Reverse-Safe Text Indexing

Bernardini, Giulia; Chen, Huiping; Fici, Gabriele; Loukides, Grigorios; Pissis, Solon P.

DOI:
10.1145/3461698

Citation for published version (Harvard):
Bernardini, G, Chen, H, Fici, G, Loukides, G & Pissis, SP 2021, 'Reverse-Safe Text Indexing', *ACM Journal of Experimental Algorithms*, vol. 26, no. 1, 3461698. https://doi.org/10.1145/3461698

Reverse-Safe Text Indexing

GIULIA BERNARDINI, Università di Milano
HUIPING CHEN, King's College London
GABRIELE FICI, Università degli Studi di Palermo
GRIGORIOS LOUKIDES, King's College London
SOLON P. PISSIS, CWI and Vrije Universiteit

We introduce the notion of reverse-safe data structures. These are data structures that prevent the reconstruction of the data they encode (i.e., they cannot be easily reversed). A data structure D is called *z-reverse-safe* when there exist at least z datasets with the same set of answers as the ones stored by D. The main challenge is to ensure that D stores as many answers to useful queries as possible, is constructed efficiently, and has size close to the size of the original dataset it encodes. Given a text of length n and an integer z, we propose an algorithm that constructs a z-reverse-safe data structure (z-RSDS) that has size $O(n)$ and answers decision and counting pattern matching queries of length at most d optimally, where d is maximal for any such z-RSDS. The construction algorithm takes $O(n^{\omega} \log d)$ time, where ω is the matrix multiplication exponent. We show that, despite the n^{ω} factor, our engineered implementation takes only a few minutes to finish for million-letter texts. We also show that plugging our method in data analysis applications gives insignificant or no data utility loss. Furthermore, we show how our technique can be extended to support applications under realistic adversary models. Finally, we show a z-RSDS for decision pattern matching queries, whose size can be sublinear in n. A preliminary version of this article appeared in ALENEX 2020.

CCS Concepts: • **Theory of computation → Pattern matching;**

Additional Key Words and Phrases: Data structures, data privacy, pattern matching, text indexing, suffix tree

ACM Reference format:
Giulia Bernardini, Huiping Chen, Gabriele Fici, Grigorios Loukides, and Solon P. Pissis. 2021. Reverse-Safe Text Indexing. *J. Exp. Algorithmics* 26, 1, Article 1.10 (July 2021), 26 pages.
https://doi.org/10.1145/3461698

1 INTRODUCTION

Data structures organize data allowing for their efficient access and modification. They are thus the workhorse of many data analysis applications, such as clustering and outlier detection (e.g., through indexes for k-nearest neighbors join queries [Böhm and Krebs 2004]), frequent pattern mining (e.g., through FP-trees [Han et al. 2000]), document retrieval (e.g., through inverted indexes [Manning et al. 2008]), graph pattern matching (e.g., through graph indexes [Wang et al. 2016]), and range search in databases (e.g., through R-trees [Guttman 1984]).

H. Chen was supported by a CSC Scholarship. G. Fici was supported in part by MIUR-PRIN 2017 project 2017K7XPAN "Algorithms, Data Structures and Combinatorics for Machine Learning." G. Loukides was supported in part by the Leverhulme Trust project RPG-2019-399.
Authors' addresses: G. Bernardini, Università di Milano–Bicocca, Viale Sarca 336, Milano, 20100, Italy; email: giulia.bernardini@unimib.it; H. Chen and G. Loukides, Department of Informatics, King's College London, Bush House, 30 Aldwych, London, WC2B 4BG, UK; emails: {huiping.chen, grigorios.loukides}@kcl.ac.uk; G. Fici, Università degli Studi di Palermo, Via Archirafi 34, Palermo, 90123, Italy; email: gabriele.fici@unipa.it; S. P. Pissis, CWI, Science Park 123, Amsterdam, 1098 XG, Netherlands, and Vrije Universiteit, Amsterdam, Netherlands; email: solon.pissis@cwi.nl.

1084-6654/2021/07-ART1.10
https://doi.org/10.1145/3461698

These applications are often fueled by data collected from individuals, such as location, genomic, or customer data, and have led to justified privacy concerns [Smith et al. 2011]. To alleviate these concerns and comply with legislation such as HIPAA [U.S. Department of Health & Human Services 1996] in the United States and GDPR [European Parliament 2015] in the European Union, it is necessary to guarantee that using data structures does not lead to the reconstruction of the stored individuals' data. This is a fundamentally different privacy goal than that of existing privacy-preserving techniques, such as anonymization [Chen et al. 2012a, b; Heatherly et al. 2013; Xu et al. 2016], sanitization [Bernardini et al. 2019, 2020a, 2020c, 2020d; Bonomi et al. 2016; Gkoulalas-Divanis and Loukides 2011; Gwadera et al. 2013; Loukides and Gwadera 2015; Wang et al. 2013], query auditing [Nabar et al. 2008], or access control [Bertino et al. 2011]. Anonymization aims at preventing the disclosure of individuals' identities and/or sensitive information. Sanitization aims at preventing the mining of confidential knowledge. Query auditing aims at preventing answering aggregate queries that leak private information. Access control is the selective restriction of access to some parts of a database. Our privacy goal is also different from that of encryption techniques, such as searchable encryption [Bezawada et al. 2015; Li et al. 2010; Qin et al. 2019], which aim to prevent unauthorized parties from accessing the data.

We consider a setting where a large group of users want to query a dataset directly via a data structure that prevents the reconstruction of the data. To this end, we introduce a novel encoding model that enables the construction of **reverse-safe data structures (RSDSs)**. The ultimate aim of an RSDS is to make the reconstruction of a dataset sufficiently unlikely so that an adversary cannot infer the dataset based on the query answers, but at the same time the RSDS stores as many answers to useful queries as possible to support applications. In addition, the RSDS should be constructed efficiently and have size close to the size of the original dataset it encodes. Our idea is inspired by encoding data structures [Raman 2015]. The ultimate aim of an encoding data structure is to break the information-theoretical lower bound, which is required to store a dataset, by storing only the answers to useful queries (e.g., range queries [Fischer and Heun 2011; Grossi et al. 2017] or nearest largest value queries [Hoffmann et al. 2018]).

Given a data structure D, we denote by $\mathcal{A}_D$ its set of *consistent* datasets: all datasets with the same set of answers as the answers stored by D. Let us denote $\alpha_D = |\mathcal{A}_D|$. Given an integer threshold $z > 1$, which we call the *privacy threshold*, we say that D is a **z-reverse-safe data structure (z-RSDS)** if and only if $\alpha_D \geq z$. A large z implies strong data privacy because an adversary cannot distinguish between the $\alpha_D \geq z$ consistent datasets, which implies that it is less likely that the adversary infers the dataset used to construct D in the first place. Still, it could be the case that D stores answers to many useful queries.

The notion of z-RSDSs is related to the privacy notion of *z-anonymity* [Samarati and Sweeney 1998]. This notion was introduced in the context of a relational database, where each record corresponds to a different individual. The notion of z-anonymity dictates that at least $z > 1$ records of the database must have the same values over a set of attributes that may lead to the disclosure of the identity of individuals in the database. The privacy goal is to prevent an adversary from distinguishing an individual among at least z individuals in the database.

In this work, we consider string data (sometimes called *text*, *word*, or *document* depending on the context). A string is a sequence of letters from an alphabet. A string may represent various types of confidential information about individuals, including their movement history [Theodorakopoulos et al. 2014], diagnosed diseases [Tamersoy et al. 2012], purchased products [Terrovitis et al. 2017], or DNA sequence [Malin and Sweeney 2000]. Our goal is to construct a z-RSDS for string data that allows for decision and counting pattern matching queries to be accurately and efficiently answered. Decision queries are fundamental for intrusion detection [Lin et al. 2008], activity monitoring [Wang et al. 2013], and cataloguing human genetic variation

[Bentley 2006], whereas counting queries are fundamental for pattern mining that is central in application domains ranging from bioinformatics [Shang et al. 2016] to marketing [Loukides and Gwadera 2015] and to public health [Banerjee et al. 2019].

Pattern matching queries in strings are answered efficiently by means of indexing data structures. These structures enable fast access to the substrings of a string, which is important in many data analysis applications [Gusfield 1997]. The main idea behind indexing a string S for efficient substring querying is that every substring of S is a prefix of some suffix of S. Indexing data structures thus arrange the suffixes of S lexicographically in an ordered tree data structure. One popular such data structure is the *suffix tree* [Weiner 1973]. The suffix tree of S is the compacted trie of all suffixes of S. The term *compacted* refers to the fact that it reduces the number of nodes by replacing each maximal branchless path segment with a single edge, and it uses intervals over S to store the labels of these edges. This ensures that the suffix tree has size linear in $|S|$: it has no more than $2|S|$ nodes. Importantly, the suffix tree answers several types of pattern matching queries over S in optimal time; see the work of Gusfield [1997] for a nice exposition.

However, the suffix tree of S, which provides (random) access to *all* substrings of S, is *not* a z-RSDS, because it uniquely represents S. The *privacy-utility trade-off* we consider here is thus to provide access only to the substrings of S whose length is at most d, for some $d \in [1, |S|)$. In particular, we want our z-RSDS to support the following types of online queries:

Decision query: Check if a string P of length $m \leq d$ is a substring of S.
Counting query: Count the occurrences of a string P of length $m \leq d$ in S.

Given a string S and a privacy threshold z, the computational challenge is to compute the *maximal* d for which a z-RSDS for indexing S can be constructed. The maximality of d offers *data utility*, since any query for a substring of S of length d or less has the same answer, irrespectively of whether it is posed on S or on the z-RSDS. The fact that the data structure is z-reverse-safe offers *data privacy*, since the probability that an adversary infers S, based solely on knowledge of the z-RSDS, is no more than $1/z$.

We are now in a position to formally define the main computational problem considered in this article.[1]

Problem 1. *Given a string S of length n and a privacy threshold $1 < z \leq n^c$, for some constant $c \geq 1$, construct a z-RSDS that answers decision and counting pattern matching queries for any pattern of length $m \leq d$, such that d is maximal, or output FAIL if no such d exists.*

In Problem 1, d is maximal and *uniform for all queries*. Another related problem definition would be to maximize the total number of sup ported queries, not necessarily of uniform maximal length.

Our contributions. We consider the word-RAM model of computations with w-bit machine words, where $w = \Omega(\log n)$, for stating our results. The main theoretical result of this work is the following, where ω denotes the matrix multiplication exponent.[2]

Theorem 1. *Given a string S of length n, there exists an $O(n^\omega \log d)$-time algorithm to construct an $O(n)$-sized z-RSDS over S for a maximal d that answers decision and counting pattern matching queries, for any pattern of length $m \leq d$, in the optimal $O(m)$ time per query. The algorithm outputs FAIL if no such d exists.*

The main ingredients of our construction algorithm include (truncated) suffix trees [Na et al. 2003; Weiner 1973], a combinatorial theorem on de Bruijn graphs [Hutchinson 1975; Karhumäki

[1] The problem of inferring a string from a text indexing data structure (see the work of Kärkkäinen et al. [2017] and references therein) is conceptually related but fundamentally different to the problem investigated here.
[2] At the time of writing this article, $\omega < 2.373$ [Williams 2012; Le Gall 2014].

et al. 2017], and fast matrix multiplication [Williams 2012; Le Gall 2014]. To the best of our knowledge, we are the first to combine these ingredients. We show that, despite the n^ω factor, our engineered implementation can construct z-RSDSs over million-letter texts in only a few minutes. To achieve this practical performance, we rely on further theoretical insight. We also show that plugging our method in data analysis applications gives insignificant or no data utility loss. Furthermore, we show how our technique can be extended at no extra cost to construct a z-RSDS that supports applications under two realistic adversary models: one with positive adversarial knowledge (an adversary knows a pattern that occurs in S), and the other with negative adversarial knowledge (an adversary knows that a pattern does not occur in S). We also show how the z-RSDS for both adversary models can be generalized to an arbitrary number of patterns. Both positive and negative adversarial knowledge have been studied in the context of z-anonymity [Li and Li 2008; Atzori et al. 2008]. Li and Li [2008] and Atzori et al. [2008] have a different privacy goal than ours (preventing the disclosure of identities of individuals and/or their sensitive information vs. preventing dataset reconstruction) and do not consider a string but a relational and a set-valued dataset, respectively. Finally, we show a different z-RSDS for decision pattern matching queries, whose size can be sublinear in n.

Organization of the article. The basic definitions and notation are introduced in Section 2. In Section 3, we propose a z-RSDS for text indexing. In Section 4, we present our construction algorithm. We then describe a series of practical improvements in Section 5. In Section 6, we present our implementation and extensive experimental results. In Section 7, we discuss how to construct an adapted version of our z-RSDS under two different adversary models. In Section 8, we show a different z-RSDS for answering decision pattern matching queries. We conclude the article in Section 9 with some final remarks and open problems.

A preliminary version of this article appeared in ALENEX 2020 [Bernardini et al. 2020b]. Compared to the preliminary version, the new materials introduced herein are the construction algorithm for negative adversarial knowledge, the generalization of both adversarial knowledge to an arbitrary number of patterns, the z-RSDS for decision pattern matching queries, and the implementation and evaluation of a version of our main construction algorithm that replaces binary search by exponential search. In fact, this is an implementation of the $O(n^\omega \log d)$-time algorithm (Theorem 1).

2 DEFINITIONS AND NOTATION

An *alphabet* Σ is a finite non-empty set whose elements are called *letters*. A *string* is a sequence of letters from Σ. We fix a string $S = S[0] \cdots S[n-1]$ over $\Sigma = \{1, \ldots, n^{O(1)}\}$. The *length* of S is denoted by $|S| = n$. We also assume that S contains at least two different letters, as otherwise the problem considered in this work is trivial. By $S[i\,.\,.\,j] = S[i] \cdots S[j]$, we denote the *substring* of S starting at position i and ending at position j of S. A substring of S is called *proper* if it is not equal to S. A substring $S[i\,.\,.\,j]$ is called a *prefix* if $i = 0$; it is called a *suffix* if $j = n - 1$. Given a positive integer k, we denote by $(S)_{k,i}$ the length-k substring of S starting at position i—that is, $(S)_{k,i} = S[i\,.\,.\,i+k-1]$, for all $0 \le i < n-k+1$. A string P has an *occurrence* in S or, more simply, it *occurs* in S if $P = (S)_{|P|,i}$, for some i. An occurrence of P is thus characterized by its starting position i in S.

The *weighted de Bruijn graph* of order k over a string S of length n is a directed multigraph $G_{S,k} = (V_{S,k}, E_{S,k})$, where the set of vertices $V_{S,k}$ is the set of length-$(k-1)$ substrings of S and $E_{S,k}$ is the multiset of edges from vertex u to vertex v for every occurrence of u and v as consecutive length-$(k-1)$ substrings of S. More formally, there is a multi-edge $(u, v) \in E_{S,k}$ with multiplicity m if and only if $u[0] \cdot v = u \cdot v[k-2]$ and this string occurs in S exactly m times. Thus, $G_{S,k}$ has exactly $n - k + 1$ edges; in general, $G_{S,k}$ contains self-loops and multi-edges (inspect Figure 3 for an example).

