_1(n)$ is a subset of the set $S_{12}(n)$ which implies $|T_1(n)| < |S_{12}(n)| = |T_2(n+1)|$. Therefore, we have the following relationship between the sizes of the proposed codebooks, also represented in Figure 1:

$$|S(n)| < |T(n)| < |T_1(n)| < |T_2(n+1)|. \tag{41}$$

Remark 12. *In Figure 3 and 4, we present the code rates and code redundancies of the reconstruction codebooks $S(n)$ (Definition 12), T(n) (Definition 14), and the codebooks $T_1(n)$ (Definition 15), and $T_2(n)$ (Definition 17) as described above. The time complexity for constructing $T_1(n)$ and $T_2(n)$ is $O(n \cdot 2^n)$, and therefore the results are presented only for $n \leqslant 25$.*

V. Conclusion.

Motivated by the problem of recovering polymer strings from their fragmented ions during mass spectrometry, we introduce a new algorithm to reconstruct a binary string from the multiset of its substring compositions. We further characterize algebraic properties of binary strings that guarantee reconstruction without backtracking thereby enlarging the space of binary strings uniquely reconstructable without backtracking compared with previously known algorithms. Additionally, we modify and extend the reconstruction code proposed in [30] to produce a new reconstruction code which is linearly larger in size, and is uniquely reconstructable by our algorithm without backtracking.

There are several combinatorial and coding-theoretic problems related to string reconstruction from substring composition that remain open. The problems of bounding the size of *reconstruction codes* as well as constructing explicit schemes with minimum redundancy remain open. Our algorithm expands the conditions for strings to be uniquely reconstructed without backtracking, and therefore characterizing the set of strings uniquely reconstructable by the algorithm in this paper is a possible step in that direction. As seen from results in Figure 4, we believe that there exist reconstruction codes with constant redundancy that can be reconstructed efficiently. Furthermore, deriving bounds on time complexity of algorithms for reconstructing strings from their substring multiset is another problem of interest.

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] Statista, "Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025," 2022. [Online]. Available: https://www.https://www.statista.com/statistics/871513/worldwide-data-created/

[3] M. Hilbert and P. López, "The world's technological capacity to store, communicate, and compute information," *science*, vol. 332, no. 6025, pp. 60–65, 2011.

[4] 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.

[5] 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.

[6] G. D. Dickinson, G. M. Mortuza, W. Clay, L. Piantanida, C. M. Green, C. Watson, E. J. Hayden, T. Andersen, W. Kuang, E. Graugnard *et al.*, "An alternative approach to nucleic acid memory," *Nature communications*, vol. 12, no. 1, p. 2371, 2021.

[7] S. D. Dahlhauser, S. R. Moor, M. S. Vera, J. T. York, P. Ngo, A. J. Boley, J. N. Coronado, Z. B. Simpson, and E. V. Anslyn, "Efficient molecular encoding in multifunctional self-immolative urethanes," *Cell Reports Physical Science*, vol. 2, no. 4, p. 100393, 2021.

[8] 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.
[9] 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.
[10] 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.
[11] Y. Erlich and D. Zielinski, "Dna fountain enables a robust and efficient storage architecture," *science*, vol. 355, no. 6328, pp. 950–954, 2017.
[12] 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.
[13] 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.
[14] 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.
[15] 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.
[16] M. Warren, "Move over, dna: ancient proteins are starting to reveal humanity's history," *Nature*, vol. 570, no. 7762, pp. 433–437, 2019.
[17] T. E. Creighton, *Proteins: structures and molecular properties*. Macmillan, 1993.
[18] R. Gabrys, S. Pattabiraman, and O. Milenkovic, "Reconstruction of sets of strings from prefix/suffix compositions," *IEEE Transactions on Communications*, vol. 71, no. 1, pp. 3–12, 2023.
[19] Z. Ye and O. Elishco, "Reconstruction of a single string from a part of its composition multiset," *IEEE Transactions on Information Theory*, vol. 70, no. 6, pp. 3922–3940, 2024.
[20] S. Marcovich and E. Yaakobi, "Reconstruction of strings from their substrings spectrum," *IEEE Transactions on Information Theory*, vol. 67, no. 7, pp. 4369–4384, 2021.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] 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.
[26] J. Acharya, H. Das, O. Milenkovic, A. Orlitsky, and S. Pan, "String reconstruction from substring compositions," *SIAM Journal on Discrete Mathematics*, vol. 29, no. 3, pp. 1340–1371, 2015.
[27] 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.
[28] 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.
[29] S. Pattabiraman, R. Gabrys, and O. Milenkovic, "Coding for polymer-based data storage," *IEEE Transactions on Information Theory*, vol. 69, no. 8, pp. 4812–4836, 2023.
[30] ——, "Reconstruction and error-correction codes for polymer-based data storage," in *2019 IEEE Information Theory Workshop (ITW)*. IEEE, 2019, pp. 1–5.
[31] 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.
[32] W. Feller, "An introduction to probability theory and its applications," *1, 2nd*, 1967.
[33] F. P. (https://mathoverflow.net/users/4312/fedor petrov), "Upper limit on the central binomial coefficient," MathOverflow, uRL:https://mathoverflow.net/q/380124 (version: 2021-01-02). [Online]. Available: https://mathoverflow.net/q/380124
[34] P. Kabal, "Combinatorial coding and lexicographic ordering," 2018.
[35] U. Gupta and H. Mahdavifar, "A new algebraic approach for string reconstruction from substring compositions," in *2022 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2022, pp. 354–359.

PLACE PHOTO HERE

Utkarsh Gupta (Student Member, IEEE) received the B.Tech degree in Mathematics and Computing from Indian Institute of Technology Delhi, New Delhi, India and the M.Sc. degree in Electrical and Computer Engineering from the University of Michigan, Ann Arbor, in 2021 and 2022, respectively. He is currently pursuing the Ph.D. degree in Electrical & Computer Engineering at Northeastern University. His research interests include coding theory, security, and privacy.

PLACE PHOTO HERE

Hessam Mahdavifar (Member, IEEE) is an Associate Professor in the Department of Electrical and Computer Engineering at Northeastern University and an Adjunct Associate Professor in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. He received the B.Sc. degree from the Sharif University of Technology, Tehran, Iran, in 2007, and the M.Sc. and the Ph.D. degrees from the University of California San Diego (UCSD), La Jolla, in 2009, and 2012, respectively, all in electrical engineering. He was with the University of Michigan, first as an Assistant Professor and later as an Associate Professor, between 2017 and 2023. Before that, he was with the Samsung US R&D between 2012 and 2016, in San Diego, US, as a staff research engineer.

He received the NSF career award in 2020. He also received Best Paper Award in 2015 IEEE International Conference on RFID, and the 2013 Samsung Best Paper Award. He also received two Silver Medals at the International Mathematical Olympiad in 2002 and 2003, and two Gold Medals at Iran National Mathematical Olympiad in 2001 and 2002. His main area of research is coding and information theory with applications to wireless communications, storage systems, security, and privacy.