Adaptive Exact Learning in a Mixed-Up World: Dealing with Periodicity, Errors and Jumbled-Index Queries in String Reconstruction

Ramtin Afshar[1], Amihood Amir[2], Michael T. Goodrich[1], and Pedro Matias[1](✉)

[1] Department of Computer Science, University of California Irvine, Irvine, USA
{afsharr,goodrich,pmatias}@uci.edu
[2] Department of Computer Science, Bar Ilan University, Ramat Gan, Israel
amir@cs.biu.ac.il

Abstract. We study the query complexity of exactly reconstructing a string from adaptive queries, such as substring, subsequence, and jumbled-index queries. Such problems have applications, e.g., in computational biology. We provide a number of new and improved bounds for exact string reconstruction for settings where either the string or the queries are "mixed-up".

Keywords: Exact learning · String reconstruction · Jumbled-index queries · Periodicity · DNA sequencing · Stringology · Substrings · Hybridization · Information security

1 Introduction

Exact learning involves asking a series of queries so as to learn a configuration or concept uniquely and without errors, e.g., see [12]. For example, imagine a game where a player, Alice, is trying to exactly learn a secret string, S, such as $S =$ `"rumpelstiltskin"`, which is known only to a magic fairy. Alice may ask the fairy questions about S, but only if they are in a form allowed by the fairy, such as "Is X a substring of S?". Any allowable question that Alice asks must be answered truthfully by the fairy. Alice's goal is to learn S by asking the fewest number of allowable questions. Her strategy is ***adaptive*** if her questions can depend on the answers to previous queries. This exact-learning string-reconstruction problem might at first seem like a contrived game, but it actually has a number of applications. For instance, in interactive DNA sequencing, the fairy's string is an unknown DNA sequence, S, and allowable queries are "Is X a substring of S?" Each such question can be answered by a hybridization experiment that exposes copies of S to a mixture containing specific primers to see

The full version of this paper is available in [5].

C. Boucher and S. V. Thankachan (Eds.): SPIRE 2020, LNCS 12303, pp. 155–174, 2020.
https://doi.org/10.1007/978-3-030-59212-7_12

which ones bind to S, e.g., see [73]. Thus, we are interested in the exact-learning complexity of adaptively learning an unknown string via queries of various given types, that is, for exactly reconstructing a string from queries. Formally, we are interested in minimizing a ***query-complexity*** measure, $Q(n)$, which, in our case, is the number of queries of certain types needed in order to exactly learn a string, S. This query-complexity concept comes from machine-learning and complexity theory, e.g., see [3,12,18,25,32,76,83].

1.1 Related Work

Motivated by DNA sequencing, Skiena and Sundaram [73] were the first to study exact string reconstruction from adaptive queries. For ***substring queries***, of the form "Is X a substring of S?", they give a bound for $Q(n)$ of $(\sigma-1)n+2\log n+O(\sigma)$, where σ is the alphabet size. For ***subsequence queries***, of the form "Is X a subsequence of S?", they prove a bound for $Q(n)$ of $\Theta(n\log\sigma+\sigma\log n)$. Recently, Iwama *et al.* [44] study the problem for binary alphabets, which removes the additive logarithmic term in this case. These papers do not consider "mixed-up" strings, however, such as strings that are periodic or periodic with errors. The abundance of repetitions and periodic runs in genomic sequences is well known and has been exploited in the last decades for biologic and medical information (see e.g. [15,16,30,33,35,53,65,66,74,82]). It is somewhat surprising that this phenomenon has not been used to achieve more efficient algorithms. Margaritis and Skiena [60] study a parallel version of exact string reconstruction from queries, which are hybrids of adaptive and non-adaptive strategies, showing, e.g., that a length-n string can be reconstructed in $O(\log^2 n)$ rounds using n substring queries per round. Tsur [77] gives a polynomial approximation algorithm for the 1-round case. As in [73], these papers do not consider bounds for $Q(n)$ based on properties of the string such as its periodicity. Cleve *et al.* [28] study string reconstruction in a quantum-computing model, showing, for example, that a sublinear number of queries are sufficient for a binary alphabet. This result does not seem to carry over to a classical computing model, however, which is the subject of our paper.

Another type of query we consider is the ***jumbled (or histogram)-index*** query, first considered in [20,21,26,37] and studied more recently in, e.g. [4,7,9,10,52,62]. Jumbled indexing has many applications. It can be used as a tool for de novo peptide identification (as in e.g. [45,50,51]), and has been used as a filter for searching an image database [27,31,75,81,85]. In this query, which has received much study of late, but has not been studied before for adaptive string reconstruction, one is given a Parikh vector, i.e., a vector of frequency counts for each character in an alphabet, and asked if there is a substring of the reference string, S, having these frequency counts and, if so, where it occurs in S. Such reconstruction may aid in narrowing down peptide identification, or focusing on image retrieval.

Another model for string reconstruction, tangential to ours and studied extensively, is the one defined by a non-adaptive oracle, e.g., see [1,2,13,14,19–22,24,26,29,34,36–38,40–43,47–49,54,56,58,59,63,64,67–72,78,79,84]. In this model we are given a set of answers to queries in advance, and we aim to understand sufficient and necessary conditions on the answers that enable the exact reconstruction of the string. This model differs from the adaptive one considered in this paper in that it focuses on the study of combinatorial properties of strings, rather than on minimizing the number of queries. We review existing literature for non-adaptive string reconstruction in more detail in the full version of the paper [5].

1.2 Our Results

We provide new and improved results for exactly reconstructing strings from adaptive substring, subsequence, and jumbled-index queries. For example, we believe we are the first to characterize query complexities for exactly reconstructing periodic strings from adaptive queries, including the following results for reconstructing a length-n periodic (i.e., "mixed-up") string, $S = p^k p'$, of smallest period p, where p' is a prefix of p and the alphabet has size σ:

- It requires at least $|p| \lg \sigma$ substring or subsequence queries.
- It can be done with $\sigma|p| + \lceil \lg |p| \rceil$ substring queries, if n is known.
- It can be done with $O(\sigma|p| + \lg n)$ substring queries, if n is unknown.
- It can be done with $\sigma\lceil \lg n \rceil + 2|p|\lceil \lg \sigma \rceil$ subsequence queries, for known n.
- It can be done with $2\sigma\lceil \lg n \rceil + 2|p|\lceil \lg \sigma \rceil$ subsequence queries, if n is unknown.

Perhaps our most technical result is that we show that we can reconstruct a length-n string, S, within Hamming distance d of a periodic string $S' = p^k p'$, of smallest period p, using $O(\min(\sigma n,\, d\sigma|p| + d|p| \lg \frac{n}{d+1}))$ substring queries, if n is unknown. We also show that we can exactly reconstruct a general length-n string, S, using $2\sigma\lceil \lg n \rceil + n\lceil \lg \sigma \rceil$ subsequence queries, if n is unknown. Such queries are another "mixed-up" setting, since there can be multiple subsequence matches for a given string. Our bound improves the previous best, decades-old result, by Skiena and Sundaram [73], who prove a query complexity of $2\sigma \lg n + 1.59 n \lg \sigma + 5\sigma$ for this case. If n is known, then $\sigma\lceil \lg n \rceil + n\lceil \lg \sigma \rceil$ subsequence queries suffice. We believe we are the first to study string reconstruction using jumbled-index queries, which are yet another "mixed-up" setting, since they simply count the frequency of each character occurring in a substring. We prove the following results:

- We can reconstruct a length-n string with $O(\sigma n)$ yes/no extended jumbled-index queries, which include a count for an end-of-string character, \$.
- For jumbled-index queries that return an index of a matching substring, string reconstruction is not possible if this index is chosen adversarially, but is possible using $O(\sigma + n \lg n)$ queries if it is chosen uniformly at random.