3 A z-RSDS FOR TEXT INDEXING

Let S be a string of length n. For a positive integer d, we define a *d-substring* of S as a substring of length d of S, or a suffix of S whose length is less than d.

The *d-truncated suffix tree* of a string S, denoted by $\mathcal{T}_d(S)$, is a path-compacted trie representing every d-substring of S [Na et al. 2003]. We make use of a terminating letter $\# \notin \Sigma$ for technical purposes. Formally, $\mathcal{T}_d(S)$ is a rooted tree satisfying the following conditions (see Figure 1 for an example):

(1) Each edge is labeled with a non-empty substring of string $S\#$ encoded as an $[i, j]$ interval over $[0, n]$.
(2) Each internal node v, except possibly the root, has at least two children. The labels of edges from v to its children start with distinct letters.
(3) Let $\mathcal{L}(v)$ denote the string obtained by concatenating labels on the path from the root to node v. For every d-substring U, there is exactly one leaf w such that $U = \mathcal{L}(w)$ (if $|U| = d$) or $U\# = \mathcal{L}(w)$ (if $|U| < d$). For each leaf w, there is at least one d-substring U such that $\mathcal{L}(w) = U$ or $\mathcal{L}(w) = U\#$.
(4) Each node v other than the root has a counter that stores the number of substrings of string $S\#$ that are equal to $\mathcal{L}(v)$.

Therefore, the number of leaves is at most n and the total number of nodes is less than $2n$. Recall that the label of the edge between node u and its child v, denoted by *label*(u, v), is represented implicitly by an interval over $[0, n]$. Thus, the space occupied by $\mathcal{T}_d(S)$ is $O(n)$. The children of internal nodes are indexed by the alphabet letters using perfect hashing to ensure $O(1)$-time access [Fredman et al. 1984]. Importantly, $\mathcal{T}_d(S)$ supports the following online pattern matching operations:

Decision query: Check if a string P of length $m \leq d$ is a substring of S in $O(m)$ time.
Counting query: Count the occurrences of a string P of length $m \leq d$ in S in $O(m)$ time.

Theorem 2 ([Na et al. 2003; Charalampopoulos et al. 2020]). *Given a string S of length n and $0 < d \leq n$, $\mathcal{T}_d(S)$ has size $O(n)$ and it can be constructed in $O(n)$ time. $\mathcal{T}_d(S)$ answers decision and counting pattern matching queries, for any pattern of length $m \leq d$, in the optimal $O(m)$ time per query.*

The following offline operations are also supported:

Frequent substrings: Find all most frequent substrings, for all lengths $1, 2, \ldots, d$, in $O(n)$ time.
Repeated substrings: Find all longest repeated substrings of length at most d in $O(n)$ time.
Unique substrings: Find all shortest unique substrings of length at most d in $O(n)$ time.

We next consider a different representation of $\mathcal{T}_d(S)$ toward defining the notion of z-RSDSs. If *label*(u, v) is represented explicitly by a string, we denote the resulting data structure by $\mathsf{TRIE}_d(S)$. In this case, string S is not part of the data structure, and thus $\mathsf{TRIE}_d(S)$ *does not*, generally, define S uniquely.

Definition 3 (d-Equivalent Strings). Given the set of all possible strings of length n over an alphabet Σ and an integer d, string S is *d-equivalent* to string S' if and only if $\mathsf{TRIE}_d(S) = \mathsf{TRIE}_d(S')$. In this case, we write $S \sim_d S'$ and say that S' is *consistent* with $\mathcal{T}_d(S)$.

See Figure 2 for an example. We can now formally define a z-RSDS for text indexing.

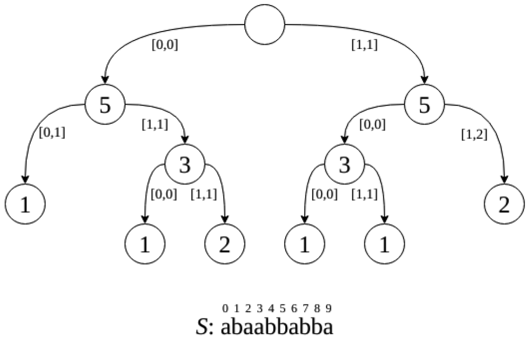


Fig. 1. $\mathcal{T}_d(S)$ for S = abaabbabba and $d = 3$. We omit edges whose labels start with letter # for clarity.

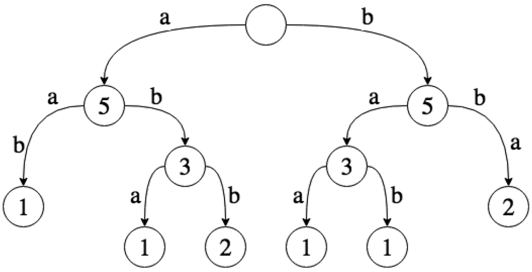


Fig. 2. Let S = abaabbabba and S' = abbaabbaba. $S \sim_3 S' \iff \text{TRIE}_3(S) = \text{TRIE}_3(S')$.

Definition 4 (z-RSDS for Text Indexing). Given an integer $z > 1$, $\mathcal{T}_d(\cdot)$ is called *z-reverse-safe* if and only if there exist at least z distinct strings that are consistent with $\mathcal{T}_d(\cdot)$.

In what follows, we denote the set of strings that are consistent with $\mathcal{T}_d(S)$ by $\mathcal{A}_d(S)$, and $|\mathcal{A}_d(S)|$ by $\alpha_d(S)$, for $d \in [1, n]$. We omit (S) when this is clear from the context, and we also set $\alpha_0(S) = \infty$ for completeness.

4 CONSTRUCTING z-RSDS

Clearly, $\mathcal{T}_n(S)$, the (non-truncated) suffix tree of S, has $\alpha_n = 1$ (i.e., it uniquely represents S), so it can never be a solution to Problem 1 since $z > 1$, by definition.

The following lemma is important for efficiency.

Lemma 5. *The sequence $\alpha_0, \alpha_1, \ldots, \alpha_n$ is monotonically non-increasing.*

Proof. Let $\mathcal{A}_d$ be the set of strings consistent with $\mathcal{T}_d$, $d \in [1, n]$, and $\alpha_d = |\mathcal{A}_d|$. Further, let S be any element of $\mathcal{A}_d$. By construction, if U is a d-substring of S, then $U = \mathcal{L}(w)$ or $U\# = \mathcal{L}(w)$, for some leaf w of $\mathcal{T}_d$. Every $(d-1)$-substring $S[i\mathinner{.\,.}i+d-2]$ of S is a prefix of the d-substring $S[i\mathinner{.\,.}i+d-1]$ of S. Thus, string S is consistent with $\mathcal{T}_{d-1}$, the path-compacted trie that represents every such $(d-1)$-substring, and thus $S \in \mathcal{A}_{d-1}$. This implies the following relation: $\mathcal{A}_n \subseteq \mathcal{A}_{n-1} \subseteq \cdots \subseteq \mathcal{A}_1$. The statement follows directly from this relation and the fact that $\alpha_0 = \infty$. □

By Lemma 5, for increasing d, $\mathcal{T}_d(S)$ generally decreases α_d and increases utility. We thus need an algorithm to compute the maximum possible d that results in a z-RSDS. We next provide an algorithm, called z-RC (for z-RSDS Construction), to find this d.

As can be seen in the pseudocode, z-RC performs binary search on n (the length of S), computing α_d until d results in a z-RSDS and d is maximal. An alternative is to perform exponential search

ALGORITHM 1: z-RC

```
   Input: string $S$ of length $n$ and integer $z > 1$
   Output: $d$ and $\mathcal{T}_d(S')$, for some $S' \in \mathcal{A}_d$, or FAIL
1  $\ell \leftarrow 0$; $r \leftarrow n$;
2  while $\ell < r$ do
3      $d \leftarrow \lfloor \frac{\ell+r}{2} \rfloor$;
4      if $\alpha_d(S) \geq z$ then
5          $\ell \leftarrow d + 1$;
6      else
7          $r \leftarrow d$;
8  if $\ell > 0$ then
9      output $d \leftarrow \ell - 1$ and $\mathcal{T}_d(S')$, for some $S' \in \mathcal{A}_d$
10 else
11     output FAIL
```

instead of binary search. We refer to this variation of z-RC as z-RCE (for z-RSDS Construction Exponential). At this point, the z-RSDS $\mathcal{T}_d(S')$ is output, where S' is an element of $\mathcal{A}_d$ chosen at random, and the algorithm terminates. If $\ell > 0$ and $\alpha_{\ell-1} = z$, then $\alpha_{\ell-1}$ is the rightmost element that equals z. Even if such an element is not found, $n - \ell$ is the number of elements that are smaller than z.

The computational challenge is thus to implement the check of line 4 efficiently and to find a consistent S' when this is possible (line 9). To this end, we start with the following simple yet crucial observation.

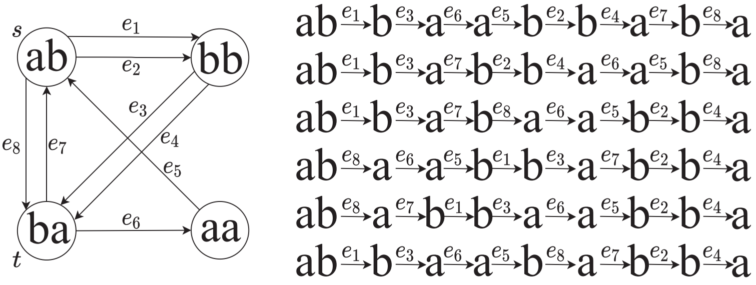
Observation 6. *Given two strings X and Y, X is d-equivalent to Y if and only if X and Y have the same multisets of substrings of length i, for every $i \in [1, d]$.*

In the terminology of combinatorics on words, d-equivalence is known as *d-abelian equivalence* [Karhumäki et al. 2013]. We report a lemma from Karhumäki et al. [2013], which gives several equivalent conditions that characterize d-equivalence.

Lemma 7 ([Karhumäki et al. 2013]). *Let X and Y be two strings of length at least d that have the same multiset of substrings of length d. The following are equivalent:*

(1) X and Y have the same multiset of substrings of length i for every $1 \leq i \leq d$;
(2) X and Y have the same prefix of length $d - 1$ and the same suffix of length $d - 1$;
(3) X and Y have the same prefix of length $d - 1$;
(4) X and Y have the same suffix of length $d - 1$.

Lemma 7 tells us that we should rely on the construction of weighted de Bruijn graphs over string S to compute $\alpha_d(S)$. The weighted de Bruijn graph of order d over string S is denoted by $G_{S,d} = (V_{S,d}, E_{S,d})$. Recall that its set of vertices $V_{S,d}$ is the set of distinct substrings of S of length $d - 1$ (we implicitly identify a vertex by the string it represents) and there is an edge $(u, v) \in E_{S,d}$ with multiplicity m if and only if $u[0] \cdot v = u \cdot v[d-2]$ and this string occurs in S exactly m times. We borrow the terminology used in the work of Kingsford et al. [2010]. Let $d^-(u)$ and $d^+(u)$ be, respectively, the in- and out-degree of vertex u of $G_{S,d}$. Let s and t be the vertices of $G_{S,d}$ corresponding, respectively, to the prefix and to the suffix of length $d - 1$ of S. Since any weighted de Bruijn graph is either Eulerian (if $s = t$) or semi-Eulerian (if $s \neq t$), we have that $d^+(u) = d^-(u)$ for all u with the possible exception of the two nodes s and t for which $d^-(s) = d^+(s) - 1$ and $d^+(t) = d^-(t) - 1$, if $s \neq t$. Clearly, S corresponds to an Eulerian path in $G_{S,d}$ that starts at s and ends at $t \neq s$ (if $s = t$, then it corresponds to an Eulerian cycle starting from s). The graph $G_{S,d}$ may



$$\mathrm{ab}\xrightarrow{e_1}\mathrm{b}\xrightarrow{e_3}\mathrm{a}\xrightarrow{e_6}\mathrm{a}\xrightarrow{e_5}\mathrm{b}\xrightarrow{e_2}\mathrm{b}\xrightarrow{e_4}\mathrm{a}\xrightarrow{e_7}\mathrm{b}\xrightarrow{e_8}\mathrm{a}$$
$$\mathrm{ab}\xrightarrow{e_1}\mathrm{b}\xrightarrow{e_3}\mathrm{a}\xrightarrow{e_7}\mathrm{b}\xrightarrow{e_2}\mathrm{b}\xrightarrow{e_4}\mathrm{a}\xrightarrow{e_6}\mathrm{a}\xrightarrow{e_5}\mathrm{b}\xrightarrow{e_8}\mathrm{a}$$
$$\mathrm{ab}\xrightarrow{e_1}\mathrm{b}\xrightarrow{e_3}\mathrm{a}\xrightarrow{e_7}\mathrm{b}\xrightarrow{e_8}\mathrm{a}\xrightarrow{e_6}\mathrm{a}\xrightarrow{e_5}\mathrm{b}\xrightarrow{e_2}\mathrm{b}\xrightarrow{e_4}\mathrm{a}$$
$$\mathrm{ab}\xrightarrow{e_8}\mathrm{a}\xrightarrow{e_6}\mathrm{a}\xrightarrow{e_5}\mathrm{b}\xrightarrow{e_1}\mathrm{b}\xrightarrow{e_3}\mathrm{a}\xrightarrow{e_7}\mathrm{b}\xrightarrow{e_2}\mathrm{b}\xrightarrow{e_4}\mathrm{a}$$
$$\mathrm{ab}\xrightarrow{e_8}\mathrm{a}\xrightarrow{e_7}\mathrm{b}\xrightarrow{e_1}\mathrm{b}\xrightarrow{e_3}\mathrm{a}\xrightarrow{e_6}\mathrm{a}\xrightarrow{e_5}\mathrm{b}\xrightarrow{e_2}\mathrm{b}\xrightarrow{e_4}\mathrm{a}$$
$$\mathrm{ab}\xrightarrow{e_1}\mathrm{b}\xrightarrow{e_3}\mathrm{a}\xrightarrow{e_6}\mathrm{a}\xrightarrow{e_5}\mathrm{b}\xrightarrow{e_8}\mathrm{a}\xrightarrow{e_7}\mathrm{b}\xrightarrow{e_2}\mathrm{b}\xrightarrow{e_4}\mathrm{a}$$

Fig. 3. $G_{S,d}$ with S = abaabbabba and $d = 3$ (on the left), and the set of d-equivalent strings (on the right).

contain other Eulerian paths (respectively, cycles). Notice, however, that if two distinct Eulerian paths (respectively, cycles) traverse the vertices of $G_{S,d}$ in the same order, but the edges in different order, then they give rise to the same string. We call these Eulerian paths (respectively, cycles) *equivalent*. We summarize these observations into the following statement, which is crucial for the correctness of the z-RC algorithm.

