Theory of Computing Systems (2026) 70:21
https://doi.org/10.1007/s00224-025-10240-z

On Strings Having the Same Length-k Substrings

Giulia Bernardini[1] · Alessio Conte[2] · Esteban Gabory[3] · Roberto Grossi[2] · Grigorios Loukides[4] · Solon P. Pissis[5] · Giulia Punzi[2] · Michelle Sweering[5]

Received: 31 January 2025 / Accepted: 18 August 2025

Abstract

Let $\mathsf{Substr}_k(X)$ denote the set of length-k substrings of a given string X for a given integer $k > 0$. We study the following basic string problem, called z-SHORTEST $\mathcal{S}_k$-EQUIVALENT STRINGS: Given a set $\mathcal{S}_k$ of n length-k strings and an integer $z > 0$, list z shortest distinct strings $T_1, \ldots, T_z$ such that $\mathsf{Substr}_k(T_i) = \mathcal{S}_k$, for all $i \in [1, z]$. The z-SHORTEST $\mathcal{S}_k$-EQUIVALENT STRINGS problem arises naturally as an encoding problem in many real-world applications; e.g. in data privacy, data compression, and bioinformatics. The 1-SHORTEST $\mathcal{S}_k$- EQUIVALENT STRINGS, referred to as SHORTEST $\mathcal{S}_k$-EQUIVALENT STRING, asks for a shortest string X such that $\mathsf{Substr}_k(X) = \mathcal{S}_k$. Our main contributions are as follows. Given a directed graph $G = (V, E)$, the DIRECTED CHINESE POSTMAN (DCP) problem asks for a shortest closed walk that visits every edge of G at least once. DCP can be solved using an algorithm for min-cost flow. We show, via a non-trivial reduction, that if SHORTEST $\mathcal{S}_k$-EQUIVALENT STRING over a *binary alphabet* has a near-linear-time solution then so does DCP. Secondly, we show that the length of a shortest string output by SHORTEST $\mathcal{S}_k$-EQUIVALENT STRING is in $\mathcal{O}(k + n^2)$. We generalize this bound by showing that the total length of z shortest strings is in $\mathcal{O}(zk + zn^2 + z^2n)$. We derive these upper bounds by showing (asymptotically tight) bounds on the total length of z shortest Eulerian walks in general directed graphs. Furthermore, we present an algorithm for solving z-SHORTEST $\mathcal{S}_k$-EQUIVALENT STRINGS in $\mathcal{O}(nk + n^2 \log^2 n + zn^2 \log n + |\mathsf{output}|)$ time. If $z = 1$, the time becomes $\mathcal{O}(nk + n^2 \log^2 n)$ by the fact that the size of the input is $\Theta(nk)$ and the size of the output is $\mathcal{O}(k + n^2)$. Finally, we also provide a direct technical application of our algorithms on strings in an existing data privacy framework. A preliminary version of this paper was announced at CPM 2022.

Keywords Directed Chinese postman · Eulerian walk · de Bruijn graph · Listing

Giulia Bernardini, Alessio Conte, Esteban Gabory, Roberto Grossi, Grigorios Loukides, Solon P. Pissis, Giulia Punzi, Michelle Sweering contributed equally to this work.

Extended author information available on the last page of the article

Published online: 01 April 2026

1 Introduction

We start with some basic definitions and notation on strings from [1]. Let $X = X[0]\cdots X[n-1]$ be a *string* of length $|X| = n$ over an alphabet Σ whose elements are called *letters*. For any two positions i and $j \geq i$ of X, $X[i\,..\,j]$ is the *fragment* of X starting at position i and ending at position j. The fragment $X[i\,..\,j]$ is an *occurrence* of the underlying *substring* $P = X[i]\cdots X[j]$; we say that P occurs at *position* i in X. A *prefix* of X is a fragment of the form $X[0\,..\,j]$ and a *suffix* of X is a fragment of the form $X[i\,..\,n-1]$. By XY or $X\cdot Y$ we denote the *concatenation* of two strings X and Y, i.e. $XY = X[0]\cdots X[|X|-1]Y[0]\cdots Y[|Y|-1]$. Given two sets of strings $\mathcal{X}$ and $\mathcal{Y}$, by $\mathcal{X}\cdot\mathcal{Y}$ we denote their *Cartesian concatenation* $\{XY,\ X\in\mathcal{X},\ Y\in\mathcal{Y}\}$. For clarity, we might omit the brackets for sets containing a single element. e.g. given a string X and a set of strings $\mathcal{Y}$, we write $X\cdot\mathcal{Y} = \{X\}\cdot\mathcal{Y} = \{XY,\ Y\in\mathcal{Y}\}$.