1.3 Preliminaries

We consider strings over the alphabet $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$ of σ letters. The size of a string X is denoted by $|X|$. We use $X[i]$ to denote the i^{th} letter of X and

$X[i..j]$ to refer to the substring of X starting at its i^{th} and ending at its j^{th} letter (e.g., $X = X[1..|X|]$). We may ignore i when expressing a prefix $X[..j]$ of X. Similarly, $X[i..]$ is a suffix of X. Occasionally, we will express concatenation of strings X and Y by $X \cdot Y$ (instead of XY) to emphasize some property of the string. A string X concatenated with itself k (resp. infinitely many) times can be expressed as X^k (resp. X^∞). The reversal of a string X is denoted by X^R.

A string, S, has ***period*** p if $S = p^k p'$, such that $k > 0$ is an integer and p' is a (possibly empty) prefix of p. Further, a string S is ***periodic*** if it has a period that repeats at least twice, i.e. $S = p^k p'$ and $k > 1$[1]. The following is a well known result concerning the periodicity of a string, due to Fine and Wilf [39], which we will need later on.

Lemma 1 (Periodicity Lemma [39]). *If p, q are periods of a string X of length $|X| \geq |p| + |q| - \gcd(|p|, |q|)$, then X also has a period of size $\gcd(|p|, |q|)$.*

A ***doubling search*** is the operation used to determine a number n from a (typically unbounded) range of possibilities. It involves doubling a query value, m, until it is greater than n, followed by a binary search to determine n itself. Its time complexity is $2\lfloor \lg n \rfloor + 1$[2].

Due to space constraints, we defer proofs of Lemmas and Theorems marked with ⊛ to the full version of the paper [5], where we also include pseudo-code for our algorithms.

2 Substring Queries

In this section, we study query complexities for a string, S, subject to yes/no ***substring*** queries, IsSubstr, i.e. queries of "Is X a substring of S?". We focus on the cases where S corresponds to an originally periodic string, that may have lost its periodicity property due to error corruption. The nature of the errors is context-dependent. For example, corruption may be caused by transmission errors or measurement errors.

There are multiple ways to model errors in strings (see [8,11,23,46,55,57,80]). In this paper, we consider Hamming distance. We say that S is a d-***corrupted periodic string*** if there exists a periodic string S' of period p, such that $|S| = |S'|$ and $\delta(S', S) \leq d$, where δ is the Hamming distance. We refer to p as an ***approximate period*** of S. Notice that, depending on d, there might exist multiple possible strings S' that originate S.

Our main result in this section is the following.

[1] Our algorithms assume that S is periodic ($k > 1$), while the Periodicity Lemma (1) only requires a string to have a period ($k > 0$).

[2] A more sophisticated version of this procedure exists (see [17]) that actually improves the constant in the time complexity, but for simplicity, we use the traditional algorithm, which is asymptotically equivalent.

Theorem 1. *We can reconstruct a length-n d-corrupted periodic string S using*

$$O\left(\min\left(\sigma n, d\sigma|p| + d|p|\lg\frac{n}{d+1}\right)\right) \text{ queries,}$$

for known d, unknown $|p|$, regardless of whether we know n, where p is a smallest approximate period of S.

The algorithm of Theorem 1 is a more elaborate version of a reconstruction algorithm for the special case of $d = 0$, i.e. when no errors occurred and $S = S'$, and when n is not known in advance.

Theorem 2. *We can reconstruct a length-n periodic string, $S = p^k p'$, of smallest period p, using $O(\sigma|p| + \lg n)$ substring queries, assuming both n and $|p|$ are unknown in advance.*

The algorithm of Theorem 2, in turn, builds from a simple reconstruction algorithm that handles the case where n is known in advance and $d = 0$.

For clarity, we will present our results in increasing order of complexity, from the least general result of $d = 0$ and known n, to the most general result of arbitrary d and unknown n.

2.1 Uncorrupted Periodic Strings of Known Size

We first give a simple algorithm to reconstruct a periodic string $S = p^k p'$ of smallest period p and known size with query complexity $O(\sigma|p|)$, and then show how to improve this algorithm to have query complexity $\sigma|p|$ plus lower-order terms. Our algorithms use a primitive developed by Skiena and Sundaram [73], which we call "***append*** (resp., ***prepend***) a letter." In the append (resp., prepend) primitive, we start with a known substring q of S, and we ask queries $\mathsf{IsSubstr}(qa_i)$ (resp., $\mathsf{IsSubstr}(a_i q)$), for each $a_i \in \Sigma$. Note that if we know that one of the qa_i (resp., $a_i q$) strings must be a substring, we can save one query, so that appending or prepending a letter uses at most $\sigma - 1$ queries in this case.

In our simple algorithm[3], we iteratively grow a candidate period, q, using the append primitive until $q^{g(q)-1}$ is a substring, where $g(x) = \lfloor n/|x| \rfloor$. Notice that q may be an "unlucky" cyclic rotation of p, which only repeats $g(p) - 1$ times, and we need to account for this possibility. Thus, once we get a substring corresponding to $q^{g(q)-1}$, we then append/prepend letters until we recover all of S.

Theorem 3. ⊛ *We can reconstruct a length-n periodic string $S = p^k p'$, of smallest period p, using $O(\sigma|p|)$ substring queries, assuming n is known in advance and $|p|$ is unknown.*

[3] Pseudo-code can be found in the full version of the paper [5], where the number of queries is also shown for each step involving queries.

With a little more effort, we can improve the constant factor in the query complexity, by showing that, for $k = \lfloor n/|p| \rfloor > 3$, the following implication holds: if $q^{g(q)-1}$ is a substring, then q must be a cyclic rotation of p.

Theorem 4. ⊛ *We can reconstruct a length-n periodic string $S = p^k p'$, of smallest period p, using at most $\sigma|p| + \lceil \lg |p| \rceil$ substring queries, assuming that: n is known in advance, $k > 3$ and $|p|$ is unknown.*

Notice that any reconstruction algorithm requires at least $|p| \lg \sigma$ queries.

Theorem 5. *Reconstructing a length-n string, $S = p^k p'$, of smallest period p, requires at least $|p| \lg \sigma$ IsSubstr queries, even if n and $|p|$ are known.*

Proof. There are $\sigma^{|p|}$ possible periods for S. Since each period corresponds to a different output of a reconstruction algorithm, A, and each query is binary, we can model any such algorithm, A, as a binary decision tree, where each internal node corresponds to an IsSubstr query. Each of the $\sigma^{|p|}$ possible periods must correspond to at least one leaf of A; hence, the minimum height of A is $\lg(\sigma^{|p|})$. □

2.2 Uncorrupted Periodic Strings of Unknown Size

As in Sect. 2.1, we iteratively grow a candidate period q and attempt to recover S by concatenating q with itself in the appropriate way. The difficulty when n is unknown is that we can no longer confidently predict $g(q)$. Thus, we can no longer issue a single query to test if q is the right period. An immediate solution is to use a doubling search. Unfortunately, this introduces a multiplicative $O(\lg n)$ term into the query complexity. To avoid it, we show how we can take advantage of the Periodicity Lemma (1) to amortize the extra work needed to recover S.