Observation 8. *(a) If $S \sim_d S'$, then S' corresponds to an Eulerian path in $G_{S,d}$ that starts from vertex s and ends at vertex $t \neq s$ (if $s = t$, then it corresponds to an Eulerian cycle starting from s). (b) The number of distinct strings that are d-equivalent to S is the number of* non-equivalent *Eulerian paths (respectively, cycles) in $G_{S,d}$.*

The number of non-equivalent Eulerian paths (respectively, cycles) in $G_{S,d}$ can be computed via the following theorem, which is attributed to Hutchinson [1975].

Theorem 9 ([Hutchinson 1975], cf. [Kingsford et al. 2010; Karhumäki et al. 2017]). *Let $A = (a_{uv})$ be the adjacency matrix of the weighted de Bruijn graph $G_{S,d} = (V_{S,d}, E_{S,d})$, with both $a_{uv} > 1$ (multi-edges) and $a_{uu} > 0$ (self-loops) allowed. Let $r_u = d^+(u) + 1$ if $u = t$ or $r_u = d^+(u)$ otherwise. The number of non-equivalent Eulerian paths starting at s and ending at t (respectively, the number of non-equivalent Eulerian cycles starting at s, when $t = s$) is given by*

$$(\det L_{S,d}) \cdot \left(\prod_{u \in V_{S,d}} (r_u - 1)! \right) \cdot \left(\prod_{(u,v) \in E_{S,d}} a_{uv}! \right)^{-1}, \tag{1}$$

where $L_{S,d} = (l_{uv})$ is the $|V_{S,d}| \times |V_{S,d}|$ matrix with $l_{uu} = r_u - a_{uu}$ and $l_{uv} = -a_{uv}$.

Let us denote by $|S|_x$ the number of occurrences of a string x in S. Since, by definition, $r_u = |S|_u$ and $a_{uv} = |S|_{u \cdot v[k-2]} = |S|_{u[0] \cdot v}$, Equation (1) is equivalent to

$$(\det L_{S,d}) \cdot \left(\prod_{u \in V_{S,d}} (|S|_u - 1)! \right) \cdot \left(\prod_{a \in \Sigma} |S|_{ua}! \right)^{-1}. \tag{2}$$

Equation (2), together with a combinatorial study of the strings that belong to the same d-equivalence class, can be found in the work of Karhumäki et al. [2017]. An example is provided with Figure 3.

It is, however, not immediate that Equation (1) (or the equivalent Equation (2)), involved in the check of line 4 in algorithm z-RC, can be computed efficiently. We show this next, starting with a known fact on de Bruijn graphs.

Fact 10 ([Cazaux et al. 2019]). *Given a string S of length n and $d < n$, its weighted de Bruijn graph $G_{S,d}$ can be constructed in $O(n)$ time.*

Lemma 11. *The determinant* $\det A$ *of an* $n \times n$ *non-singular matrix* A *can be computed in* $O(n^{\omega})$ *time.*

Proof. The decomposition of a non-singular matrix $A = LU$, where L and U is a lower and upper triangular matrix, respectively, is known as LU decomposition and can be computed in the same time as matrix multiplication [Bunch and Hopcroft 1974]. Given this decomposition, the determinant can be computed as $\det A = \det L \cdot \det U = \prod_{i=1}^{n} l_{ii} \cdot \prod_{i=1}^{n} u_{ii}$. This is because the determinant of any triangular matrix (e.g., L and U) is the product of its diagonal entries. □

Lemma 12. *Given* $\det L_{S,d}$*, the check of line 4 in algorithm* z*-RC can be performed in* $O(n \log n)$ *time.*

Proof. We unfold all factorials involved in the two products of Equation (1). Let us first consider the leftmost product. Observe that the total number of multiplications involved is no more than n because the sum of out-degrees over all nodes of $G_{S,d}$ is no more than n. Moreover, observe that each factor of the product is represented by $\log n$ bits because its value is no more than n. We assume a word-RAM algorithm that takes $O(n_1 + n_2)$ arithmetic operations to multiply an n_1-bit integer by an n_2-bit integer [Knuth 1998] resulting in an $(n_1 + n_2)$-bit integer. Using a $\log n$-depth divide and conquer, we can multiply these n integers in time $O(\frac{n}{2^1} 2^0 \log n + \frac{n}{2^2} 2^1 \log n + \ldots + \frac{n}{2^{\log n}} 2^{\log n - 1} \log n) = O(n \log n)$. Using an analogous argument, the rightmost product can be computed in $O(n \log n)$ time, because $G_{S,d}$ has no more than n edges, which implies that the product has at most n factors.

The leftmost product results in an $(n \log n)$-bit integer (we multiply n $\log n$-bit integers). By Hadamard's inequality [Gradshteyn and Ryzhik 2007], an upper bound on the value of $\det L_{S,d}$ is $B^n \cdot n^{n/2}$, where B is an upper bound on the values in $L_{S,d}$. Since here $B \le n$, an upper bound on $\det L_{S,d}$ is $n^n \cdot n^{n/2} = n^{3n/2}$, which can be expressed using $\log(n^{3n/2}) = 1.5n \log n$ bits. Multiplying $\det L_{S,d}$ by the leftmost product is thus done in $O(n \log n)$ time. The rightmost product also results in an $(n \log n)$-bit integer, which we multiply by z. Since $z \le n^c$ is a $c \log n$-bit integer, this is done in $O(n \log n)$ time. Thus, line 4 is checked in $O(n \log n)$ time if $\det L_{S,d}$ is known. □

We arrive at the main theoretical result.

Theorem 1. *Given a string* S *of length* n*, there exists an* $O(n^{\omega} \log d)$*-time algorithm to construct an* $O(n)$*-sized* z*-RSDS over* S *for a maximal* d *that answers decision and counting pattern matching queries, for any pattern of length* $m \le d$*, in the optimal* $O(m)$ *time per query. The algorithm outputs FAIL if no such* d *exists.*

Proof. The correctness of the z-RC algorithm follows by Lemma 5 (monotonicity) and Observation 8 (d-equivalence). The correctness of querying follows by the definition of d-equivalent strings.

The construction time follows by Fact 10, Lemmas 11 and 12, Theorem 2, and the binary search cost over $[0, n]$. Specifically, the check of line 4 is implemented in $O(n^{\omega})$ time by Fact 10 (de Bruijn graph construction) and Lemmas 11 and 12 (computing the number of non-equivalent Eulerian paths). If we find a valid d, we choose an Eulerian path (respectively, cycle) of $G_{S,d}$ to construct a string S' and then construct $\mathcal{T}_d(S')$ using Theorem 2 (truncated suffix tree contruction) in $O(n)$ time (line 9). The z-RSDS size and the time per query follow by Theorem 2. If no such d exists, the algorithm outputs FAIL.

If we apply exponential search (instead of binary search) as in z-RCE, we get an $O(n^{\omega} \log d)$-time construction. □

Colbourn et al. [1996] gave an algorithm allowing for sampling of a random arborescence rooted at a given node to be carried out in the same time as counting all such arborescences, which forms the basis of counting Eulerian paths and cycles in directed multigraphs. Hence, by plugging the algorithm of Colbourn et al. in our construction algorithm (line 9), we can also choose a string $S' \sim_d S$ *randomly* in the same time complexity.

5 ENGINEERING THE z-RC ALGORITHM

In what follows, we describe a series of practical improvements, which are based on theoretical insight.

5.1 Improvement I: Reducing the BS Interval

Lemma 13. *Let S be a string and $r(S)$ be the length of a longest substring of S occurring at least twice in S. $\mathcal{T}_d(S)$ cannot be a z-RSDS over S if $d \geq r(S) + 2$.*

Proof. Let I be the set of substrings of length $r(S) + 2$ of string S. Having set I is a sufficient condition for the unique reconstruction of S from I [Fici et al. 2006; Carpi and de Luca 2001]. This implies that, if $d \geq r(S) + 2$, $\mathcal{T}_d(S)$ defines S in a unique way (i.e., $\alpha_d = 1$), and thus $\mathcal{T}_d(S)$ cannot be a z-RSDS (since by definition $z > 1$). □

Note that the upper bound of $r(S) + 1$ can be computed in $O(n)$ time using the suffix tree of S [Farach-Colton 1997], which is much faster than computing the bounds found by an exponential search. This is because exponential search takes $O(n^\omega)$ time for each of its iterations. As a consequence of Lemma 13, we can reduce the binary search interval from $[0, n]$ to $[0, r(S) + 1]$ in $O(n)$ time. Furthermore, it is known that $r(S)$ tends to $\log_{|\Sigma|} n$ as n tends to infinity under a Bernoulli i.i.d. model (cf. Fici et al. [2006]). It should be clear that there is a trade-off between the value of d (utility of the z-RSDS) and the time taken by the binary search (due to the size of the interval).

5.2 Improvement II: Checking Prefixes of S

Lemma 14. *Let S be a string and P be a prefix of S. Further, let $\mathcal{A}_d(P)$ (respectively, $\mathcal{A}_d(S)$) be the set of strings that are consistent with $\mathcal{T}_d(P)$ (respectively, with $\mathcal{T}_d(S)$). It holds that $\alpha_d(P) \leq \alpha_d(S)$.*

Proof. We show the lemma by showing that for any string X and any letter a, $\alpha_d(X) \leq \alpha_d(Xa)$. This implies that $\alpha_d(P) \leq \alpha_d(S)$. Indeed, by Lemma 7, it follows that if X' is d-equivalent to X, then $X'a$ is d-equivalent to Xa. □

Lemma 14 lets us implement the check in line 4 of the z-RC algorithm by operating on the prefixes of S. The length of a longest substring of every prefix P of S occurring at least twice in P can be computed by means of the **longest previous factor (LPF)** array [Crochemore et al. 2013]. The LPF array gives, for each position i in S, the length of a longest substring occurring both at i and to the left of i in S. We can thus construct an array R, where $R[i]$ stores the length of a longest substring occurring at least twice in the prefix $P = S[0\,..\,i]$ of S, by traversing the LPF array. Then, we only need to perform the check $\alpha_d(P) \geq z$ when $d < R[i] + 2$. This is because of applying Improvement I on P. The LPF array, and thus array R, can be computed both in $O(n)$ time [Crochemore et al. 2013]. Note that $R[i] \leq R[i+1]$. Thus, having R, we can find (whether there exists) a prefix $P = S[0\,..\,i]$ satisfying $d < R[i] + 2$, for all d, in $O(n)$ time in total.

5.3 Improvement III: Sparse LU Decomposition

Let $G_{S,d} = (V_{S,d}, E_{S,d})$ be the weighted de Bruijn graph for which we must compute the determinant $\det L_{S,d}$. $L_{S,d}$ is a $|V_{S,d}| \times |V_{S,d}|$ non-singular matrix, where $|V_{S,d}|$ is the number of distinct substrings of length $d-1$ occurring in S. Hence, we have that $|V_{S,d}| \leq \min(|\Sigma|^{d-1}, n-d+1)$. If $|V_{S,d}| = O(n^{1/\omega})$, then $\det L_{S,d}$ is computed in $O(n)$ time by Lemma 11. If $|V_{S,d}| = \Theta(n)$, then $L_{S,d}$ is sparse: it has no more than $|V_{S,d}| + n - d + 1$ non-zero elements, because in the worst case there is a non-zero element for each edge and there are $n-d+1$ edges with multiplicity 1. Thus, in any case, $L_{S,d}$ cannot contain more than $2n-d+1$ non-zero elements. We can therefore employ highly optimized algorithms for sparse LU decomposition (e.g., Demmel et al. [1999]; Gilbert and Peierls [1988]) to compute $\det L_{S,d}$ efficiently. Let $\text{flops}(XY)$ be the number of multiplications of non-zero elements performed while computing the product XY by conventional matrix multiplication. The algorithm of Gilbert and Peierls [1988], for instance, takes $O(\text{flops}(LU)+m)$ time to compute the LU decomposition of a matrix with m non-zero elements. Thus, in our case, computing $\det L_{S,d}$ takes $O(\text{flops}(LU)+n)$ time.

6 IMPLEMENTATIONS AND EXPERIMENTS

6.1 Implementations

We have implemented the following algorithms in C++: (I) z-RC with Improvement III; (II) z-RCB (for Binary search interval reduction), which implements Improvements I and III; (III) z-RCE with Improvement III; and (VI) z-RCBP (for Binary search interval reduction and Prefix checking), which implements Improvements I, II, and III. Our implementation is available at https://bitbucket.org/reverse-safe-data-structures/rsds. We have omitted the results of the versions of the algorithms without Improvement III, as they were too slow to be practical.

For Improvement II, we have combined the idea described in Section 5.2 with exponential search: we start from an initial prefix P_0 of S that has length $|P_0| = \kappa$ and use it to perform the check in line 4 of our algorithm in Section 4. Due to Lemma 14, we know that $\alpha_d(P_0) \geq z$ implies $\alpha_d(S) \geq z$; we thus check if $\alpha_d(P_0) \geq z$, because this is clearly more efficient than checking $\alpha_d(S) \geq z$. If $\alpha_d(P_0) < z$, $\alpha_d(S) \geq z$ may or may not hold. In this case, we consider a longer prefix of S that has length $|P_1| = 2^1 \cdot \kappa$ and proceed similarly. Clearly, significant computational savings can be brought when the last considered prefix P_i has small length $|P_i| = 2^i \cdot \kappa$, whereas in the worst case $P_i = S$, and the total cost of our algorithm with Improvement II is twice the cost of the algorithm without it due to doubling. We also apply Improvement I on these prefixes: if $d \geq R[i]+2$ for prefix P_i, we do not check $\alpha_d(P_i) \geq z$, because Lemma 13 already ensures that $\alpha_d(P_i) = 1 < z$.

For Improvement III, we used the Sparse LU decomposition function of the open source Eigen library (v. 3.3.7) [Eigen Library 2020], which is based on the algorithm of Demmel et al. [1999], to compute $\det L_{S,d}$.

6.2 Experimental Setup and Datasets

We have evaluated z-RC, z-RCB, z-RCE, and z-RCBP in terms of data utility and efficiency. We do not compare our methods to existing approaches, as they are not alternatives to our work as mentioned in Section 1.