Let $\mathsf{Substr}_k(X)$ denote the set of length-k substrings of a finite string X (also called the k-spectrum of X [2]). We consider the following basic problem on strings.

z- SHORTEST $\mathcal{S}_k$- EQUIVALENT STRINGS
Input: A set $\mathcal{S}_k$ of n length-k strings over an integer alphabet $\Sigma = [0, nk)$; and an integer $z > 0$.
Output: A list $\mathcal{T}_z = T_1, \ldots, T_z$ of z distinct strings over Σ, such that for all $i \in [1, z]$, $\mathsf{Substr}_k(T_i) = \mathcal{S}_k$ and for every string T' not in $\mathcal{T}_z$ with $\mathsf{Substr}_k(T') = \mathcal{S}_k$, $|T'| \geq |T_i|$, for all $i \in [1, z]$; or FAIL if that is not possible.

In particular, if $z = 1$ the problem consists in finding a shortest string X such that $\mathsf{Substr}_k(X) = \mathcal{S}_k$. In this case, we call the problem SHORTEST $\mathcal{S}_k$-EQUIVALENT STRING. We solve z-SHORTEST $\mathcal{S}_k$-EQUIVALENT STRINGS by reducing it to listing Eulerian walks on a directed graph: the de Bruijn graph of order k of $\mathcal{S}_k$. Let us first recall a few basic definitions before formally defining the problem in scope. Given $\mathcal{S}_k$, the *de Bruijn graph* (dBG) of order k of $\mathcal{S}_k$ is a directed graph $G_{\mathcal{S}_k} = (V, E)$, where V is the set of length-$(k-1)$ substrings of the strings in $\mathcal{S}_k$, and $G_{\mathcal{S}_k}$ contains an edge (u, v) if and only if the string $S = u[0]\cdot v$ is equal to the string $u\cdot v[k-2]$ and $S \in \mathcal{S}_k$. A *walk* in a directed graph $G = (V, E)$ is a sequence of edges from E that joins a sequence of nodes from V. An *Eulerian walk* in G is a walk that visits all edges in E at least once. Any string X such that $\mathsf{Substr}_k(X) = \mathcal{S}_k$ corresponds to an Eulerian walk W in the dBG of order k of $\mathcal{S}_k$ and vice versa. We formally define the problem of listing Eulerian walks in a directed graph.

z- SHORTEST EULERIAN WALKS
Input: A directed graph $G = (V, E)$ and an integer $z > 0$.
Output: A list $\mathcal{W}_z = W_1, \ldots, W_z$ of z distinct Eulerian walks of G, such that for every Eulerian walk W' of G not in $\mathcal{W}_z$, $|W'| \geq |W_i|$, for all $i \in [1, z]$; or FAIL if that is not possible.

If $z = 1$, we call the problem SHORTEST EULERIAN WALK. Let us denote the total size of $\mathcal{W}_z$ (that is, the total length of the walks) by $||\mathcal{W}_z||$. We show the following result.[1]

[1] We assume that basic arithmetic operations take constant time, which is the case when $z = \mathrm{poly}(|E|)$.

Theorem 1 *The z-Shortest Eulerian Walks problem can be solved in:*

- $\mathcal{O}(|E||V|\log^2|V| + z|V|^3 + ||\mathcal{W}_z||)$ *time;*
- *or* $\mathcal{O}(|E||V|\log^2|V| + z(|E||V| + |V|^2\log|V|) + ||\mathcal{W}_z||)$ *time.*

We also investigate the combinatorial bounds on the total length $||\mathcal{W}_z||$ of the z shortest Eulerian walks in directed graphs. We show the following result.