Let us describe the algorithm[4]. We start with an empty candidate period q. At each iteration, we add a letter to q, using the append primitive and, using a doubling search, determine the ***run-length*** t of q, i.e. the maximum integer t such that q^t is a substring of S. If $t = 1$, we advance to the next iteration and repeat this process. If, on the other hand, $t > 1$, we use q to determine the largest substring T that has a period of size $|q|$. This can be done efficiently, using doubling searches, by determining the largest suffix l of q and the largest prefix r of q, such that $\mathsf{IsSubstr}(l \cdot q^t \cdot r)$. Once T is determined, we check whether it corresponds to S by checking if there is any letter preceding and succeeding T. If T corresponds to S, we output it. Otherwise, we update q to be any largest substring of T whose size is assuredly less than $|p|$: using Periodicity Lemma (1), we argue in Lemma 2 below that, if q is not a cyclic rotation of p, then p must be as large as *almost* the entire substring T; more specifically, it must be the case that $|p| > |T| - |q| + 1$. Thus, we update q to be a length-$(|T| - |q| + 1)$ prefix of T (any other substring of T would also work). We use this fact to get

[4] Pseudo-code can be found in the full version of the paper [5], where the number of queries is also shown for each step involving queries.

a faster convergence to a cyclic rotation of p, while making sure that we do not overshoot $|p|$. Indeed, this observation will enable us to incur a $O(\lg n)$ additive factor, instead of a multiplicative one. After updating q, we advance to the next iteration, where a new letter is appended to q, and repeat this process until $T = S$.

Lemma 2. *Let T be the largest proper substring of $S = p^k p'$, of smallest period p, such that: $|q|$ is the length of the smallest period of T. Then, $|p| > |T| - |q| + 1$.*

Proof. Let us assume, by contradiction, that $|p| \leq |T| - |q| + 1$. Then, $|T| \geq |q| + |p| - 1$ and, thus, $|T| \geq |q| + |p| - \gcd(|q|, |p|)$. In addition, if p is a period of S, then T must have a period of size $|p|$. So, by the Periodicity Lemma (1), T also has a period of size $\gcd(|q|, |p|)$. Moreover, since T is the largest proper substring of S, $|p|$ is not a multiple of $|q|$. Therefore, T must have a period shorter than $|q|$, a contradiction. □

Let $q_1, q_2, \ldots, q_m$ be the sequence of m candidate periods of increasing length, each of which is the result of the append/prepend primitive at the beginning of every iteration, e.g. $|q_1| = 1$. In addition, let us use t_i to denote the run-length of q_i. Correctness of our algorithm follows from the following two lemmas.

Lemma 3. *The algorithm successfully returns $S = p^k p'$, of smallest period p, if there exists an iteration $i \in \{1, 2, \ldots, m\}$, such that q_i is a cyclic rotation of p.*

Proof. If $t_i > 1$, then it is easy to see that the string T computed at iteration i, must correspond to S. If $f_i = 1$, then the algorithm essentially switches to the letter-by-letter algorithm, appending or prepending letters until the end, when $q_m = S$. □

Lemma 4. *There exists an iteration $i \in \{1, 2, \ldots, m\}$, such that q_i is a cyclic rotation of p.*

Proof. Let us assume that there is no such iteration i. Then, since all the q_i's are increasing in length, it must be the case that there exists an iteration $j \in \{1, 2, \ldots, m-1\}$, such that: $|q_j| < |p|$, but $|q_{j+1}| > |p|$. However, it follows from Lemma 2 (when $f_t > 1$) and the fact that we add a single letter to q_j (when $f_t = 1$) that p must be at least as large as q_{j+1}, a contradiction. □

The following lemma shows that we can charge the logarithmic factors, incurred in each iteration j, to the work that would have been required to find the letters introduced in q_{j+1}. This establishes the amortization in query complexity.

Lemma 5. ⊛ *The number of queries performed in the j^{th} iteration is at most $\sigma(|q_{j+1}| - |q_j|) + O(\sigma)$, for $j < m$, or $O(\sigma + \lg n)$, for $j = m$.*

Theorem 2 follows from Lemmas 3 to 5. A detailed proof can be found in the full version of the paper [5].

2.3 Corrupted Periodic Strings

Let us assume throughout the remainder of this section that S is a d-corrupted periodic string of approximate period p. Again, the main idea of the algorithm described in this section consists of: (1) determining a cyclic rotation of a true period (in this case, there might be multiple true periods), by iteratively growing a candidate period q, and (2) using q to recover S accordingly. However, in the presence of errors, each of these steps becomes more difficult to realize efficiently. For example, in the first step, we might be growing a candidate period q that includes an error. So, in order to rightfully reject the hypothesis that q is at most as large as some approximate period p, our algorithm should be able to tell the difference between (i) $|p| = |q|$ and q includes an error and (ii) $|p| > |q|$. Otherwise, the algorithm will keep on growing q until it is equal to S, possibly incurring σn queries. In addition, the second step of using q to determine S requires more work, since the presence of errors discards the possibility of simply concatenating q with itself the required number of times. Because of these issues, it is crucial that our algorithm understands when a candidate period is or not free of errors. Thus, the algorithm relies on the following.

Lemma 6. *Let A be any length-$(2d+1)|p|$ substring of a d-corrupted periodic string S of approximate period p, corresponding to the concatenation of length-$|p|$ substrings $q_1, q_2, \ldots, q_{2d+1}$. Then, a cyclic rotation of p must be the only substring q_j appearing at least $d+1$ times in $q_1, q_2, \ldots, q_{2d+1}$.*

Proof. Clearly, there is some q_i that is a cyclic rotation of p. Moreover, there is some q_j that appears at least $d+1$ times in $q_1, q_2, \ldots, q_{2d+1}$, or the number of errors would exceed d, by the pigeonhole principle. If $i \neq j$, then each occurrence of q_j, contributes at least 1 error, resulting in at least $d+1$ errors, a contradiction. Finally, q_j must be the only string with $d+1$ appearances in $q_1, q_2, \ldots, q_{2d+1}$, by the pigeonhole principle. $\square$

Let us give the details for our algorithm[5], which is able to recover S, even when its size n is unknown. We maintain an initially empty substring, A, of S, by extending it with $2d+1$ letters in each iteration, using the append and prepend primitives (as described in Sect. 2.1), potentially incurring an extra σ queries for detecting a left or right endpoint of S. In the case that $n = |S| < |p|(2d+1)$, the last iteration requires only $\min(2d+1, |S| - |A|)$ new letters. Thus, after adding letters to A in the i^{th} iteration, A is a substring of S of size at most $i(2d+1)$. Before advancing to the next iteration, we determine the only possible length-i candidate period q that could have originated A with at most d errors (by Lemma 6). At this point we do not know if some approximate period p has size $|p| = i$, so we try to use q to recover the rest of the string, halting whenever the total number of errors exceeds d, in which case we advance to the next iteration and repeat this process for a new candidate period of size $i+1$. This logic is in

[5] Pseudo-code can be found in the full version of the paper [5], where the number of queries is also shown for each step involving queries.

the subroutine $\mathsf{Expand}(q)$, described next(See footnote 5). It initializes a string T to q and expands it by doing the following at each iteration:

1. Appending to T the largest periodic substring of period $\overrightarrow{q}$, where $\overrightarrow{q}$ is the appropriate cyclic rotation of q that aligns with the right-endpoint of T. This can be done efficiently by determining the maximum value of x, using a doubling search, for which

$$\mathsf{IsSubstr}(T \cdot (\overrightarrow{q}^{\infty}[..\, x])),$$

   incurring $2\lfloor \lg x \rfloor + 1$ queries. The cyclic rotation $\overrightarrow{q}$ can be determined with no additional queries, by maintaining the value x', which is the value of x in the previous iteration, i.e. $\overrightarrow{q}$ is the cyclic rotation of q starting at the index $(x' \mod |q| + 2)$ of q.
2. Prepending to T the largest periodic substring of period $\overleftarrow{q}$, where $\overleftarrow{q}$ is the appropriate cyclic rotation of q that aligns with the left-endpoint of T. This can be done efficiently by determining the maximum value of y, using a doubling search, for which

$$\mathsf{IsSubstr}(((\overleftarrow{q}^{R})^{\infty}[..\, y])^{R} \cdot T),$$

   incurring $2\lfloor \lg y \rfloor + 1$ queries. The cyclic rotation $\overleftarrow{q}$ can be determined with no additional queries in a similar fashion to $\overrightarrow{q}$.
3. Determining, if they exist, the letters immediately to the left and to the right of T, using 2σ queries, and adding them to T.

The expansion process in $\mathsf{Expand}(q)$ halts when either the total number of errors with respect to q, $\delta(T, q^{\infty}[..|T|])$, exceeds d (in which case we advance to the next iteration), or when $T = S$ (in which case we return T).

Remark 1. $\mathsf{Expand}(q)$ successfully returns S if and only if q is a cyclic rotation of some approximate period.

Lemma 7. *The number of queries performed during any call to* Expand *is* $O(d\sigma + d \lg \frac{n}{d+1})$.

Proof. Each call to Expand uses at most $2(d+1)\sigma$ queries to determine the corrupted letters, as well as the left/right endpoints of S – the total number of iterations of the while loop in Expand is $d+1$, since every iteration except the last introduces at least 2 errors in T, and each iteration incurs 2σ queries.

In addition, the number of queries used by $\mathsf{Expand}(q)$ during the doubling searches is

$$\sum_{j=1}^{|q|} \left(2\lfloor \lg x_j \rfloor + 2\lfloor \lg y_j \rfloor + 2\right),$$

where x_j and y_j denote, respectively, the lengths of the substrings determined via doubling searches in steps 1 and 2, during the j^{th} call to Expand. Since the total number of iterations is $d+1$, there is at most $d+2$ such x_j's and y_j's. Moreover, the above summation is maximized when all the x_j's and y_j's have the same average value of at most $(n-d)/(d+1)$. This follows from Jensen's inequality and concavity of log. Thus, the overall time complexity is

$$O\left(d\sigma + d\lg\frac{n}{d+1}\right).$$

□

Correctness and query complexity of our algorithm follows from Remark 1 and Lemmas 6 and 7, giving us:

Theorem 6. ⊛ *We can reconstruct a length-n d-corrupted periodic string S using $O(d\sigma|p| + d|p|\lg\frac{n}{d+1})$ queries, for known d, unknown $|p|$, regardless of whether we know n, where p is a smallest approximate period of S.*

If n is known, we could save the queries used to check the left and right endpoints of S, but this does not alter the query complexity asymptotically.

We assume a small enough number of errors, following [6]. In particular, if $d = O(k/(1+\lg n))$, our algorithm is an improvement to the $O(\sigma n)$ letter-by-letter algorithm of Skiena and Sundaram [73] for general strings, where $k = \lfloor n/|p| \rfloor$. Thus, our algorithm performs better if there is, on average, at most 1 error in every other $O(1+\lg n)^{\text{th}}$ non-overlapping occurrence of p. If the number of errors is not small enough, then one should run the letter-by-letter algorithm intercalated with ours, to get an upper bound of $O(\sigma n)$ queries, giving us Theorem 1, introduced at the beginning of this section.

3 Subsequence Queries

We study the query complexity for a length-n string, S, subject to yes/no ***subsequence*** queries, IsSubseq, i.e., queries of the form "Is X a subsequence of S?" We begin with a simple lower bound.

Theorem 7. ⊛ *Reconstructing a length-n periodic string, $S = p^k p'$, of smallest period p, requires at least $|p|\lg\sigma$ IsSubseq queries, even if n and $|p|$ are known.*

Let us next describe an algorithm for reconstructing a periodic length-n periodic string, $S = p^k p'$, of smallest period p. We begin by performing either binary searches (if n is known) or doubling search (if n is unknown), using queries of the form IsSubseq(a^i) to determine the number of a's in S, for each $a \in \Sigma$. From all of these queries, we can determine the value of n if it was previously unknown. This part of our algorithm requires either $\sigma\lceil\lg n\rceil$ or $2\sigma\lceil\lg n\rceil$ queries in total, depending on whether we knew n at the outset.

If the number of a's in S is n, for any $a \in \Sigma$, then we are done, so let us assume the number of a's in S is less than n, for each $a \in \Sigma$. Thus, when we complete all our doubling/binary searches, for each letter, $a \in \Sigma$ that occurs a nonzero number of times in S, we have a maximal subsequence, S_a, of S, consisting of a's. Moreover, since S is periodic with a period that repeats k times, each S_a is periodic with a period that repeats k times. Unfortunately, at this point in the algorithm, we may not be able to determine k. So next we create a binary merge tree, T, with each of its leaves associated with a nonempty subsequence, S_a, much in the style of the well-known merge-sort algorithm, so that T has height $\lceil \lg \sigma \rceil$. We then perform a bottom-up merge-like procedure in T using $\mathsf{IsSubseq}$ queries, as follows.

Let v be an internal node in T, with children x and y for which we have inductively determined periodic subsequences, S_x and S_y, respectively, of S. Let $n_x = |S_x|$ and $n_y = |S_y|$. To create the subsequence, S_v, for v, we need to perform a merge procedure to interleave S_x and S_y. To do this, we maintain indices i and j in S_x and S_y, respectively, such that we have already determined an interleaving, $S_v[..i + j]$, of $S_x[..i]$ and $S_y[..j]$. Initially, $i = j = 0$. We then perform the query $\mathsf{IsSubseq}(S_v[..i+j] \cdot S_x[i+1] \cdot S_y[j+1..n_y])$. Suppose the answer to this query is "yes". In this case, we set $S_v[..i+j+1] = S_v[..i+j] \cdot S_x[i+1]$ and we increment i. If, on the other hand, the answer to the above query is "no", then we set $S_v[..i + j + 1] = S_v[..i + j] \cdot S_y[j + 1]$, because in this case we know that $\mathsf{IsSubseq}(S_v[..i+j] \cdot S_y[j+1] \cdot S_x[i+1..n_x])$ would return "yes". If this latter condition occurs, then we increment j.

Let q_v denote this new interleaving prefix, $S_v[..i + j]$, and let $\hat{k} = \lfloor n/|q_v| \rfloor$. If ${q_v}^{\hat{k}}{q_v}'$ is a plausible interleaving of S_x and S_y, where ${q_v}'$ is a prefix of q_v, then we next ask the query $\mathsf{IsSubseq}({q_v}^{\hat{k}}{q_v}')$. If the answer is "yes", then we set $S_v = {q_v}^{\hat{k}}{q_v}'$ and this completes the merge. Otherwise, we continue incrementally interleaving S_x and S_y, using the current values of i and j, by iterating the procedure described above. Clearly, this merge procedure asks at most $2|q_v|$ queries in total.