We used the following publicly available datasets: MSNBC (MSN), which contains page categories visited by users on msnbc.com over a 24-hour period; the complete genome of *Escherichia coli* (EC); KASANDR (KAS), which contains product categories in the Kelkoo price comparison site; and a dataset containing 27 Primate mitochondrial genomes (PR). MSN was used in the work of Gkoulalas-Divanis and Loukides [2011], Gwadera et al. [2013], and Loukides and Gwadera [2015], EC was used in the work of Bernardini et al. [2020a], was used in the work of Sidana et al. [2017], and PR was used in the work of Thankachan et al. [2017]. We also generated a uniformly random string of length 100M over an alphabet of size 10, and used its prefixes of length 1M, . . . , 100M as synthetic datasets, referred to as $\text{SYN}_{1\text{M}}, \ldots, \text{SYN}_{100\text{M}}$, respectively. Each dataset

Table 1. Characteristics of Datasets Used

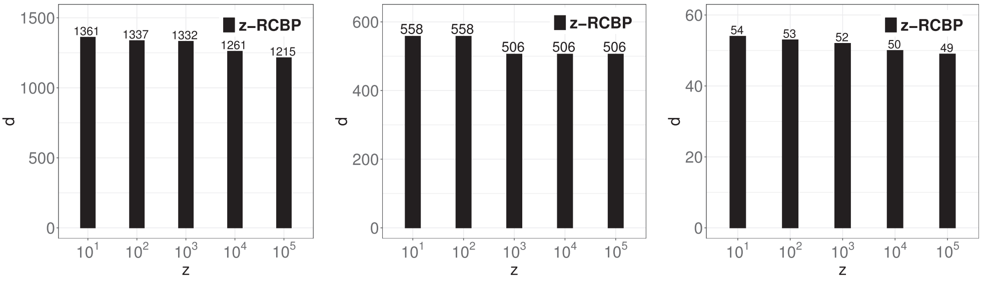
Dataset	**Data**	**Total**	**Alphabet**	$r(S)$
S	**Domain**	**Length** n	**Size** $\lvert\Sigma\rvert$	
MSN	Web	4,698,764	17	14,794
EC	Genomic	4,641,652	4	2,815
KAS	E-commerce	15,844,718	94	63
PR	Genomic	446,246 (27 strings)	4	$[13, 311]$
SYN_{100M}	Synthetic	100,000,000	10	16

contains a single string, except for PR, which contains 27 strings (one for each mitochondrial genome). In PR, we applied our methods to each string independently. Table 1 summarizes the characteristics of the datasets.

To evaluate data utility, we report the length d found by our methods for different values of z, and also investigate the accuracy of performing two classes of data analysis applications: *pattern mining* [Zaki et al. 2014] and *phylogenetic tree reconstruction* [Thankachan et al. 2017]. Unlike decision and counting pattern matching queries of length at most d, which are answered exactly using the z-RSDS constructed by our methods, these applications are not guaranteed to be performed accurately on the output encoding. Yet, we show that plugging in our approach gives insignificant or no data utility loss in these applications. We now discuss each of these applications.

(Closed) frequent pattern mining. Frequent patterns and closed frequent patterns in string datasets model knowledge that aids decision making [Arimura and Uno 2007; Shang et al. 2016] and can be used for data classification and clustering [Zaki et al. 2014]. Given a string S and a user-specified threshold minSup, a pattern is *frequent* if its relative frequency in S, also referred to as *support*, is at least minSup. A frequent pattern of S is *closed* if none of its superstrings has the same relative frequency in S. Closed frequent patterns are typically fewer than the frequent ones, and they are mined much more efficiently. Their benefit is that they uniquely determine the set of frequent patterns and their exact frequency. Our methods allow mining the frequent and closed frequent patterns of length at most d and only those. Thus, our methods preserve data utility well when the d computed is sufficiently large for low minSup values. In our experiments, we used the algorithm of Arimura and Uno [2007] to mine a more general class of frequent and closed frequent patterns having up to $w \in [0, 3]$ occurrences of a wildcard letter $\diamond$. A pattern with wildcards occurs in a string S if it is a subtring of S after replacing the wildcard letters with alphabet letters (e.g., pattern $\mathsf{a}\diamond\diamond\mathsf{e}$ occurs in $S = \mathsf{b}\underline{\mathsf{abde}}\mathsf{b}$). Mining patterns with wildcards poses a further challenge to our approach, since (closed) frequent patterns with wildcards are a superset of the (closed) frequent patterns and are typically longer.

Phylogenetic tree reconstruction. A phylogenetic tree illustrates the evolutionary relationships among a set of species. In our context, each species is represented by a different string in a collection of strings. To reconstruct phylogenetic trees, we applied the methodology of Thankachan et al. [2017] on the PR dataset, the only dataset comprised of a collection of strings. In other words, we compute the pairwise **Average Common Substring with** k **mismatches (**k**-ACS)** distance [Ulitsky et al. 2006; Leimeister and Morgenstern 2014] between the 27 strings in PR, using the ALFRED-G [Thankachan et al. 2017] algorithm, and then apply the neighbor-joining algorithm [Saitou and Nei 1987] to reconstruct the phylogenetic tree. We apply the methodology to S and to $S' \sim_d S$, $S' \neq S$: intuitively, data utility is preserved well when the phylogenetic tree for S is similar to the one for S'. Following Thankachan et al. [2017], we measured similarity using the **normalized Robinson-Foulds (nRF)** distance [Robinson and Foulds 1981].



(a) (b) (c)

Fig. 4. Length d for different z values for EC (a), MSN (b), and KAS (c). The length d for each z and dataset also appears on top of each bar.

Unless otherwise stated, we used $z = 100$ and $\kappa = 1,000$. We compiled all implementations using gcc version 7.3.0 with the -O3 and -msse3 flags and executed them on a machine with an Intel Xeon E5-2640 at 2.66 GHz and 160 GB of RAM.

6.3 Data Utility

Recall that our approach allows for answering pattern matching queries of length at most d in optimal time, and at the same time it prevents the reconstruction of the original dataset. In this section, we demonstrate that z-RCBP, and z-RC, z-RCB, which by design create the same output as z-RCBP, allow for other meaningful data analysis tasks to be applied with insignificant or no utility loss.

6.3.1 Length d. We first show that z-RCBP provides access to very long substrings of the original dataset (i.e., the output length d is large). Figure 4(a), (b), and (c) show d for different values of the privacy threshold z in EC, MSN, and KAS, respectively. As expected, d decreases when z increases. However, d is still very large even when z is set to 100,000. Specifically, d is in the order of several hundreds for EC and MSN and around 50 for KAS. This implies (I) no accuracy loss for applying the pattern matching queries described in Section 3 on very long substrings and (II) strong privacy against dataset reconstruction.

6.3.2 Frequent Pattern Mining. We demonstrate that z-RCBP allows for accurately mining frequent and closed frequent patterns with up to $w \in [0, 3]$ wildcard letters at very low minSup values. To this aim, we have computed the *smallest possible* value of minSup such that the mined frequent and closed frequent patterns have length no more than d. We denote this value by τ. Clearly, our method has no data utility loss for any minSup $\geq \tau$. For EC, the smallest such minSup value (up to eight decimal digits) was $\tau = 1.8 \cdot 10^{-6}$. Figure 5(a) and (b) show the number and the maximal length of the mined patterns with minSup $= \tau = 1.8 \cdot 10^{-6}$ for EC. For MSN, the smallest such minSup value (up to four decimal digits) was $\tau = 3.1 \cdot 10^{-3}$. The results for mining MSN with minSup $= \tau = 3.1 \cdot 10^{-3}$ in Figure 6(a) and (b). For KAS, the smallest such minSup value (up to eight decimal digits) was $1.6 \cdot 10^{-6}$. Figure 7(a) and (b) show the number and the maximal length of the mined patterns with minSup $= \tau = 1.6 \cdot 10^{6}$ for KAS. The plots in Figures 5, 6, and 7 show that a large number of (potentially interesting) patterns can still be mined from the randomly selected S', even if some of them occur a small number of times in S (since τ was very low). Thus, our method permits the fundamental task of frequent pattern mining to be performed accurately.

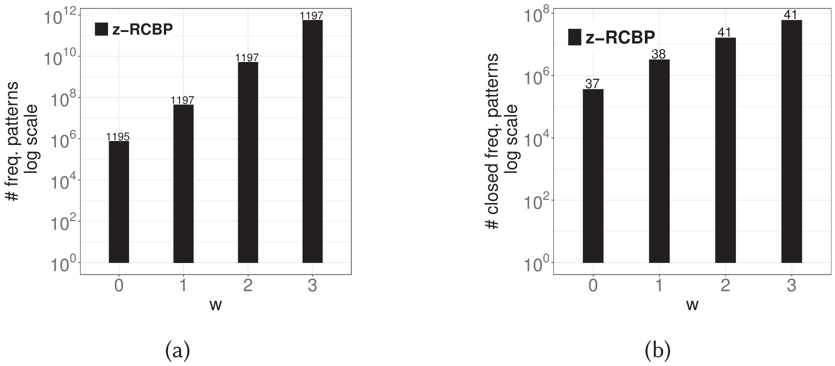


Fig. 5. Number of frequent patterns (a) and closed frequent patterns (b) with up to $w \in [0,3]$ wildcards mined from EC using minSup $= 1.8 \cdot 10^{-6}$. The length of the longest mined pattern is on the top of each bar.

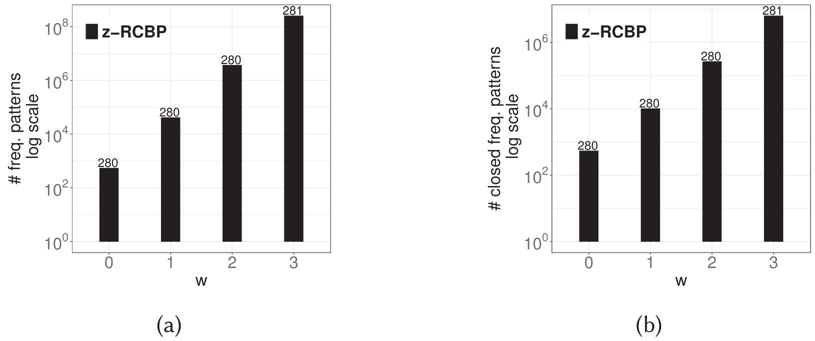
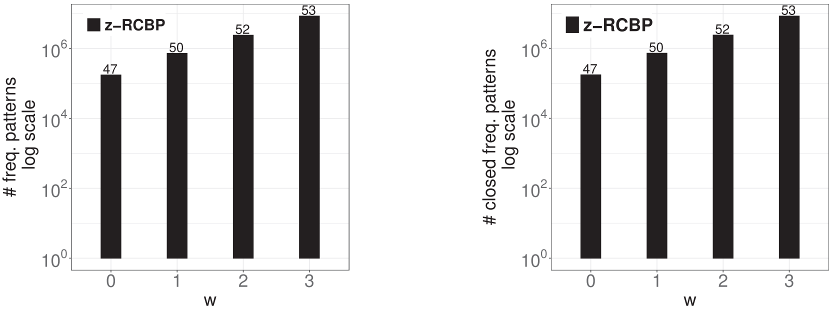


Fig. 6. Number of frequent patterns (a) and closed frequent patterns (b) with up to $w \in [0,3]$ wildcards mined from MSN using minSup $= 3.1 \cdot 10^{-3}$. The length of the longest mined pattern is on the top of each bar.

6.3.3 Phylogenetic Tree Reconstruction. We next demonstrate that z-RCBP leads to phylogenetic trees constructed from $S' \sim_d S$, $S' \neq S$, which are either the same or very similar with respect to the nRF distance to the phylogenetic trees constructed from S. Figure 8 shows the nRF distance between these trees. The trees were obtained using the k-ACS distance for different k values in $[0,9]$ and the neighbor-joining algorithm as in the work of Thankachan et al. [2017]. Note that the tree constructed from S was the same to the one constructed from S' in 6 out of 10 cases, implying no data utility loss, and in the remaining 4 cases, the nRF had a very small value of 0.04, implying insignificant data utility loss for this fundamental bioinformatics task.

6.4 Runtime

In this section, we show that, despite the n^{ω} factor, z-RCBP takes only a few minutes to finish for million-letter texts. Figure 9(a) shows the runtime of z-RC, z-RCB, z-RCE, and z-RCBP using the synthetic datasets as input. Recall that the largest synthetic dataset is $\text{SYN}_{100\text{M}}$ and the other datasets are prefixes of $\text{SYN}_{100\text{M}}$. z-RCBP was substantially more efficient than all other algorithms and scaled better with the dataset size, confirming the necessity of Improvements I and II for being able to apply our methodology to large texts. However, z-RC was the slowest, and it did not finish within 48 hours for $\text{SYN}_{10\text{M}}, \ldots, \text{SYN}_{100\text{M}}$. Note that z-RCE was more efficient than z-RC, since it



(a)

(b)

Fig. 7. Number of frequent patterns (a) and closed frequent patterns (b) with up to $w \in [0, 3]$ wildcards mined from KAS using minSup $= 1.6 \cdot 10^{-6}$. The length of the longest mined pattern is on the top of each bar.

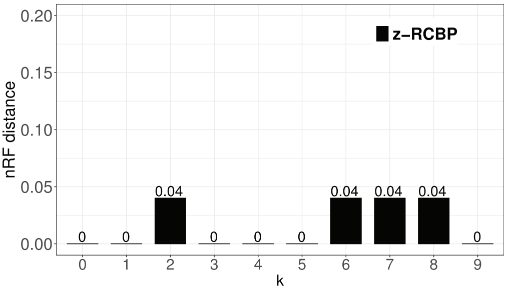


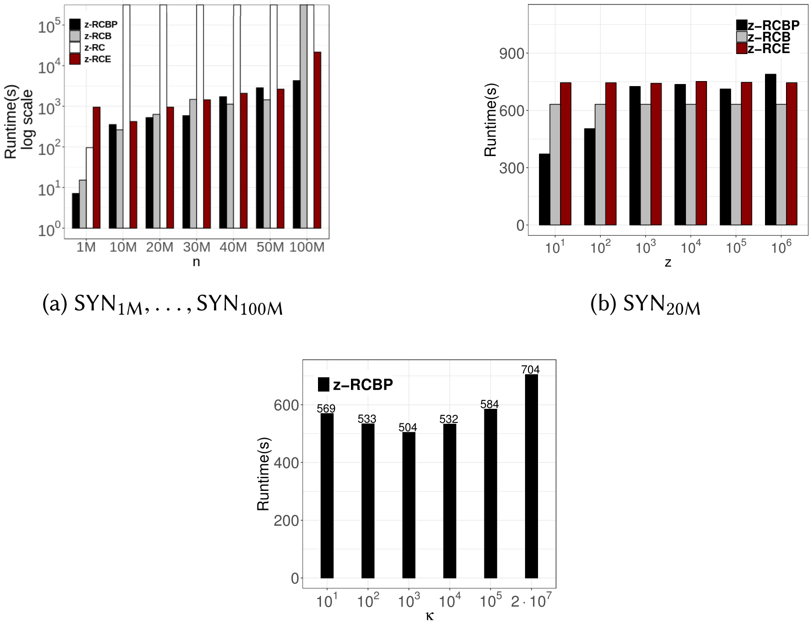
Fig. 8. nRF distance vs. k between phylogenetic trees constructed for S and for $S' \sim_d S$, $S' \neq S$, using the k-ACS distance in PR.

performs fewer iterations. The reason is that $d < 20$ for these datasets. In addition, z-RCE was comparable to z-RCB. The reason is that d was very close to $r(S) + 1$.