Theorem 2 $||\mathcal{W}_1|| \leq |V||E|$ *and this bound is asymptotically tight. Moreover,* $||\mathcal{W}_z|| \leq z|V||E| + z^2|V|$ *and this bound is asymptotically tight.*

By employing Theorem 2, we show similar bounds for the total length $||\mathcal{T}_z||$ of the output of z-Shortest $\mathcal{S}_k$-Equivalent Strings:

Theorem 3 $||\mathcal{T}_1|| = \mathcal{O}(k + n^2)$ *and this bound is asymptotically tight. Moreover,* $||\mathcal{T}_z|| = \mathcal{O}(zk + zn^2 + z^2n)$ *and this bound is asymptotically tight.*

By employing Theorems 1 and 3, obtain the following result (recalling that the input size of z-Shortest $\mathcal{S}_k$-Equivalent Strings is $\Theta(nk)$):

Theorem 4 *The z-Shortest $\mathcal{S}_k$-Equivalent Strings problem can be solved in* $\mathcal{O}(nk + n^2\log^2 n + zn^2\log n + ||\mathcal{T}_z||)$ *time. If* $z = 1$ *this becomes* $\mathcal{O}(nk + n^2\log^2 n)$.

Furthermore, we give the following reduction of independent interest. Given a directed graph $G = (V, E)$, the Directed Chinese Postman (DCP) problem (also known as the Route Inspection problem) asks for a shortest circuit that visits every edge of G at least once. DCP can be solved using an algorithm for computing a min-cost flow [3]. To this end, we can use the classic *network simplex algorithm* that runs in $\tilde{\mathcal{O}}(|E||V|)$ time [4, 5] or the most recent $|E|^{1+o(1)}$-time algorithm by Chen et al. [6]. Interestingly, we show here, via a non-trivial reduction, that there is a connection between the two problems: namely, if the Shortest $\mathcal{S}_k$- Equivalent String problem over a *binary alphabet* has a near-linear-time solution, then so does DCP.

Motivation and Related Work The main theoretical motivation for this work comes from the following "gap" in the literature. Let $\mathcal{M}_k$ be a *multiset* (rather than a set) of length-k strings. Counting (resp. listing) all distinct strings whose multiset of length-k strings is $\mathcal{M}_k$ corresponds to counting (resp. listing) all node-distinct Eulerian trails (i.e. walks that do not repeat any edges) in the de Bruijn multigraph of order k of $\mathcal{M}_k$ [7–10]. Counting all node-distinct Eulerian trails can be done in polynomial time by employing the well-known BEST theorem [11] (see also [12] for the analogous result on strings). Efficient algorithms for listing z node-distinct Eulerian trails are also known [13]. However, the analogous, perhaps more basic, results for counting or listing all strings whose *set* of length-k strings is $\mathcal{S}_k$ are, to the best of our knowledge, unknown. Here we focus on listing and observe that each string whose set of length-k strings is $\mathcal{S}_k$ corresponds to some Eulerian walk in the dBG of order k of $\mathcal{S}_k$. Counting remains wide open, as an analogous to the BEST theorem for Eulerian walks is unknown. In fact, a fundamental difference is that $\mathcal{M}_k$, by definition, gives the exact

length of all strings whose multiset of length-k substrings is $\mathcal{M}_k$, while $\mathcal{S}_k$ gives only a lower bound on the length of each string whose set of length-k substrings is $\mathcal{S}_k$.

The practical motivation for this work comes from the fact that the z-SHORTEST $\mathcal{S}_k$-EQUIVALENT STRINGS problem arises naturally as an encoding problem in many real-world applications. In data privacy, the output strings can be used to construct reverse-safe data structures for pattern matching when $\mathcal{S}_k$ comes from a private string [10, 14]. In data compression, the output strings can be used to compactly represent a set $\mathcal{S}_k$ of length-k strings [15]. In bioinformatics, the output strings correspond to different possible genome reconstructions [16] when $\mathcal{S}_k$ is a set of sequences generated by a sequencing experiment.

Our study falls into a wider line of research that is concerned with algorithmic problems on strings that can be formulated as problems on dBGs [14, 17–27]. Particularly related to our work are the studies by Schmidt and Alanko [20] and Schmidt et al. [22], which address the problem of computing a compact representation of the length-k strings (or k-mers) present in a set I of strings through a new set I' of strings with minimal total length. Notably, their definition of a k-mer differs from ours, as it incorporates the biological notion of the *reverse complement* and leads to a de Bruijn *bigraph*, a variant of standard dBGs. Nevertheless, their solutions share similarities with ours in spirit, as they involve completing a de Bruijn bigraph to make it Eulerian.