Theorem 8. ⊛ *We can determine a length-n periodic string, $S = p^k p'$, of smallest period p of unknown size, using $2\sigma\lceil \lg n \rceil + 2|p|\lceil \lg \sigma \rceil$ IsSubseq queries, if n is unknown. If n is known, then $\sigma\lceil \lg n \rceil + 2|p|\lceil \lg \sigma \rceil$ IsSubseq queries suffice.*

A simple modification of our algorithm also implies the following.

Theorem 9. ⊛ *We can determine a length-n string, S, using $2\sigma\lceil \lg n \rceil + n\lceil \lg \sigma \rceil$ IsSubseq queries, without knowing the value of n in advance. If n is known, then $\sigma\lceil \lg n \rceil + n\lceil \lg \sigma \rceil$ IsSubseq queries suffice.*

This latter theorem improves a result of Skiena and Sundaram [73], who prove a query bound of $2\sigma \lg n + 1.59n \lg \sigma + 5\sigma$ when n is unknown.

4 Jumbled-Index Queries

Jumbled-indexing involves preprocessing a given string, S, so as to determine whether there exists a substring of S whose letter frequencies match the given ***Parikh vector***, i.e., a vector $\psi = (f_1, \ldots, f_\sigma)$ such that f_i is the number of occurrences in S of $a_i \in \Sigma$, e.g., see [4,7,9,10,52,62]. In this section, we study the query complexity for reconstructing an unknown length-n string, S, using jumbled-index queries. As observed by Acharya *et al.* [1,2], strings and their reversals have the same "composition multiset". This immediately implies the following negative result.

Lemma 8. ⊛ *If S is not a palindrome, then S cannot be reconstructed by yes/no jumbled-index queries, which return whether there is a substring in S with a given Parikh vector.*

Given that simple yes/no jumbled-index queries are not sufficient for string reconstruction, let us consider an extended type of yes/no jumbled-index query.

– ***Jumbled-Indexing with End-of-string symbol "\$"*** (JIE): given an ***extended*** Parikh vector, $\psi = (f_1, \ldots, f_\sigma, f_\$)$, for the letters in Σ and an end-of-string symbol, \$, which is not in Σ, this query returns a yes/no response as to whether there is a substring of $S\$$ with extended Parikh vector ψ.

Unlike the yes/no jumbled-index queries, this variant enables full reconstruction.

Theorem 10. *We can reconstruct a length-n string, S, using $(\sigma - 1)n$ JIE queries, if n is known, or $\sigma(n + 1)$ JIE queries, if n is unknown.*

Proof. Our method is to use a letter-by-letter reconstruction algorithm via an adaption of the prepend-a-letter primitive for substring queries. Suppose n is unknown. Let ψ be an extended Parikh vector for a known suffix, s, of $S\$$; initially, $\psi = (0, 0, \ldots, 0, 1)$ and $s = \$$. Then we perform a jumbled-index query for ψ_i, for each $a_i \in \Sigma$, where $\psi_i = \psi$ except that ψ_i adds 1 to the f_i value in ψ. If one of these, say, ψ_i, returns "yes", then we prepend a_i to our known suffix and we repeat this procedure using ψ_i for ψ. If all of these queries return "no", then we are done. If n is known, on the other hand, then we can skip this last test of all-no responses and we can also save at least one query with each iteration, with the algorithm otherwise being the same. □

We can also consider jumbled-index queries that return an index of a matching substring for a given Parikh vector, if such a substring exists. Though related, notice that this type of query is not subsumed by the query studied in Acharya *et al.* [1,2], which returns the number of occurrences (instead of position) of matching substrings in S. There is some ambiguity, however, if there is more than one matching substring; hence, we should consider how to handle such multiple matches. For example, if a jumbled-index query returns the indices of all matching substrings, then σ queries are clearly sufficient to reconstruct any length-n string, for any n, without knowing the value of n in advance. Thus, let us consider two more-interesting types of jumbled-index queries.

– ***Adversarial Jumbled-Indexing*** (AJI): given a Parikh vector, $\psi = (f_1, \ldots, f_\sigma)$, this query returns, in an adversarial manner, one of the starting indices of a matching substring, if such a string exists. If there is no matching substring, this query returns False.

– ***Random Jumbled-Indexing*** (RJI): given a Parikh vector, $\psi = (f_1, \ldots, f_\sigma)$, this query returns, uniformly at random, one of the indices of a substring with Parikh vector ψ if such a substring exists in S. If there is no such substring, this query returns False.

Unfortunately, for the AJI variant, there are some strings that cannot be fully reconstructed, but this is admittedly not obvious. In fact, the unreconstructability characterization of [1,2] fails for AJI queries, because the symmetry property used in their construction of pairwise "equicomposable" strings inherently yields matching substrings with symmetric (e.g. different) positions in S.

Nevertheless, we give a construction of an infinite family of pairwise undistinguishable strings, i.e. two strings such that, for every possible query, there exists an answer (positive or negative) that is common to both strings. Clearly, the adversarial strategy is to output these common answers when given either of these strings. In particular, for all $b \geq 1$, consider the two binary strings of length $4b + 14$ given below, which differ only in the middle section, consisting of `01` in the first string and `10` in the second:

$$S_1 = \texttt{101101(10)}^b 01(10)^b\texttt{010010}$$
$$S_2 = \texttt{101101(10)}^b 10(10)^b\texttt{010010}$$

Theorem 11. ⊛ *The strings S_1 and S_2 cannot be distinguished using AJI queries, for $b \geq 1$.*

In contrast, the query variant RJI can be used to reconstruct any length-n string, S, without knowing the value of n in advance. In particular, it is possible to reconstruct any length-n string, S, using $O(\sigma + n \log n)$ RJI queries with high probability. Our algorithm for doing this involves a reduction to a multi-window coupon-collector problem.

Let ψ_i be a Parikh vector that is all 0's except for a count of 1 for the letter $a_i \in \Sigma$. Note that an RJI query using ψ_i will return one of the n_i locations in S with an a_i uniformly at random (if $n_i > 0$). If $n_i = 0$, for any $i = 1, 2, \ldots, \sigma$, we learn this fact immediately after one RJI query for ψ_i, so let us assume, w.l.o.g., that $n_i > 0$, for all $i = 1, 2, \ldots, \sigma$, after performing an initial σ number of RJI queries.

Recall that in the ***coupon-collector*** problem, a collector visits a coupon window each day and requests a coupon from an agent, who chooses one of n coupons uniformly at random and gives it to the collector, e.g., see [61]. The expected number of days required for the collector to get al.l n coupons is nH_n, where H_n is the n^{th} Harmonic number. But this assumes the collector knows when they have received all n coupons (i.e., the collector knows the value of n).

In a coupon-collector formulation of our reconstruction problem, we instead have σ coupon windows, one for each letter $a_i \in \Sigma$, where each window i has n_i coupons that differ from the coupons for the other windows, and we do not know the value of any n_i. Each day the collector must choose one of the coupon windows, i, and request one of its coupons (corresponding to an RJI query for ψ_i), which is chosen uniformly at random from the n_i coupons for window i. We are interested in a strategy and analysis for the collector to collect all $n = n_1 + n_2 + \cdots + n_\sigma$ coupons, with high probability (i.e., with probability at least $1 - 1/n$).