We also measured the runtime of z-RCB, z-RCE, and z-RCBP for different z values (see Figure 9(b)). We do not report the runtime of z-RC because it did not finish within 48 hours. The runtime of z-RCBP is much less when z is small, because z-RCBP considered fairly short prefixes. Specifically, z-RCBP was two times faster than z-RCB on average, and three times faster when $z = 10$. The runtime of z-RCB and of z-RCE was not affected substantially by z. This is because these algorithms output the same d as z-RCBP for all z values (i.e., they construct the same output) but operate on the entire string S. z-RCB was comparable to z-RCE for the reason explained earlier.

Next, we studied the impact of the initial prefix length κ on the runtime of z-RCBP, the only method that uses Improvement II (see Figure 9(c)). The runtime of z-RCBP decreased when κ increased, but only up to $\kappa = 1,000$. Until then, prefixes were too short (i.e., the condition $\alpha_d(S') \geq z$ did not hold), so longer prefixes were considered. For $\kappa > 1,000$, z-RCBP took more time because it is more expensive to check the condition on longer prefixes (e.g., z-RCBP took 40% more time when $\kappa = |S|$ compared to when $\kappa = 1,000$).

Similar results were observed for the other datasets. For example, as can be seen in Table 2, in the case of MSN and KAS, z-RCBP was again the fastest, z-RC was the slowest, and z-RCB and z-RCE performed similarly. In the case of the EC dataset, however, z-RCE was the slowest. The reason is that EC led to a larger d than that of MSN, so z-RCE had to perform more iterations.



(a) $\mathrm{SYN}_{1\mathrm{M}}, \ldots, \mathrm{SYN}_{100\mathrm{M}}$

(b) $\mathrm{SYN}_{20\mathrm{M}}$

(c) $\mathrm{SYN}_{20\mathrm{M}}$

Fig. 9. Runtime vs. n (a) (z-RC did not finish within 48 hours for $\mathrm{SYN}_{10\mathrm{M}}, \ldots, \mathrm{SYN}_{100\mathrm{M}}$ and z-RCB did not finish within 48 hours for $\mathrm{SYN}_{100\mathrm{M}}$). Runtime vs. z (b) and κ (c).

Table 2. Runtime (in Seconds) for All Implementations

Dataset	z-RCB	z-RCE	z-RC	z-RCBP
MSN	438.49	421.96	659.17	**347.34**
EC	364.84	725.26	571.8	**339.18**
KAS	710.55	1022.59	2555.8	**649.09**

Note: The minimum is set in bold.

6.5 Memory Usage

In this section, we report the peak memory usage of all implementations. Figure 10 shows these results when the synthetic datasets (i.e., prefixes of $\mathrm{SYN}_{100\mathrm{M}}$) are given as input. All implementations required a larger amount of memory when they were applied to longer strings. In addition, the amount of required memory was similar for all implementations. This is because most of the memory is needed for computing $\det L_{S,d}$ (i.e., the sparse LU decomposition in Improvement III, which is employed by all algorithms). For example, the $\det L_{S,d}$ computations required approximately 90 GB out of the 96 GB of memory that were needed when z-RCE was applied to $\mathrm{SYN}_{100\mathrm{M}}$. Similar results were observed for the other datasets, as can be seen in Table 3.

6.6 Disregarded Prefixes

Last, we demonstrate that applying Improvement I on the prefixes of S, which are used in Improvement II, allows for disregarding a large ratio of them from the computation. In other words, we often avoid computing $\alpha_d(P)$ for a prefix P of S, because when $d \geq r(P) + 2$, we have that $\alpha_d(P) = 1 < z$ by Lemma 13. Specifically, Figure 11 shows that the ratio of disregarded prefixes

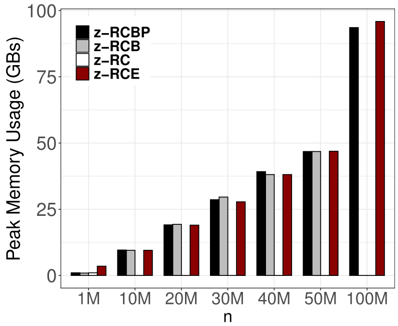


Fig. 10. Peak memory usage (in GB) vs. n (we do not report the results for z-RC for SYN$_{10M}$, . . . , SYN$_{100M}$ and for z-RCB for SYN$_{100M}$, as these implementations did not finish within 48 hours).

Table 3. Peak Memory Usage (in GB) for All Implementations

Dataset	z-RCB	z-RCE	z-RC	z-RCBP
MSN	15.3	15.3	15.3	15.4
EC	32.9	32.9	32.9	33.0
KAS	15.3	15.3	15.3	15.7

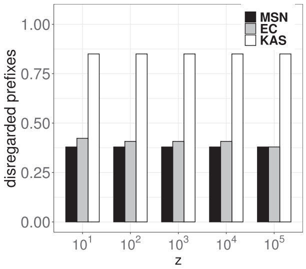


Fig. 11. Ratio of disregarded prefixes vs. z in MSN, EC, and KAS.

over all prefixes considered for MSN and EC is at least 0.38, whereas that for KAS is 0.85. The benefit of the improvement when this ratio is large is time efficiency, since computing $\alpha_d(P)$ to check whether $\alpha_d(P) \geq z$ can be expensive particularly for a long prefix P of S.

7 APPLICATION TO ADVERSARY MODELS

In this section, we discuss adapted versions of our z-RSDS that can be applied to two different adversary models, in which an adversary possesses positive and negative knowledge, respectively. In the context of z-anonymity, positive adversarial knowledge has been considered in the work of Loukides et al. [2010], Mohammed et al. [2010], Poulis et al. [2014], Terrovitis and Mamoulis [2008], and Terrovitis et al. [2011], and negative adversarial knowledge in the work of Atzori et al. [2008] and Li and Li [2008]. Unlike our work, none of these works considers adversarial knowledge in the context of a string.

7.1 Adversary Model I: Positive Adversarial Knowledge

Our privacy goal is to limit the probability of inferring string S when the adversary possesses the following knowledge.

Definition 15 (Positive Adversarial Knowledge). A pair $\mathcal{K} = (\mathcal{T}_d(S'), \tilde{S})$, where $S' \sim_d S$ and $\tilde{S}$ is a (possibly empty) substring of S.

The adversarial knowledge $\mathcal{K}$ is comprised of $\mathcal{T}_d(S')$, which is accessible by the adversary, and of $\tilde{S}$ which is the adversary's *background knowledge*. Background knowledge is obtained by an adversary, typically from external data sources and/or communication with individuals represented in the input dataset [Terrovitis et al. 2011; Loukides et al. 2010; Poulis et al. 2014; Mohammed et al. 2010; Terrovitis and Mamoulis 2008]. As it will become clear later, such knowledge may make a z-RSDS more likely to be reversed. Thus, when certain background knowledge is known or can be assumed, it should be modeled and taken into account in the construction of a z-RSDS to ensure that the z-RSDS remains sufficiently unlikely to reverse.

We model the background knowledge as a substring to capture manifested attacks [Loukides et al. 2010; Terrovitis and Mamoulis 2008] in which the adversary observes an individual's actions within a time frame. The actions are represented by $\tilde{S}$. For example, when S models the diagnoses in an individual's electronic health record, $\tilde{S}$ models the diagnoses assigned to the individual during a hospital visit, which may be known by a hospital employee [Loukides et al. 2010]. Similarly, when S models an individual's credit card purchases, $\tilde{S}$ models the products purchased by the individual during a visit to a shop, which may be known by a shop employee [Terrovitis and Mamoulis 2008]. $\tilde{S}$ may be specified by the data provider [Bernardini et al. 2020a] or the data custodian [Terrovitis et al. 2017], according to policies. Note that from $\mathcal{T}_d(S')$, the adversary can also learn the following (see Figure 1): the length $n = |S|$, the maximal string depth d, and the suffixes of S of length at most $d - 1$. Thus, we did not include such information in $\mathcal{K}$ explicitly.

An adversary may not be able to uniquely infer S, based on their knowledge $\mathcal{K}$. This is because they have to distinguish S among the set of strings that are d-equivalent to S and have $\tilde{S}$ as substring. In fact, the probability that the adversary infers S, based solely on their knowledge $\mathcal{K}$, is defined as follows.

Definition 16 (Inference Probability of S). The *inference probability* of S, based on the knowledge $\mathcal{K}$, is defined as $\mathcal{P}(I_S \mid \mathcal{K}) = {}^{1}/_{|\mathcal{A}_{\mathcal{K}}|}$, where I_S is the event "the adversary infers S" and $\mathcal{A}_{\mathcal{K}}$ is the set of strings consistent with $\mathcal{T}_d(S')$ having $\tilde{S}$ as substring.

$\mathcal{P}(I_S \mid \mathcal{K})$ is defined based on the following: (I) the fact that the adversary can construct all strings that are consistent with $\mathcal{T}_d(S')$ and contain $\tilde{S}$ as substring (see Section 7.1.1), and (II) the *random worlds assumption* [Bacchus et al. 1996] (i.e., each such instance is equally likely). This assumption is followed by most related works (see the work of Xiao et al. [2010] and references therein).

We aim at constructing a $\mathcal{T}_d(S')$, for some $S' \sim_d S$ chosen at random, that an adversary cannot use to infer S with sufficiently large $\mathcal{P}(I_S \mid \mathcal{K})$. We also require $\mathcal{T}_d(S')$ to have maximal d to support the operations discussed in Section 3 on larger substrings, thereby providing higher utility. This leads to the following computational problem.

Problem 2. *Given a string S of length n, a substring $\tilde{S}$ of S, and a privacy threshold $1 < z \leq n^c$, for some constant $c \geq 1$, construct a $\mathcal{T}_d(S')$ such that (I) $S' \sim_d S$, (II) d is maximal, and (III) $\mathcal{P}(I_S \mid \mathcal{K}) \leq \frac{1}{z}$, for $\mathcal{K} = (\mathcal{T}_d(S'), \tilde{S})$; otherwise, output FAIL if no such d exists.*

7.1.1 Construction Algorithm. We next show how the z-RC algorithm for constructing a z-RSDS can be applied in an extended way to solve Problem 2. In this case, we need to account for $\mathcal{A}_{\mathcal{K}}$, a modified version of $\mathcal{A}_d$: a string S' is in $\mathcal{A}_\mathcal{K}$ if and only if it is d-equivalent to S and contains $\tilde{S}$ as a substring. In the graph formulation of the problem, we need to ensure that a path representing $\tilde{S}$ must always be visited. Thus, we modify the z-RC algorithm as follows.

Let $\alpha_\mathcal{K} = |\mathcal{A}_\mathcal{K}|$. Consider a binary search iteration for some value of d, in which we must check whether $\alpha_\mathcal{K} \geq z$. We first construct the weighted de Bruijn graph $G_{S,d}$. If $|\tilde{S}| \leq d$, all strings in $\mathcal{A}_d$ contain $\tilde{S}$ as a substring by construction and so we do not need to modify the algorithm for such an $\tilde{S}$. Intuitively, in this case, knowledge of $\tilde{S}$ is of no use to the adversary. We thus consider the case when $|\tilde{S}| > d$. A substring $\tilde{S}$ of length $|\tilde{S}|$ is represented by a path $v_1 v_2 \ldots v_h$ in $G_{S,d}$, with $h = |\tilde{S}| - d + 2$, as the first node v_1 represents a prefix of $|\tilde{S}|$ of length $d-1$, and each traversed edge appends one letter to this prefix. We remove the edges of a path $v_1 v_2 \ldots v_h$ in $G_{S,d}$ representing *some occurrence* of $\tilde{S}$ in S, $|\tilde{S}| > d$. (There may be multiple such paths.) We add a *shortcut edge* $e_{\tilde{S}} = (v_1, v_h)$ directed from node v_1 to node v_h to represent an occurrence of string $\tilde{S}$. We refer to this procedure as *collapsing* the path $v_1 v_2 \ldots v_h$. Let us denote the resulting graph by $G_{S,d,\tilde{S}}$ (see Figure 12 vs. Figure 3).

Lemma 17. *(a) If $S \sim_d S'$ and $\tilde{S}$ is a substring of S', then S' corresponds to an Eulerian path in $G_{S,d,\tilde{S}}$ that starts from vertex s and ends at vertex $t \neq s$ (if $s = t$, then it corresponds to an Eulerian cycle starting from s). (b) $\alpha_\mathcal{K}$ is equal to the number of non-equivalent Eulerian paths (respectively, cycles) in $G_{S,d,\tilde{S}}$.*

Proof. (a) Trivial. (b) We first observe that $G_{S,d,\tilde{S}}$ is Eulerian (respectively, semi-Eulerian) by construction, because $G_{S,d}$ is Eulerian (respectively, semi-Eulerian) and the preceding procedure does not change the parity of any vertex. Indeed, consider path $v_1 v_2 \ldots v_h$ in $G_{S,d}$ representing the string $\tilde{S}$, which we replace with a shortcut edge $e_{\tilde{S}}$. Exactly one outgoing and one incoming edge is removed from each $v_2, \ldots, v_{h-1}$; one outgoing edge is removed from v_1 and replaced with outgoing edge $e_{\tilde{S}}$, and one incoming edge is removed from v_h and replaced with $e_{\tilde{S}}$ incoming. Moreover, since the multiplicity of any substring U of length d is given by the multiplicity of the edge from node $U[0\,..\,d-2]$ to node $U[1\,..\,d-1]$, the multiplicities of d-substrings are not affected by this transformation.

To show that the number of non-equivalent Eulerian paths (respectively, cycles) in $G_{S,d,\tilde{S}}$ is at most $\alpha_\mathcal{K}$, consider any Eulerian path (respectively, cycle) in $G_{S,d,\tilde{S}}$. By definition, there is at least one occurrence of $\tilde{S}$ given by edge $e_{\tilde{S}}$, and it thus represents a string that belongs to $\mathcal{A}_\mathcal{K}$. Symmetrically, to show that $\alpha_\mathcal{K}$ is at most the number of non-equivalent Eulerian paths (respectively, cycles) in $G_{S,d,\tilde{S}}$, consider a string $U \in \mathcal{A}_\mathcal{K}$. Among the (possibly multiple) Eulerian paths (respectively, cycles) in $G_{S,d}$ that represent U, consider one that has $v_1 v_2 \ldots v_h$ as subpath as representative of its equivalence class: such path exists because of Observation 8 and the properties of weighted de Bruijn graphs. This path corresponds to the path in $G_{S,d,\tilde{S}}$, where $v_1 v_2 \ldots v_h$ is replaced with $v_1 v_h$. □

Theorem 18. *Problem 2 can be solved in $O(n^\omega \log d)$ time.*

Proof. The correctness of the construction algorithm follows by Lemma 17 and the fact that it is correct to apply binary or exponential search due to the monotonicity of $\alpha_\mathcal{K}$ (the monotonicity proof is almost identical to that of Lemma 5 and is thus omitted).