The SHORTEST $\mathcal{S}_k$- EQUIVALENT STRING problem is also related to Simon's congruence [28]: two strings are $\sim_k$-*congruent* if they have the same set of *subsequences* of length at most k. For details on the combinatorial properties of the congruence see [28–34] and for some algorithmic works see [33, 35–43]. A long-standing open problem was to design an algorithm which, given two strings S and T, computes the largest k for which $S \sim_k T$. Gawrychowski et al. [40] have recently settled the problem optimally by showing a linear-time algorithm.

A preliminary version of this work appeared in [18]. This extended version contains the following new contributions. First, in Section 4.1.1 we refine Theorem 3 by showing that the stated bounds are tight. Second, in Section 4.4, we detail a special case in which we can lower the combinatorial upper bound on the total length of z shortest Eulerian walks. Lastly, in Section 6 we provide a direct technical application of our algorithms in data privacy, following the reverse-safe data structure framework [10].

Paper Organization In Section 2 we provide some basic definitions and notation. In Section 3 we present the reduction from DCP to the SHORTEST $\mathcal{S}_k$- EQUIVALENT STRING problem. In Section 4 we show the combinatorial bounds on the length of a shortest Eulerian walk, and on the total length of z shortest Eulerian walks. From these bounds, we infer the bounds on the length of the strings output for z-SHORTEST $\mathcal{S}_k$-EQUIVALENT STRINGS. In Section 5 we present our algorithm for solving the z-SHORTEST EULERIAN WALKS problem. From this algorithm, we infer our solution to the z-SHORTEST $\mathcal{S}_k$-EQUIVALENT STRINGS problem. In Section 6, we discuss a direct technical application of our algorithms in the reverse-safe data structure framework. We conclude the paper with Section 7, where we list some future directions.

2 Preliminaries

We consider directed graphs $G = (V, E)$ such that there is at most one directed edge (u, v) for any $u, v \in V$ (that is, all edges in E have multiplicity 1). For a graph $G = (V, E)$, we call *multiplicity function* on G a mapping $E \to \mathbb{N}$. We denote by $G_\mu(V, E)$ the *multigraph* with *underlying graph* G and multiplicity function μ: G_μ is a version of G having $\mu(e)$ copies of each edge $e \in E$. We call G_μ an *extension* of G. We call v the *head* of an edge (u, v), and we call u the edge *tail*. Given a node u, if there is no edge (v, u) for any $v \in V$ then u is called a *source*, if there is no edge (u, v) for any $v \in V$ then u is a *sink*.

A *walk* in G is any sequence $W = e_1 e_2 \dots e_{|W|}$ of edges in E such that the head of e_i is equal to the tail of e_{i+1}, for all $i \in [1, |W| - 1]$; $|W|$ is the *length* of W. A walk may traverse any edge multiple times. We denote by $I(W)$ the tail of e_1, by $L(W)$ the head of $e_{|W|}$ and by $\mathsf{mult}_e(W)$ the number of times W visits e, for any $e \in E$.

A graph $G = (V, E)$ is *strongly connected* if, for any two nodes $u, v \in V$ with $u \neq v$, there exist both a walk from u to v and a walk from v to u. The *strongly connected components* (SCCs in short) of G are its inclusion-maximal strongly connected subgraphs. A graph is *weakly connected* if replacing all of its directed edges with undirected edges yields a connected graph.

A walk W is *closed* (and in this case we call it a *circuit*) if $I(W) = L(W)$, that is, if it starts and ends at the same node. A walk W is *Eulerian* if it traverses all the edges of G at least once, that is, if the set $\{e_i\}_{i \in [1,|W|]} = E$. A walk that does not traverse any edge twice is a *trail*. A circuit that is also a trail is called a *cycle*.

A walk W on a given multigraph G_μ is Eulerian if $\mathsf{mult}_e(W) \geq \mu(e)$ for every $e \in E$; and it is an Eulerian *trail* if $\mathsf{mult}_e(W) = \mu(e)$ for every $e \in E$. A (multi)graph $G = (V, E)$ is *semi-Eulerian* if it admits an Eulerian trail and *Eulerian* if it admits an Eulerian cycle. We denote by $EW(G)$ and $ET(G)$ the set of Eulerian walks and the set of Eulerian trails on G, respectively. In the case of directed graphs with multiplicities, Eulerian or semi-Eulerian multigraphs can be characterized by the following reformulation of the well-known Euler's theorem.