Note that although we do not know the value of any n_i, we can nonetheless test whether the collector has collected all n coupons. In particular, suppose we have received RJI responses for all indices, $1, 2, \ldots, n$, for letters in S, and let n_i be the number of a_i's we have found so far. Let $\psi' = (n_1, n_2, \ldots, n_\sigma)$, and let ψ'_i be equal to ψ' except that we increment n_i by 1. If an RJI query for each ψ'_i returns False, then we know we have fully reconstructed S. Thus, if $n = 1$, then we can determine this and S after 2σ RJI queries, so let us assume that $n \geq 2$. Further, we can assume we have a bound, $N \geq 2$, which is at least n and at most twice n, by a simple doubling strategy, where we double N any time a test for n fails and we set N equal to any RJI query response that is larger than N. Therefore, the remaining problem is to solve the multi-window coupon-collector problem.

Our strategy for the multi-window coupon-collector problem is simply to visit the coupon windows in phases, so that in phase i we repeatedly visit window i until we are confident we have all of its n_i coupons, for which the following lemma will prove useful.

Lemma 9. ⊛ *Let T_i be the number of trips to window i needed to collect all its $n_i \geq 1$ coupons. Then, for any real number β:*

$$\Pr\left(T_i > \beta n_i \ln N\right) \leq \frac{n_i}{N^\beta}.$$

Our strategy, then, is to let $\beta \geq 2$ be constant, and in phase i, implement a doubling strategy where we perform $\beta N_i \log N$ RJI queries for ψ_i, such that N_i is an upper bound estimate for n_i, which we double each time we get more than N_i distinct responses to our queries in this phase. So by the end of the phase i, $n_i \leq N_i \leq 2n_i$. This gives us:

Theorem 12. ⊛ *A string, S, of unknown size, n, can be reconstructed using $O(\sigma + n \log n)$ RJI queries, with high probability.*

5 Conclusion and Open Questions

We have studied the reconstruction of strings under the following settings, by giving efficient reconstruction algorithms and proving lower bounds: (i) periodic strings of known and unknown sizes, with and without mismatch errors, using substring queries; (ii) periodic strings of known and unknown sizes, using subsequence queries and (iii) general strings, using variations of jumbled-indexing queries. For the non-optimal algorithms given here, it would be nice to know whether there exist matching lower bounds, or whether there exist faster algorithms. We mention additional possible future work in the full version of the paper [5].

Acknowledgments. This research was funded in part by the U.S. National Science Foundation under grant 1815073. Amihood Amir was partly supported by BSF grant 2018141 and ISF grant 1475-18.

References

1. Acharya, J., Das, H., Milenkovic, O., Orlitsky, A., Pan, S.: Quadratic-backtracking algorithm for string reconstruction from substring compositions. In: 2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, 29 June–4 July 2014, pp. 1296–1300. IEEE (2014). https://doi.org/10.1109/ISIT.2014.6875042
2. Acharya, J., Das, H., Milenkovic, O., Orlitsky, A., Pan, S.: String reconstruction from substring compositions. SIAM J. Discrete Math. **29**(3), 1340–1371 (2015). https://doi.org/10.1137/140962486
3. Afshani, P., Agrawal, M., Doerr, B., Doerr, C., Larsen, K.G., Mehlhorn, K.: The query complexity of finding a hidden permutation. In: Brodnik, A., López-Ortiz, A., Raman, V., Viola, A. (eds.) Space-Efficient Data Structures, Streams, and Algorithms. LNCS, vol. 8066, pp. 1–11. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40273-9_1
4. Afshani, P., van Duijn, I., Killmann, R., Nielsen, J.S.: A lower bound for jumbled indexing. In: 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 592–606 (2020). https://doi.org/10.1137/1.9781611975994.36
5. Afshar, R., Amir, A., Goodrich, M.T., Matias, P.: Adaptive exact learning in a mixed-up world: dealing with periodicity errors, and jumbled-index queries in string reconstruction. arXiv preprint arXiv:2007.08787 (2029). https://arxiv.org/abs/2007.08787
6. Amir, A., Eisenberg, E., Levy, A., Porat, E., Shapira, N.: Cycle detection and correction. ACM Trans. Alg. **9**(1) (2012). Article no. 13
7. Amir, A., Apostolico, A., Hirst, T., Landau, G.M., Lewenstein, N., Rozenberg, L.: Algorithms for jumbled indexing, jumbled border and jumbled square on run-length encoded strings. Theor. Comput. Sci. **656**, 146–159 (2016). https://doi.org/10.1016/j.tcs.2016.04.030. http://www.sciencedirect.com/science/article/pii/S030439751630069X
8. Amir, A., et al.: Pattern matching with address errors: rearrangement distances. J. Comput. Syst. Sci. **75**(6), 359–370 (2009). https://doi.org/10.1016/j.jcss.2009.03.001
9. Amir, A., Butman, A., Porat, E.: On the relationship between histogram indexing and block-mass indexing. Philos. Trans. Roy. Soc. Math. Phys. Eng. Sci. **372**(2016) (2014). https://doi.org/10.1098/rsta.2013.0132. https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2013.0132
10. Amir, A., Chan, T.M., Lewenstein, M., Lewenstein, N.: On hardness of jumbled indexing. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 114–125. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43948-7_10
11. Amir, A., Hartman, T., Kapah, O., Levy, A., Porat, E.: On the cost of interchange rearrangement in strings. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 99–110. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75520-3_11
12. Angluin, D.: Queries and concept learning. Mach. Learn. **2**(4), 319–342 (1988). https://doi.org/10.1023/A:1022821128753