For a given d, finding a path $v_1 v_2 \ldots v_h$ in $G_{S,d}$ and replacing it with $v_1 v_h$ can be done while constructing $G_{S,d}$ at no extra cost. Recall that $v_1 v_2 \ldots v_h$ represents some occurrence of string $\tilde{S}$ in S, and that all occurrences of $\tilde{S}$ in S can be found in $O(n)$ time using the suffix tree of S [Weiner 1973]. The time complexity thus follows from the proof of Theorem 1. □

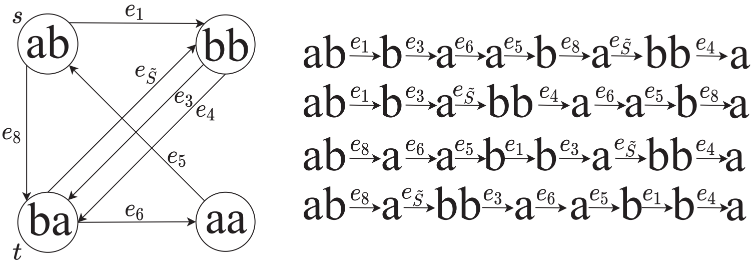

s
ab
e_1
bb
$e_{\tilde{S}}$
e_3
e_4
e_8
e_5
ba
e_6
aa
t
$\mathrm{ab}\xrightarrow{e_1}\mathrm{b}\xrightarrow{e_3}\mathrm{a}\xrightarrow{e_6}\mathrm{a}\xrightarrow{e_5}\mathrm{b}\xrightarrow{e_8}\mathrm{a}\xrightarrow{e_{\tilde{S}}}\mathrm{bb}\xrightarrow{e_4}\mathrm{a}$
$\mathrm{ab}\xrightarrow{e_1}\mathrm{b}\xrightarrow{e_3}\mathrm{a}\xrightarrow{e_{\tilde{S}}}\mathrm{bb}\xrightarrow{e_4}\mathrm{a}\xrightarrow{e_6}\mathrm{a}\xrightarrow{e_5}\mathrm{b}\xrightarrow{e_8}\mathrm{a}$
$\mathrm{ab}\xrightarrow{e_8}\mathrm{a}\xrightarrow{e_6}\mathrm{a}\xrightarrow{e_5}\mathrm{b}\xrightarrow{e_1}\mathrm{b}\xrightarrow{e_3}\mathrm{a}\xrightarrow{e_{\tilde{S}}}\mathrm{bb}\xrightarrow{e_4}\mathrm{a}$
$\mathrm{ab}\xrightarrow{e_8}\mathrm{a}\xrightarrow{e_{\tilde{S}}}\mathrm{bb}\xrightarrow{e_3}\mathrm{a}\xrightarrow{e_6}\mathrm{a}\xrightarrow{e_5}\mathrm{b}\xrightarrow{e_1}\mathrm{b}\xrightarrow{e_4}\mathrm{a}$

Fig. 12. $G_{S,d,\tilde{S}}$ with S = abaabbabba, $d = 3$, $\tilde{S}$ = babb, and $\alpha_{\mathcal{K}} = 4$.

7.2 Adversary Model II: Negative Adversarial Knowledge

In this section, we consider the case where an adversary possesses negative knowledge. The adversary model is somewhat dual to the one defined in Section 7.1. Specifically, in our model, the adversary possesses the following *negative* knowledge.

Definition 19 (Negative Adversarial Knowledge). A pair $\bar{\mathcal{K}} = (\mathcal{T}_d(S'), R)$, where $S' \sim_d S$ and R is a non-empty string that does not occur in S.

It should be clear that the background knowledge in Definition 19 may be used to reverse the z-RSDS, and it should therefore be modeled and taken into account in the construction of a z-RSDS. A procedure that is entirely analogous to the one described in Section 7.1.1 can be used to ensure that the z-RSDS remains sufficiently unlikely to reverse, as we explain next.

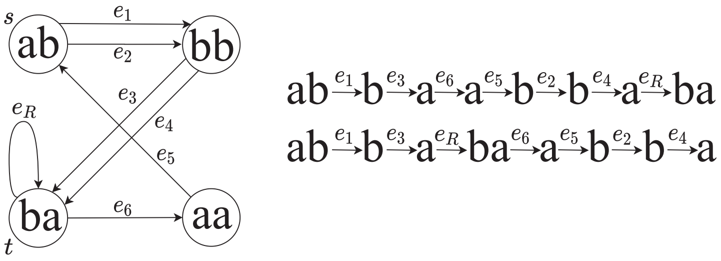
We now have to account for $\mathcal{A}_{\bar{\mathcal{K}}}$, which is again a modified version of $\mathcal{A}_d$: a string S' is in $\mathcal{A}_{\bar{\mathcal{K}}}$ if and only if it is d-equivalent to S and does not contain R as a substring. The inference probability $\mathcal{P}(I_S \mid \bar{\mathcal{K}})$, based on the negative knowledge $\bar{\mathcal{K}}$, is defined in much the same way as the one for positive adversarial knowledge: $\mathcal{P}(I_S \mid \bar{\mathcal{K}}) = {}^{1}/_{|\mathcal{A}_{\bar{\mathcal{K}}}|}$. Once again, the problem is to construct a $\mathcal{T}_d(S')$ such that $S' \sim_d S$, d is maximal and $\mathcal{P}(I_S \mid \bar{\mathcal{K}}) \leq {}^{1}/_{z}$.

Problem 3. *Given a string S of length n, a string R that does not occur in S, and a privacy threshold $1 < z \leq n^c$, for some constant $c \geq 1$, construct a $\mathcal{T}_d(S')$ such that (I) $S' \sim_d S$, (II) d is maximal, and (III) $\mathcal{P}(I_S \mid \bar{\mathcal{K}}) \leq \frac{1}{z}$, for $\bar{\mathcal{K}} = (\mathcal{T}_d(S'), R)$; otherwise, output FAIL if no such d exists.*

Let $\alpha_{\bar{\mathcal{K}}} = |\mathcal{A}_{\bar{\mathcal{K}}}|$, and consider a binary search iteration for some value of d, in which we must check whether $\alpha_{\bar{\mathcal{K}}} \geq z$. We construct a graph $G_{S,d,R}$ exactly as if R was required to occur in S', as described in Section 7.1.1. By definition, all strings corresponding to non-equivalent Eulerian paths in $G_{S,d,R}$ contain at least one occurrence of R: the complement of such set in $\mathcal{A}_d$ is the set of d-equivalent strings that do not contain any occurrence of R, which is precisely what we aim at. The following lemma thus connects $\alpha_{\bar{\mathcal{K}}}$ with the number of non-equivalent Eulerian paths (respectively, cycles) in $G_{S,d}$ and $G_{S,d,R}$. The proof is similar to the one of Lemma 17 and is therefore omitted.

Lemma 20. *$\alpha_{\bar{\mathcal{K}}}$ is equal to the difference between the number α_d of non-equivalent Eulerian paths (respectively, cycles) in $G_{S,d}$ and the number $\alpha_{\mathcal{K}}$ of non-equivalent Eulerian paths (respectively, cycles) in $G_{S,d,R}$.*

Notice that, in contrast with the case of positive adversarial knowledge, it may happen that R is not represented by any path in $G_{S,d}$: in this case, we simply set $\alpha_{\bar{\mathcal{K}}} = \alpha_d$, as no string $S' \sim_d S$ has R as a substring. Figure 13 illustrates an example where R is actually represented in $G_{S,d}$: the four strings $S' \sim_d S$ that do not have R as a substring are, in fact, the ones depicted in Figure 12. The following theorem can be proved much the same way as Theorem 18.



$$\mathrm{ab}\xrightarrow{e_1}\mathrm{b}\xrightarrow{e_3}\mathrm{a}\xrightarrow{e_6}\mathrm{a}\xrightarrow{e_5}\mathrm{b}\xrightarrow{e_2}\mathrm{b}\xrightarrow{e_4}\mathrm{a}\xrightarrow{e_R}\mathrm{ba}$$
$$\mathrm{ab}\xrightarrow{e_1}\mathrm{b}\xrightarrow{e_3}\mathrm{a}\xrightarrow{e_R}\mathrm{ba}\xrightarrow{e_6}\mathrm{a}\xrightarrow{e_5}\mathrm{b}\xrightarrow{e_2}\mathrm{b}\xrightarrow{e_4}\mathrm{a}$$

Fig. 13. $G_{S,d,R}$ with S = abaabbabba, $d = 3$, R = baba, and so $\alpha_{\bar{\mathcal{K}}} = \alpha_d - \alpha_{\mathcal{K}} = 6 - 2 = 4$.

Theorem 21. *Problem 3 can be solved in $O(n^{\omega} \log d)$ time.*

7.3 Generalization to a Collection of Patterns

Both the model of positive and the model of negative adversarial knowledge can be straightforwardly extended to the case of multiple disjoint pattern occurrences in S. We begin by defining the notion of disjoint pattern occurrences in S, as follows.

Definition 22 (Disjoint Pattern Occurrences). Given a string S and a pair of strings (S_1, S_2), we call (i_1, i_2) *disjoint pattern occurrences* of S_1 and S_2 in S, when S_1 occurs at position i_1 in S, S_2 occurs at position i_2 in S, and the intervals $[i_1, i_1 + |S_1| - 1]$ and $[i_2, i_2 + |S_2| - 1]$ are disjoint.

Formally, in the positive adversarial knowledge model, the adversary possesses the following knowledge: a collection $\mathcal{S}$ of k patterns such that there exists an ordering $S_1, S_2, \ldots, S_k$ of these patterns that forms a sequence of pairwise disjoint pattern occurrences in $\mathcal{S}$.

The case of multiple disjoint pattern occurrences we consider is of practical importance. It can correspond to a collection of adversary observations within multiple disjoint time frames; each time frame corresponds to a pattern, and thus the patterns satisfy Definition 22. In fact, the examples of attacks discussed in Section 7.1 naturally generalize to multiple patterns. For example, S_i can model the diagnoses assigned to an individual during the i-th hospital visit of the individual.

It can be readily verified that a solution for the case when an adversary possesses a collection $\mathcal{S}$ can be obtained by repeating the procedure of collapsing patterns in the de Bruijn graph to obtain $G_{S,d,\mathcal{S}}$: this is possible because all such patterns are edge-wise disjoint, thanks to the condition that any two patterns have disjoint pattern occurrences in S. In the example of Figure 14, the knowledge of two non-overlapping substrings of S is sufficient to uniquely identify S, thus to make the construction algorithm fail for any $z > 1$. Note that in this example, the adversary knows an additional string S_2 compared to the example of Figure 12, and this additional knowledge makes the construction algorithm fail.

The negative adversarial knowledge model can be extended in a similar way. In this case, the adversary possesses the following knowledge: a collection of patterns $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ such that no permutation of these patterns occurs in S with pairwise disjoint occurrences. Since this condition is precisely complementary to the positive knowledge for multiple patterns, we follow the same strategy as for the negative knowledge model for one pattern: we construct the graph $G_{S,d,\mathcal{R}}$, compute the number of non-equivalent Eulerian paths in this graph, and subtract it from α_d.

8 A z-RSDS FOR DECISION QUERIES

Let us recall that a non-empty string R that does not occur in a string S is called *absent* from S, and it is called *minimal absent* if furthermore all proper substrings of R do occur at least once

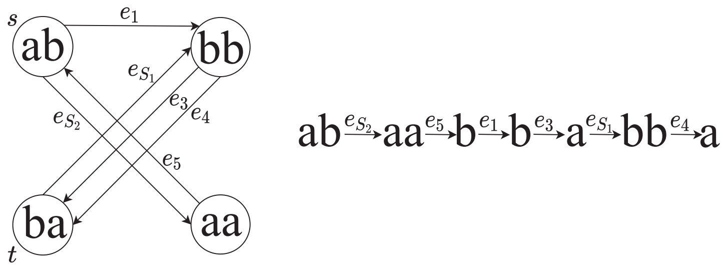

s
ab
e_1
bb
e_{S_1}
e_3
e_4
e_{S_2}
e_5
ba
aa
t
ab $\xrightarrow{e_{S_2}}$ aa $\xrightarrow{e_5}$ b $\xrightarrow{e_1}$ b $\xrightarrow{e_3}$ a $\xrightarrow{e_{S_1}}$ bb $\xrightarrow{e_4}$ a

Fig. 14. G_{S,d,S_1,S_2} with S = abaabbabba, $d = 3$, S_1 = babb, S_2 = abaa, and so the only Eulerian path in the graph spells S itself.

in S. **Minimal absent words (MAWs)** are used in many applications [Silva et al. 2015; Pratas and Silva 2020; Fici et al. 2006; Garcia et al. 2011; Chairungsee and Crochemore 2012; Ota and Morita 2010; Crochemore et al. 2000] and their theory is well developed [Mignosi et al. 2002; Fici and Gawrychowski 2019; Fici et al. 2019], also from an algorithmic and data structure point of view [Ayad et al. 2019; Barton et al. 2014, 2015; Charalampopoulos et al. 2018a, b; Crochemore et al. 1998, 2020; Fujishige et al. 2016; Mieno et al. 2020]. For example, it is well known that, given two strings S and S', one has $S = S'$ if and only if S and S' have the same set of MAWs [Mignosi et al. 2002].

We now prove that all strings in the same d-equivalence class have precisely the same set of MAWs of length up to d (and therefore any two distinct d-equivalent strings must have some longer MAW not in common).

Lemma 23. *Let S be a string and R a MAW of S of length at most d. If S' is d-equivalent to S, then R is a MAW of S'.*

Proof. Let us first assume that the length of R is exactly d. Given a de Bruijn graph G of order d, we call edge (u, v) *fake-feasible* if it does not exist in G but it *could* exist in G (i.e., if u and v overlap by $d-2$ letters). It holds that a string R of length d is a MAW of S if and only if $R = u \cdot v[d-2]$ and (u, v) is a fake-feasible edge in the de Bruijn graph G of order d of string S. This follows by the definition of MAWs: all proper substrings of R (in particular, its longest proper prefix u and suffix v) do occur in S but R does not. Since all strings in the d-equivalence class of S have the same G, it follows that they have the same set of fake feasible edges and hence the same set of MAWs of length d. The statement follows from the fact that if two strings are d-equivalent, then they are also $(d-1)$-equivalent [Karhumäki et al. 2013]. □

Example 24. All strings in Figure 3 have the same MAWs of length up to 3, namely aaa and bbb. Let us remark that Lemma 23 is not a characterization of a d-equivalence class: aaa and bbb are also the MAWs of length up to 3 of string aabaababba, which is not d-equivalent to the strings of Figure 3.

Let us now describe an application of Lemma 23. We start with a straightforward fact.

Fact 25. *A string X occurs in a string Y if and only if no MAW of Y occurs in X. Equivalently, a string X does not occur in a string Y if and only if a MAW of Y occurs in X.*

Let S be a string of length n and d be an integer so that the number of distinct strings that are d-equivalent to S is at least z. (Note that S is d-equivalent to itself.) Then Lemma 23 and Fact 25 tell us that the set of MAWs of S of length up to d suffices to construct a z-RSDS over S for d that answers decision pattern matching queries of length $m \leq d$ in the optimal $O(m)$ time per query.

In particular, we can construct the **Aho-Corasick (AC)** automaton [Aho and Corasick 1975; Dori and Landau 2006] of the set of MAWs of S of length up to d in $O(M)$ time, where M is the total length of these MAWs. Then, given a string P of length m, we check if any MAW of S of length up to d occurs in P in $O(m)$ time. By Fact 25, we give a positive answer if no MAW of S of length up d occurs in P. The size of this z-RSDS is the size of the AC automaton, which is $O(M)$ and can be sublinear in $|S| = n$. Note that M can be computed in $O(n)$ time using the data structure presented in the work of Charalampopoulos et al. [2018a]. Thus, if $M < n$, we can construct the AC automaton instead of the z-RSDS presented in Section 3. The data structure of Charalampopoulos et al. [2018a] can also enumerate the set of MAWs of S of length up to d in the optimal $O(n + M)$ time.

Let us conclude this section with an example.

Example 26. Recall that all strings in Figure 3 have the same MAWs of length up to $d = 3$, namely aaa and bbb, and let $z = 6$. The AC automaton of {aaa, bbb} is a z-RSDS over S answering decision pattern matching queries of length $m \leq d = 3$. Given, for instance, query $P =$ aba, we check that the AC automaton does not locate any occurrence of {aaa, bbb} in P, and thus we return a positive answer. Given query $Q =$ aaa, we check that the AC automaton locates an occurrence of aaa from {aaa, bbb} at position 0 of Q, and thus we return a negative answer. Note that $M = 6 < n = 10$.

9 FINAL REMARKS

We have introduced the notion of z-RSDSs and presented such data structures for text indexing. Let us remark that the algorithmic contribution of this work is computing the maximal d and constructing a string $S' \sim_d S$. From thereon, one could:

- Employ *any privacy-preserving technique* over S' (e.g., Bernardini et al. [2020a, c]) to ensure that certain privacy-utility trade-offs are maintained. Such a technique could hide sensitive patterns that are still present in S' while maintaining the utility of S' in data analysis tasks.
- Construct *any compressed index* over S' (e.g., Grossi and Vitter [2005]; Kosolobov and Sivukhin [2019]). Such an index could answer queries of length $m \leq d$ and output FAIL for queries of length $m > d$ by storing d using $\log d$ extra bits.

There are at least five directions for future work:

(1) Improve the time complexity of the construction (Theorem 1).
(2) Improve the time complexity of the construction (Theorem 1) for certain values of z.
(3) Design a space-efficient construction algorithm.
(4) In Problem 1, the z-RSDS is truncated at a maximal uniform string depth d. Another related problem definition would be to truncate the z-RSDS at maximal non-uniform string depths.
(5) In Section 8, we proposed a small z-RSDS for decision pattern matching queries of length at most d, when d is given. An open problem is to efficiently compute the maximal such d.

ACKNOWLEDGMENTS

We would like to thank the associate editor and the anonymous reviewers for their constructive comments. We acknowledge the use of the Rosalind HPC cluster hosted by King's College London.

REFERENCES

Alfred V. Aho and Margaret J. Corasick. 1975. Efficient string matching: An aid to bibliographic search. *Commun. ACM* 18, 6 (1975), 333–340. https://doi.org/10.1145/360825.360855

Hiroki Arimura and Takeaki Uno. 2007. An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. *J. Comb. Optim.* 13, 3 (2007), 243–262. https://doi.org/10.1007/s10878-006-9029-1

Maurizio Atzori, Francesco Bonchi, Fosca Giannotti, and Dino Pedreschi. 2008. Anonymity preserving pattern discovery. *VLDB J.* 17, 4 (2008), 703–727. https://doi.org/10.1007/s00778-006-0034-x

Lorraine A. K. Ayad, Golnaz Badkobeh, Gabriele Fici, Alice Héliou, and Solon P. Pissis. 2019. Constructing antidictionaries in output-sensitive space. In *Proc. 9th DCC*. IEEE, Los Alamitos, CA, 538–547. https://doi.org/10.1109/DCC.2019.00062

Fahiem Bacchus, Adam J. Grove, Joseph Y. Halpern, and Daphne Koller. 1996. From statistical knowledge bases to degrees of belief. *Artif. Intell.* 87, 1–2 (1996), 75–143. https://doi.org/10.1016/S0004-3702(96)00003-3

Imon Banerjee, Kevin Li, Martin Seneviratne, Michelle Ferrari, Tina Seto, James D. Brooks, Daniel L. Rubin, and Tina Hernandez-Boussard. 2019. Weakly supervised natural language processing for assessing patient-centered outcome following prostate cancer treatment. *JAMIA Open* 2, 1 (2019), 150–159. https://doi.org/10.1093/jamiaopen/ooy057

Carl Barton, Alice Héliou, Laurent Mouchard, and Solon P. Pissis. 2014. Linear-time computation of minimal absent words using suffix array. *BMC Bioinform.* 15 (2014), 388. https://doi.org/10.1186/s12859-014-0388-9

Carl Barton, Alice Héliou, Laurent Mouchard, and Solon P. Pissis. 2015. Parallelising the computation of minimal absent words. In *Proc. 11th PPAM. Revised Selected Papers, Part II.* Springer, 243–253. https://doi.org/10.1007/978-3-319-32152-3_23

David R. Bentley. 2006. Whole-genome re-sequencing. *Curr. Opin. Genet. Dev.* 16, 6 (2006), 545–552. https://doi.org/10.1016/j.gde.2006.10.009

Giulia Bernardini, Huiping Chen, Alessio Conte, Roberto Grossi, Grigorios Loukides, Nadia Pisanti, Solon P. Pissis, Giovanna Rosone, and Michelle Sweering. 2020a. Combinatorial algorithms for string sanitization. *ACM Trans. Knowl. Discov. Data* 15, 1 (2020), Article 8, 34 pages. https://doi.org/10.1145/3418683

Giulia Bernardini, Huiping Chen, Alessio Conte, Roberto Grossi, Grigorios Loukides, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. 2019. String sanitization: A combinatorial approach. In *Proc. ECML/PKDD*. https://doi.org/10.1007/978-3-030-46150-8_37

Giulia Bernardini, Huiping Chen, Gabriele Fici, Grigorios Loukides, and Solon P. Pissis. 2020b. Reverse-safe data structures for text indexing. In *Proc. 22nd ALENEX*. 199–213. https://doi.org/10.1137/1.9781611976007.16

Giulia Bernardini, Huiping Chen, Grigorios Loukides, Nadia Pisanti, Solon P. Pissis, Leen Stougie, and Michelle Sweering. 2020c. String sanitization under edit distance. In *Proc. 31st CPM*. Article 7, 14 pages. https://doi.org/10.4230/LIPIcs.CPM.2020.7

Giulia Bernardini, Alessio Conte, Garance Gourdel, Roberto Grossi, Grigorios Loukides, Nadia Pisanti, Solon P. Pissis, Giulia Punzi, Leen Stougie, and Michelle Sweering. 2020d. Hide and mine in strings: Hardness and algorithms. In *Proc. 20th ICDM*. IEEE, Los Alamitos, CA, 924–929. https://doi.org/10.1109/ICDM50108.2020.00103

Elisa Bertino, Gabriel Ghinita, and Ashish Kamra. 2011. Access control for databases: Concepts and systems. *Found. Trends Databases* 3, 1–2 (2011), 1–148. https://doi.org/10.1561/1900000014

Bruhadeshwar Bezawada, Alex X. Liu, Bargav Jayaraman, Ann L. Wang, and Rui Li. 2015. Privacy preserving string matching for cloud computing. In *Proc. 35th ICDCS*. IEEE, Los Alamitos, CA, 609–618. https://doi.org/10.1109/ICDCS.2015.68

Christian Böhm and Florian Krebs. 2004. The k-nearest neighbour join: Turbo charging the KDD process. *Knowl. Inf. Syst.* 6, 6 (2004), 728–749. https://doi.org/10.1007/s10115-003-0122-9

Luca Bonomi, Liyue Fan, and Hongxia Jin. 2016. An information-theoretic approach to individual sequential data sanitization. In *Proc. 9th WSDM*. ACM, New York, NY, 337–346. https://doi.org/10.1145/2835776.2835828

James R. Bunch and John E. Hopcroft. 1974. Triangular factorization and inversion by fast matrix multiplication. *Math. Comp.* 28, 125 (1974), 231–236. http://www.jstor.org/stable/2005828.

Arturo Carpi and Aldo de Luca. 2001. Words and special factors. *Theor. Comput. Sci.* 259, 1–2 (2001), 145–182. https://doi.org/10.1016/S0304-3975(99)00334-5

Bastien Cazaux, Thierry Lecroq, and Eric Rivals. 2019. Linking indexing data structures to de Bruijn graphs: Construction and update. *J. Comput. Syst. Sci.* 104 (2019), 165–183. https://doi.org/10.1016/j.jcss.2016.06.008

Supaporn Chairungsee and Maxime Crochemore. 2012. Using minimal absent words to build phylogeny. *Theor. Comput. Sci.* 450 (2012), 109–116. https://doi.org/10.1016/j.tcs.2012.04.031

Panagiotis Charalampopoulos, Maxime Crochemore, Gabriele Fici, Robert Mercas, and Solon P. Pissis. 2018b. Alignment-free sequence comparison using absent words. *Inf. Comput.* 262, Part (2018), 57–68. https://doi.org/10.1016/j.ic.2018.06.002

Panagiotis Charalampopoulos, Maxime Crochemore, and Solon P. Pissis. 2018a. On extended special factors of a word. In *Proc. 25th SPIRE*. 131–138. https://doi.org/10.1007/978-3-030-00479-8_11

Panagiotis Charalampopoulos, Costas S. Iliopoulos, Chang Liu, and Solon P. Pissis. 2020. Property suffix array with applications in indexing weighted sequences. *ACM J. Exp. Algorithmics* 25, 1 (April 2020), Article 1.8, 16 pages. https://doi.org/10.1145/3385898

Rui Chen, Gergely Acs, and Claude Castelluccia. 2012a. Differentially private sequential data publication via variable-length n-grams. In *Proc. 4th CCS*. ACM, New York, NY, 638–649. https://doi.org/10.1145/2382196.2382263

Rui Chen, Benjamin C. M. Fung, Bipin C. Desai, and Nériah M. Sossou. 2012b. Differentially private transit data publication: A case study on the montreal transportation system. In *Proc. 18th KDD*. ACM, New York, NY, 213–221. https://doi.org/10.1145/2339530.2339564

Charles J. Colbourn, Wendy J. Myrvold, and Eugene Neufeld. 1996. Two algorithms for unranking arborescences. *J. Algorithms* 20, 2 (1996), 268–281. https://doi.org/10.1006/jagm.1996.0014

Maxime Crochemore, Alice Héliou, Gregory Kucherov, Laurent Mouchard, Solon P. Pissis, and Yann Ramusat. 2020. Absent words in a sliding window with applications. *Inf. Comput.* 270 (2020), 104461. https://doi.org/10.1016/j.ic.2019.104461

Maxime Crochemore, Lucian Ilie, Costas S. Iliopoulos, Marcin Kubica, Wojciech Rytter, and Tomasz Walen. 2013. Computing the longest previous factor. *Eur. J. Comb.* 34, 1 (2013), 15–26. https://doi.org/10.1016/j.ejc.2012.07.011

Maxime Crochemore, Filippo Mignosi, and Antonio Restivo. 1998. Automata and forbidden words. *Inf. Process. Lett.* 67 (1998), 111–117. https://doi.org/10.1016/S0020-0190(98)00104-5

Maxime Crochemore, Filippo Mignosi, Antonio Restivo, and Sergio Salemi. 2000. Data compression using antidictionaries. *Proc. IEEE* 88, 11 (2000), 1756–1768. https://doi.org/10.1109/5.892711

James Weldon Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. 1999. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl.* 20, 3 (1999), 720–755. https://doi.org/10.1137/S0895479895291765

U.S. Department of Health & Human Services. 1996. Health Insurance Portablility and Accountability Act of 1996. Retrieved May 19, 2021 from https://aspe.hhs.gov/report/health-insurance-portability-and-accountability-act-1996.

Shiri Dori and Gad M. Landau. 2006. Construction of Aho Corasick automaton in linear time for integer alphabets. *Inf. Process. Lett.* 98, 2 (2006), 66–72. https://doi.org/10.1016/j.ipl.2005.11.019

Eigen Library. 2020. Eigen Library. Retrieved May 19, 2021 from http://eigen.tuxfamily.org.

European Parliament. 2015. General Data Protection Regulation. Retrieved May 19, 2021 from http://data.consilium.europa.eu/doc/document/ST-9565-2015-INIT/en/pdf.

Martin Farach-Colton. 1997. Optimal suffix tree construction with large alphabets. In *Proc. 38th FOCS*. IEEE, Los Alamitos, CA, 137–143. https://doi.org/10.1109/SFCS.1997.646102

Gabriele Fici and Pawel Gawrychowski. 2019. Minimal absent words in rooted and unrooted trees. In *26th SPIRE*. Lecture Notes in Computer Science, Vol. 11811. Springer, 152–161. https://doi.org/10.1007/978-3-030-32686-9_11

Gabriele Fici, Filippo Mignosi, Antonio Restivo, and Marinella Sciortino. 2006. Word assembly through minimal forbidden words. *Theor. Comput. Sci.* 359, 1–3 (2006), 214–230. https://doi.org/10.1016/j.tcs.2006.03.006

Gabriele Fici, Antonio Restivo, and Laura Rizzo. 2019. Minimal forbidden factors of circular words. *Theor. Comput. Sci.* 792 (2019), 144–153. https://doi.org/10.1016/j.tcs.2018.05.037

Johannes Fischer and Volker Heun. 2011. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM J. Comput.* 40, 2 (2011), 465–492. https://doi.org/10.1137/090779759

Michael L. Fredman, János Komlós, and Endre Szemerédi. 1984. Storing a sparse table with $O(1)$ worst case access time. *J. ACM* 31, 3 (1984), 538–544. https://doi.org/10.1145/828.1884

Yuta Fujishige, Yuki Tsujimaru, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. 2016. Computing DAWGs and minimal absent words in linear time for integer alphabets. In *41st MFCS. Leibniz International Proceedings in Informatics*, Vol. 58. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, Article 38, 14 pages. https://doi.org/10.4230/LIPIcs.MFCS.2016.38

Sara P. Garcia, O. J. Pinho, João M. O. S. Rodrigues, Carlos A. C. Bastos, and Paulo J. S. G. Ferreira. 2011. Minimal absent words in prokaryotic and eukaryotic genomes. *PLoS One* 6, 1 (2011), e16065. https://doi.org/10.1371/journal.pone.0016065

John R. Gilbert and Tim Peierls. 1988. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sci. Comput.* 9, 5 (1988), 862–874. https://doi.org/10.1137/0909058

Aris Gkoulalas-Divanis and Grigorios Loukides. 2011. Revisiting sequential pattern hiding to enhance utility. In *Proc. 17th KDD*. ACM, New York, NY, 1316–1324. https://doi.org/10.1145/2020408.2020605

Izrail S. Gradshteyn and Iosif M. Ryzhik. 2007. *Table of Integrals, Series, and Products* (7 ed.). Elsevier/Academic Press, Amsterdam, Netherlands.