13. Arratia, R., Martin, D., Reinert, G., Waterman, M.S.: Poisson process approximation for sequence repeats and sequencing by hybridization. J. Comput. Biol. **3**(3), 425–463 (1996). https://doi.org/10.1089/cmb.1996.3.425
14. Batu, T., Kannan, S., Khanna, S., McGregor, A.: Reconstructing strings from random traces. In: Munro, J.I. (ed.) Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, 11–14 January 2004, pp. 910–918. SIAM (2004). http://dl.acm.org/citation.cfm?id=982792.982929
15. Benson, G.: Tandem repeats finder: a program to analyze DNA sequence. Nucleic Acids Res. **27**(2), 573–580 (1999)
16. Benson, G., Waterman, M.: A method for fast database search for all k-nucleotide repeats. Nucleic Acids Res. **22**, 4828–4836 (1994)
17. Bentley, J.L., Yao, A.C.: An almost optimal algorithm for unbounded searching. Inf. Process. Lett. **5**(3), 82–87 (1976). https://doi.org/10.1016/0020-0190(76)90071-5
18. Bernasconi, A., Damm, C., Shparlinski, I.: Circuit and decision tree complexity of some number theoretic problems. Inf. Comput. **168**(2), 113–124 (2001). https://doi.org/10.1006/inco.2000.3017. http://www.sciencedirect.com/science/article/pii/S0890540100930177
19. Bresler, G., Bresler, M., Tse, D.: Optimal assembly for high throughput shotgun sequencing. BMC Bioinform. **14**(2013). Article number. S18. https://doi.org/10.1186/1471-2105-14-S5-S18
20. Burcsi, P., Cicalese, F., Fici, G., Lipták, Z.: Algorithms for jumbled pattern matching in strings. Int. J. Found. Comput. Sci. **23**(2), 357–374 (2012). https://doi.org/10.1142/S0129054112400175
21. Butman, A., Eres, R., Landau, G.M.: Scaled and permuted string matching. Inf. Process. Lett. **92**(6), 293–297 (2004). https://doi.org/10.1016/j.ipl.2004.09.002
22. Carpi, A., de Luca, A.: Words and special factors. Theor. Comput. Sci. **259**(1–2), 145–182 (2001). https://doi.org/10.1016/S0304-3975(99)00334-5
23. Cayley, A.: LXXVII. Note on the theory of permutations. Lond. Edinb. Dublin Philos. Mag. J. Sci. **34**(232), 527–529 (1849)
24. Chang, Z., Chrisnata, J., Ezerman, M.F., Kiah, H.M.: Rates of DNA sequence profiles for practical values of read lengths. IEEE Trans. Inf. Theory **63**(11), 7166–7177 (2017). https://doi.org/10.1109/TIT.2017.2747557
25. Choi, S.S., Kim, J.H.: Optimal query complexity bounds for finding graphs. Artif. Intell. **174**(9), 551–569 (2010). https://doi.org/10.1016/j.artint.2010.02.003. http://www.sciencedirect.com/science/article/pii/S0004370210000251
26. Cicalese, F., Fici, G., Lipták, Z.: Searching for jumbled patterns in strings. In: Holub, J., Zdárek, J. (eds.) Proceedings of the Prague Stringology Conference 2009, Prague, Czech Republic, 31 August–2 September 2009, pp. 105–117. Prague Stringology Club, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague (2009). http://www.stringology.org/event/2009/p10.html
27. Cieplinski, L.: MPEG-7 color descriptors and their applications. In: Skarbek, W. (ed.) CAIP 2001. LNCS, vol. 2124, pp. 11–20. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44692-3_3
28. Cleve, R., et al.: Reconstructing strings from substrings with quantum queries. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 388–397. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31155-0_34
29. Dakic, T.: On the turnpike problem. Simon Fraser University BC, Canada (2000)

30. Deininger, P.: SINEs: short interspersed repeated DNA elements in higher eukaryotes. In: Berg, D., Howe, M. (eds.) Mobile DNA, Chap. 27, pp. 619–636. American Society for Microbiology (1989)
31. Deselaers, T., Keysers, D., Ney, H.: Features for image retrieval: an experimental comparison. Inf. Retrieval **11**(2), 77–107 (2008). https://doi.org/10.1007/s10791-007-9039-3
32. Dobzinski, S., Vondrak, J.: From query complexity to computational complexity. In: Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing, STOC 2012, pp. 1107–1116. ACM, New York (2012). https://doi.org/10.1145/2213977.2214076
33. Domaniç, N.O., Preparata, F.P.: A novel approach to the detection of genomic approximate tandem repeats in the levenshtein metric. J. Comput. Biol. **14**(7), 873–891 (2007)
34. Dudík, M., Schulman, L.J.: Reconstruction from subsequences. J. Comb. Theory Ser. A **103**(2), 337–348 (2003). https://doi.org/10.1016/S0097-3165(03)00103-1
35. Dudley, J., Lin, M.T., Le, D., Eshleman, J.R.: Microsatellite instability as a biomarker for PD-1 blockade. Clin. Cancer Res. **22**(4), 813–820 (2016)
36. Elishco, O., Gabrys, R., Médard, M., Yaakobi, E.: Repeat-free codes. In: IEEE International Symposium on Information Theory, ISIT 2019, Paris, France, 7–12 July 2019, pp. 932–936. IEEE (2019). https://doi.org/10.1109/ISIT.2019.8849483
37. Eres, R., Landau, G.M., Parida, L.: Permutation pattern discovery in biosequences. J. Comput. Biol. **11**(6), 1050–1060 (2004). https://doi.org/10.1089/cmb.2004.11.1050
38. Fici, G., Mignosi, F., Restivo, A., Sciortino, M.: Word assembly through minimal forbidden words. Theor. Comput. Sci. **359**(1–3), 214–230 (2006). https://doi.org/10.1016/j.tcs.2006.03.006
39. Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. Proc. Am. Math. Soc. **16**(1), 109–114 (1965)
40. Gabrys, R., Milenkovic, O.: The hybrid k-Deck problem: reconstructing sequences from short and long traces. In: 2017 IEEE International Symposium on Information Theory, ISIT 2017, Aachen, Germany, 25–30 June 2017, pp. 1306–1310. IEEE (2017). https://doi.org/10.1109/ISIT.2017.8006740
41. Gabrys, R., Milenkovic, O.: Unique reconstruction of coded sequences from multiset substring spectra. In: 2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, 17–22 June 2018, pp. 2540–2544. IEEE (2018). https://doi.org/10.1109/ISIT.2018.8437909
42. Ganguly, S., Mossel, E., Rácz, M.Z.: Sequence assembly from corrupted shotgun reads. In: IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, 10–15 July 2016, pp. 265–269. IEEE (2016). https://doi.org/10.1109/ISIT.2016.7541302
43. Holenstein, T., Mitzenmacher, M., Panigrahy, R., Wieder, U.: Trace reconstruction with constant deletion probability and related results. In: Teng, S. (ed.) Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, 20–22 January 2008, pp. 389–398. SIAM (2008). http://dl.acm.org/citation.cfm?id=1347082.1347125
44. Iwama, K., Teruyama, J., Tsuyama, S.: Reconstructing strings from substrings: optimal randomized and average-case algorithms (2018)
45. Jeong, K., Bandeira, N., Kim, S., Pevzner, P.A.: Gapped spectral dictionaries and their applications for database searches of tandem mass spectra. Mol Cell Proteomics (2011). https://doi.org/10.1074/mcp.M110.002220

46. Jerrum, M.: The complexity of finding minimum-length generator sequences. Theor. Comput. Sci. **36**, 265–289 (1985). https://doi.org/10.1016/0304-3975(85)90047-7
47. Kalashnik, L.: The reconstruction of a word from fragments. In: Numerical Mathematics and Computer Technology, pp. 56–57 (1973)
48. Kannan, S., McGregor, A.: More on reconstructing strings from random traces: insertions and deletions. In: Proceedings of the 2005 IEEE International Symposium on Information Theory, ISIT 2005, Adelaide, South Australia, Australia, 4–9 September 2005, pp. 297–301. IEEE (2005). https://doi.org/10.1109/ISIT.2005.1523342
49. Kiah, H.M., Puleo, G.J., Milenkovic, O.: Codes for DNA sequence profiles. IEEE Trans. Inf. Theory **62**(6), 3125–3146 (2016). https://doi.org/10.1109/TIT.2016.2555321
50. Kim, S., Bandeira, N., Pevzner, P.A.: Spectral profiles: a novel representation of tandem mass spectra and its applications for de novo peptide sequencing and identification. Mol. Cell. Proteomics **8**, 1391–1400 (2009)
51. Kim, S., Gupta, N., Bandeira, N., Pevzner, P.A.: Spectral dictionaries: integrating de novo peptide sequencing with database search of tandem mass spectra. Mol. Cell. Proteomics **8**(1), 53–69 (2009)
52. Kociumaka, T., Radoszewski, J., Rytter, W.: Efficient indexes for jumbled pattern matching with constant-sized alphabet. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 625–636. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40450-4_53
53. Kolpakov, R., Kucherov, G.: mreps: efficient and flexible detection of tandem repeats in DNA. Nucleic Acids Res. **31**, 3672–3678 (2003). http://www.loria.fr/mreps/
54. Krasikov, I., Roditty, Y.: On a reconstruction problem for sequences. J. Comb. Theory Ser. A **77**(2), 344–348 (1997). https://doi.org/10.1006/jcta.1997.2732
55. Levenshtein, V.I.: Binary codes capable of correcting, deletions, insertions and reversals. Soviet Phys. Dokl. **10**, 707–710 (1966)
56. Levenshtein, V.I.: Efficient reconstruction of sequences. IEEE Trans. Inf. Theory **47**(1), 2–22 (2001). https://doi.org/10.1109/18.904499
57. Lowrance, R., Wagner, R.A.: An extension of the string-to-string correction problem. J. ACM **22**(2), 177–183 (1975). https://doi.org/10.1145/321879.321880
58. Manvel, B., Meyerowitz, A., Schwenk, A.J., Smith, K., Stockmeyer, P.K.: Reconstruction of sequences. Discrete Math. **94**(3), 209–219 (1991). https://doi.org/10.1016/0012-365X(91)90026-X
59. Marcovich, S., Yaakobi, E.: Reconstruction of strings from their substrings spectrum. CoRR abs/1912.11108 (2019). http://arxiv.org/abs/1912.11108
60. Margaritis, D., Skiena, S.S.: Reconstructing strings from substrings in rounds. In: IEEE 36th Symposium on Foundations of Computer Science (FOCS), pp. 613–620, October 1995. https://doi.org/10.1109/SFCS.1995.492591
61. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis, 2nd edn. Cambridge University Press, Cambridge (2017)
62. Moosa, T.M., Rahman, M.S.: Indexing permutations for binary strings. Inf. Process. Lett. **110**(18), 795–798 (2010). https://doi.org/10.1016/j.ipl.2010.06.012. http://www.sciencedirect.com/science/article/pii/S0020019010002012
63. Motahari, A.S., Bresler, G., Tse, D.N.C.: Information theory of DNA shotgun sequencing. IEEE Trans. Inf. Theory **59**(10), 6273–6289 (2013). https://doi.org/10.1109/TIT.2013.2270273

64. Motahari, A.S., Ramchandran, K., Tse, D., Ma, N.: Optimal DNA shotgun sequencing: noisy reads are as good as noiseless reads. In: Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, 7–12 July 2013, pp. 1640–1644. IEEE (2013). https://doi.org/10.1109/ISIT.2013.6620505
65. Parisi, V., Fonzo, V.D., Aluffi-Pentini, F.: STRING: finding tandem repeats in DNA sequences. Bioinformatics **19**(14), 1733–1738 (2003)
66. Pellegrini, M., Renda, M.E., Vecchio, A.: TRStalker: an efficient heuristic for finding fuzzy tandem repeats. Bioinformatics [ISMB] **26**(12), 358–366 (2010)
67. Sala, F., Gabrys, R., Schoeny, C., Mazooji, K., Dolecek, L.: Exact sequence reconstruction for insertion-correcting codes. In: IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, 10–15 July 2016, pp. 615–619. IEEE (2016). https://doi.org/10.1109/ISIT.2016.7541372
68. Scott, A.D.: Reconstructing sequences. Discrete Math. **175**(1–3), 231–238 (1997). https://doi.org/10.1016/S0012-365X(96)00153-7
69. Shomorony, I., Courtade, T.A., Tse, D.N.C.: Do read errors matter for genome assembly? In: IEEE International Symposium on Information Theory, ISIT 2015, Hong Kong, China, 14–19 June 2015, pp. 919–923. IEEE (2015). https://doi.org/10.1109/ISIT.2015.7282589
70. Shomorony, I., Kamath, G.M., Xia, F., Courtade, T.A., Tse, D.N.C.: Partial DNA assembly: a rate-distortion perspective. In: IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, 10–15 July 2016, pp. 1799–1803. IEEE (2016). https://doi.org/10.1109/ISIT.2016.7541609
71. Simon, I.: Piecewise testable events. In: Brakhage, H. (ed.) GI-Fachtagung 1975. LNCS, vol. 33, pp. 214–222. Springer, Heidelberg (1975). https://doi.org/10.1007/3-540-07407-4_23
72. Skiena, S., Smith, W.D., Lemke, P.: Reconstructing sets from interpoint distances (extended abstract). In: Seidel, R. (ed.) Proceedings of the Sixth Annual Symposium on Computational Geometry, Berkeley, CA, USA, 6–8 June 1990, pp. 332–339. ACM (1990). https://doi.org/10.1145/98524.98598
73. Skiena, S., Sundaram, G.: Reconstructing strings from substrings. J. Comput. Biol. **2**(2), 333–353 (1995). https://doi.org/10.1089/cmb.1995.2.333
74. Sokol, D.: TRedD - a database for tandem repeats over the edit distance. Database J. Biol. Databases Curation **2010**(baq003) (2010). https://doi.org/10.1093/database/baq003
75. Tan, K., Ooi, B.C., Yee, C.Y.: An evaluation of color-spatial retrieval techniques for large image databases. Multimed. Tools Appl. **14**(1), 55–78 (2001). https://doi.org/10.1023/A:1011359607594
76. Tardos, G.: Query complexity, or why is it difficult to separate $NP^A \cap coNP^A$ from P^A by random oracles A? Combinatorica **9**(4), 385–392 (1989). https://doi.org/10.1007/BF02125350
77. Tsur, D.: Tight bounds for string reconstruction using substring queries. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX/RANDOM -2005. LNCS, vol. 3624, pp. 448–459. Springer, Heidelberg (2005). https://doi.org/10.1007/11538462_38
78. Ukkonen, E.: Approximate string matching with q-grams and maximal matches. Theor. Comput. Sci. **92**(1), 191–211 (1992). https://doi.org/10.1016/0304-3975(92)90143-4
79. Viswanathan, K., Swaminathan, R.: Improved string reconstruction over insertion-deletion channels. In: Teng, S. (ed.) Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California,

USA, 20–22 January 2008, pp. 399–408. SIAM (2008). http://dl.acm.org/citation.cfm?id=1347082.1347126
80. Wagner, R.A.: On the complexity of the extended string-to-string correction problem. In: Rounds, W.C., Martin, N., Carlyle, J.W., Harrison, M.A. (eds.) Proceedings of the 7th Annual ACM Symposium on Theory of Computing, Albuquerque, New Mexico, USA, 5–7 May 1975, pp. 218–223. ACM (1975). https://doi.org/10.1145/800116.803771
81. Wang, J., Hua, X.: Interactive image search by color map. ACM Trans. Intell. Syst. Technol. **3**(1), 12:1–12:23 (2011)
82. Wexler, Y., Yakhini, Z., Kashi, Y., Geiger, D.: Finding approximate tandem repeats in genomic sequences. In: RECOMB, pp. 223–232 (2004)
83. Yao, A.C.C.: Decision tree complexity and Betti numbers. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, STOC 1994, pp. 615–624. ACM, New York (1994). https://doi.org/10.1145/195058.195414
84. Zenkin, A., Leont'ev, V.K.: On a non-classical recognition problem. USSR Comput. Math. Math. Phys. **24**(3), 189–193 (1984)
85. Zhou, W., Li, H., Tian, Q.: Recent advance in content-based image retrieval: a literature survey. CoRR abs/1706.06064 (2017). http://arxiv.org/abs/1706.06064