Roberto Grossi, John Iacono, Gonzalo Navarro, Rajeev Raman, and S. Rao Satti. 2017. Asymptotically optimal encodings of range data structures for selection and top-k queries. *ACM Trans. Algorithms* 13, 2 (2017), Article 28, 31 pages. https://doi.org/10.1145/3012939

Roberto Grossi and Jeffrey Scott Vitter. 2005. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.* 35, 2 (2005), 378–407. https://doi.org/10.1137/S0097539702402354

Dan Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences—Computer Science and Computational Biology*. Cambridge University Press.

Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proc. SIGMOD*. ACM, New York, NY, 47–57. https://doi.org/10.1145/602259.602266

Robert Gwadera, Aris Gkoulalas-Divanis, and Grigorios Loukides. 2013. Permutation-based sequential pattern hiding. In *Proc. 13th ICDM*. IEEE, Los Alamitos, CA, 241–250. https://doi.org/10.1109/ICDM.2013.57

Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. In *Proc. SIGMOD*, Vol. 29. 1–12. https://doi.org/10.1145/342009.335372

Raymond D. Heatherly, Grigorios Loukides, Joshua C. Denny, Jonathan L. Haines, Dan M. Roden, and Bradley A. Malin. 2013. Enabling genomic-phenomic association discovery without sacrificing anonymity. *PLoS One* 8 (Feb. 2013), 1–13. https://doi.org/10.1371/journal.pone.0053875

Michael Hoffmann, John Iacono, Patrick K. Nicholson, and Rajeev Raman. 2018. Encoding nearest larger values. *Theor. Comput. Sci.* 710 (2018), 97–115. https://doi.org/10.1016/j.tcs.2017.02.017

Joan P. Hutchinson. 1975. On words with prescribed overlapping subsequences. *Utilitas Mathematica* 7 (1975), 241–250.

Juhani Karhumäki, Svetlana Puzynina, Michaël Rao, and Markus A. Whiteland. 2017. On cardinalities of k-abelian equivalence classes. *Theor. Comput. Sci.* 658 (2017), 190–204. https://doi.org/10.1016/j.tcs.2016.06.010

Juhani Karhumäki, Aleksi Saarela, and Luca Q. Zamboni. 2013. On a generalization of abelian equivalence and complexity of infinite words. *J. Comb. Theory, Ser. A* 120, 8 (2013), 2189–2206. https://doi.org/10.1016/j.jcta.2013.08.008

Juha Kärkkäinen, Marcin Piatkowski, and Simon J. Puglisi. 2017. String inference from longest-common-prefix array. In *Proc. 44th ICALP*. Article 62, 14 pages. https://doi.org/10.4230/LIPIcs.ICALP.2017.62

Carl Kingsford, Michael C. Schatz, and Mihai Pop. 2010. Assembly complexity of prokaryotic genomes using short reads. *BMC Bioinform.* 11, 1 (2010), 21. https://doi.org/10.1186/1471-2105-11-21

Donald Ervin Knuth. 1998. *The Art of Computer Programming, Volume II: Seminumerical Algorithms* (3rd ed.). Addison-Wesley.

Dmitry Kosolobov and Nikita Sivukhin. 2019. Compressed multiple pattern matching. In *30th CPM. Leibniz International Proceedings in Informatics*, Vol. 128. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, Article 13, 14 pages. https://doi.org/10.4230/LIPIcs.CPM.2019.13

François Le Gall. 2014. Powers of tensors and fast matrix multiplication. In *Proc. 25th ISSAC*. ACM, New York, NY, 296–303. https://doi.org/10.1145/2608628.2608664

Chris-Andre Leimeister and Burkhard Morgenstern. 2014. Kmacs: The k-mismatch average common substring approach to alignment-free sequence comparison. *Bioinformatics* 30, 14 (May 2014), 2000–2008. https://doi.org/10.1093/bioinformatics/btu331

Jin Li, Qian Wang, Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. 2010. Fuzzy keyword search over encrypted data in cloud computing. In *Proc. 29th INFOCOM*. IEEE, Los Alamitos, CA, 1–5. https://doi.org/10.1109/INFCOM.2010.5462196

Tiancheng Li and Ninghui Li. 2008. Injector: Mining background knowledge for data anonymization. In *Proc. 24th ICDE*. IEEE, Los Alamitos, CA, 446–455. https://doi.org/10.1109/ICDE.2008.4497453

Po-Ching Lin, Ying-Dar Lin, Yuan-Cheng Lai, and Tsern-Huei Lee. 2008. Using string matching for deep packet inspection. *IEEE Comput.* 41, 4 (2008), 23–28. https://doi.org/10.1109/MC.2008.138

Grigorios Loukides, Aris Gkoulalas-Divanis, and Bradley Malin. 2010. Anonymization of electronic medical records for validating genome-wide association studies. *Proc. Natl. Acad. Sci. USA* 107, 17 (2010), 7898–7903. https://doi.org/10.1073/pnas.0911686107

Grigorios Loukides and Robert Gwadera. 2015. Optimal event sequence sanitization. In *Proc. 15th SDM*. 775–783. https://doi.org/10.1137/1.9781611974010.87

Bradley Malin and Latanya Sweeney. 2000. Determining the identifiability of DNA database entries. In *Proc. AMIA*. 537–541. http://knowledge.amia.org/amia-55142-a2000a-1.606968/t-001-1.609408/f-001-1.609409/a-108-1.609653/a-109-1.609650.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

Takuya Mieno, Yuki Kuhara, Tooru Akagi, Yuta Fujishige, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. 2020. Minimal unique substrings and minimal absent words in a sliding window. In *46th SOFSEM*. Lecture Notes in Computer Science, Vol. 12011. Springer, 148–160. https://doi.org/10.1007/978-3-030-38919-2_13

Filippo Mignosi, Antonio Restivo, and Marinella Sciortino. 2002. Words and forbidden factors. *Theor. Comput. Sci.* 273, 1–2 (2002), 99–117. https://doi.org/10.1016/S0304-3975(00)00436-9

Noman Mohammed, Benjamin C. M. Fung, Patrick C. K. Hung, and Cheuk-Kwong Lee. 2010. Centralized and distributed anonymization for high-dimensional healthcare data. *ACM Trans. Knowl. Discov. Data* 4 (2010), Article 18, 33 pages. https://doi.org/10.1145/1857947.1857950

Joong Chae Na, Alberto Apostolico, Costas S. Iliopoulos, and Kunsoo Park. 2003. Truncated suffix trees and their application to data compression. *Theor. Comput. Sci.* 304, 1 (2003), 87–101. https://doi.org/10.1016/S0304-3975(03)00053-7

Shubha U. Nabar, Krishnaram Kenthapadi, Nina Mishra, and Rajeev Motwani. 2008. A survey of query auditing techniques for data privacy. In *Privacy-Preserving Data Mining: Models and Algorithms*, Charu C. Aggarwal and Philip S. Yu (Eds.). Springer, 415–431. https://doi.org/10.1007/978-0-387-70992-5_17

Takahiro Ota and Hiroyoshi Morita. 2010. On the adaptive antidictionary code using minimal forbidden words with constant lengths. In *Proc ISITA 2010*. IEEE, Los Alamitos, CA, 72–77. https://doi.org/10.1109/ISITA.2010.5649621

Giorgos Poulis, Spiros Skiadopoulos, Grigorios Loukides, and Aris Gkoulalas-Divanis. 2014. Apriori-based algorithms for k^{m}-anonymizing trajectory data. *Trans. Data Priv.* 7, 2 (2014), 165–194. http://www.tdp.cat/issues11/abs.a194a14.php.

Diogo Pratas and Jorge M. Silva. 2020. Persistent minimal sequences of SARS-CoV-2. *Bioinformatics* 36 (2020), 5129–5132. https://doi.org/10.1093/bioinformatics/btaa686

Hong Qin, Hao Wang, Xiaochao Wei, Likun Xue, and Lei Wu. 2019. Privacy-preserving wildcards pattern matching protocol for IoT applications. *IEEE Access* 7 (2019), 36094–36102. https://doi.org/10.1109/ACCESS.2019.2900519

Rajeev Raman. 2015. Encoding data structures. In *Proc. 9th WALCOM*. 1–7. https://doi.org/10.1007/978-3-319-15612-5_1

David F. Robinson and Les R. Foulds. 1981. Comparison of phylogenetic trees. *Math. Biosci.* 53, 1 (1981), 131–147. https://doi.org/10.1016/0025-5564(81)90043-2

Naruya Saitou and Masatoshi Nei. 1987. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.* 4, 4 (July 1987), 406–425. https://doi.org/10.1093/oxfordjournals.molbev.a040454

Pierangela Samarati and Latanya Sweeney. 1998. Generalizing data to provide anonymity when disclosing information (Abstract). In *17th ACM SIGACT-SIGMOD-SIGART*, Alberto O. Mendelzon and Jan Paredaens (Eds.). ACM, New York, NY, 188. https://doi.org/10.1145/275487.275508

Jingbo Shang, Jian Peng, and Jiawei Han. 2016. MACFP: Maximal approximate consecutive frequent pattern mining under edit distance. In *Proc. 16th SDM*. 558–566. https://doi.org/10.1137/1.9781611974348.63

Sumit Sidana, Charlotte Laclau, Massih R. Amini, Gilles Vandelle, and André Bois-Crettez. 2017. KASANDR: A large-scale dataset with implicit feedback for recommendation. In *Proc. 40th SIGIR*. ACM, New York, NY, 1245–1248. https://doi.org/10.1145/3077136.3080713

Raquel M. Silva, Diogo Pratas, Luísa Castro, Armando J. Pinho, and Paulo J. S. G. Ferreira. 2015. Three minimal sequences found in Ebola virus genomes and absent from human DNA. *Bioinformatics* 31, 15 (2015), 2421–2425. https://doi.org/10.1093/bioinformatics/btv189

Henry J. Smith, Tamara Dinev, and Heng Xu. 2011. Information privacy research: An interdisciplinary review. *Manag. Inf. Syst. Q.* 35, 4 (2011), 989–1015. http://misq.org/catalog/product/view/id/1518/s/information-privacy-research-an-interdisciplinary-review/.

Acar Tamersoy, Grigorios Loukides, Mehmet Ercan Nergiz, Yücel Saygin, and Bradley Malin. 2012. Anonymization of longitudinal electronic medical records. *IEEE Trans. Inf. Technol. Biomed.* 16, 3 (2012), 413–423. https://doi.org/10.1109/TITB.2012.2185850

Manolis Terrovitis and Nikos Mamoulis. 2008. Privacy preservation in the publication of trajectories. In *Proc. 9th MDM*. IEEE, Los Alamitos, CA, 65–72. https://doi.org/10.1109/MDM.2008.29

Manolis Terrovitis, Nikos Mamoulis, and Panos Kalnis. 2011. Local and global recoding methods for anonymizing set-valued data. *VLDB J.* 20, 1 (2011), 83–106. https://doi.org/10.1007/s00778-010-0192-8

Manolis Terrovitis, Giorgos Poulis, Nikos Mamoulis, and Spiros Skiadopoulos. 2017. Local suppression and splitting techniques for privacy preserving publication of trajectories. *IEEE Trans. Knowl. Data Eng.* 29, 7 (2017), 1466–1479. https://doi.org/10.1109/TKDE.2017.2675420

Sharma V. Thankachan, Sriram P. Chockalingam, Yongchao Liu, Ambujam Krishnan, and Srinivas Aluru. 2017. A greedy alignment-free distance estimator for phylogenetic inference. *BMC Bioinform.* 18, 8 (2017), Article 238, 8 pages. https://doi.org/10.1186/s12859-017-1658-0

George Theodorakopoulos, Reza Shokri, Carmela Troncoso, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. 2014. Prolonging the hide-and-seek game: Optimal trajectory privacy for location-based services. In *Proc. 13th WPES*. ACM, New York, NY, 73–82. https://doi.org/10.1145/2665943.2665946

Igor Ulitsky, David Burstein, Tamir Tuller, and Benny Chor. 2006. The average common substring approach to phylogenomic reconstruction. *J. Comput. Biol.* 13, 2 (2006), 336–350. https://doi.org/10.1089/cmb.2006.13.336

Di Wang, Yeye He, Elke Rundensteiner, and Jeffrey F. Naughton. 2013. Utility-maximizing event stream suppression. In *Proc. SIGMOD*. ACM, New York, NY, 589–600. https://doi.org/10.1145/2463676.2465305

Jing Wang, Nikos Ntarmos, and Peter Triantafillou. 2016. Indexing query graphs to speedup graph query processing. In *Proc. 19th EDBT*. 41–52. https://doi.org/10.5441/002/edbt.2016.07

Peter Weiner. 1973. Linear pattern matching algorithms. In *Proc. 14th SWAT*. IEEE, Los Alamitos, CA, 1–11. https://doi.org/10.1109/SWAT.1973.13

Virginia Vassilevska Williams. 2012. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th STOC*. ACM, New York, NY, 887–898. https://doi.org/10.1145/2213977.2214056

Xiaokui Xiao, Yufei Tao, and Nick Koudas. 2010. Transparent anonymization: Thwarting adversaries who know the algorithm. *ACM Trans. Database Syst.* 35, 2 (2010), Article 8, 48 pages. https://doi.org/10.1145/1735886.1735887

Shengzhi Xu, Xiang Cheng, Sen Su, Ke Xiao, and Li Xiong. 2016. Differentially private frequent sequence mining. *IEEE Trans. Knowl. Data Eng.* 28, 11 (2016), 2910–2926. https://doi.org/10.1109/TKDE.2016.2601106

Mohammed J. Zaki, Wagner Meira Jr, and Wagner Meira. 2014. *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press. http://www.dataminingbook.info/.

Received October 2020; revised April 2021; accepted April 2021