Proposition 1 ([44]) Let G be a directed weakly connected graph, and μ be a multiplicity function on G. The multigraph G_μ is Eulerian if and only if the balancing conditions hold:

$$\sum_{(u,w)\in E} \mu(u, w) - \sum_{(w,u)\in E} \mu(w, u) = 0 \qquad for\ any\ fixed\ u \in V;$$

and is semi-Eulerian if the equality above holds, or if there exists (a,b) ∈ V such that:

$$\sum_{(u,w)\in E} \mu(u, w) - \sum_{(w,u)\in E} \mu(w, u) = 0 \qquad for\ any\ fixed\ u \in V \setminus \{a, b\}$$

$$\sum_{(a,w)\in E} \mu(a, w) - \sum_{(w,a)\in E} \mu(w, a) = 1 \qquad \sum_{(b,w)\in E} \mu(b, w) - \sum_{(w,b)\in E} \mu(w, b) = -1.$$

3 Reducing Directed Chinese Postman to Shortest Equivalent String

DIRECTED CHINESE POSTMAN (DCP)
Input: A directed graph $G = (V, E)$.
Output: A shortest Eulerian circuit, or FAIL if that is not possible.

The main goal of this section is to reduce DCP to SHORTEST $\mathcal{S}_k$-EQUIVALENT STRING. In Section 3.1 we show a simple linear-time reduction that uses an alphabet of size $\mathcal{O}(|V|)$. In Section 3.2 we then show a more involved near-linear-time reduction that uses a binary alphabet. The latter reduction has the following important implication: if SHORTEST $\mathcal{S}_k$-EQUIVALENT STRING over a binary alphabet has a near-linear-time solution, then so does DCP.

3.1 Large Alphabet

Lemma 1 *Given a directed graph $G = (V, E)$, we define $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ such that $\widetilde{V} = V \cup \{a, b\}$, where a, b are two bogus nodes, and $\widetilde{E} = E \cup \{(a, u), (u, b)\}$ for an arbitrary fixed node $u \in V$. Then from any shortest Eulerian walk on $\widetilde{G}$, we can compute a shortest Eulerian circuit on G in constant time.*

Proof Let $\widetilde{W}$ be a shortest Eulerian walk on $\widetilde{G}$. By definition, the bogus node a is a source, and since $\widetilde{W}$ must traverse edge (a, u), it always starts from a; symmetrically, it must end at b. The rest of the walk is on G, and since a and b are connected only to u, the latter is both the second and second-to-last node crossed by $\widetilde{W}$. The edges traversed by $\widetilde{W}$ between these two visits of u form an Eulerian circuit W on G. Assume for a contradiction that an Eulerian circuit on G shorter than W exists. Then we could make it start and end at u (possibly applying a rotation), add edges (a, u) and (u, b) at the beginning and at the end, and obtain an Eulerian walk on $\widetilde{G}$ shorter than $\widetilde{W}$, a contradiction. Thus W is a shortest Eulerian circuit on G, and we can compute it from $\widetilde{W}$ in constant time by removing the first and last edge traversed by $\widetilde{W}$. □

Theorem 5 *Any instance of DCP can be reduced to an instance of the* SHORTEST $\mathcal{S}_k$-EQUIVALENT STRING *problem in linear time.*

Proof Let $\mathcal{I}_{\text{DCP}} := G = (V, E)$ be an instance of DCP. We define $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ as in Lemma 1. A walk W in $\widetilde{G}$ can be expressed as a sequence $v_0, \ldots, v_{|W|}$ of nodes from $\widetilde{V}$ such that $(v_i, v_{i+1}) \in \widetilde{E}$ for every $i = 0, \ldots, |W| - 1$. Equivalently, such a walk W can be seen as a string $v_0 \ldots v_{|W|}$ over the alphabet $\widetilde{V}$ such that its set of length-2 substrings $\mathcal{S}_2(W)$ is included in $\widetilde{E}$. By definition, W is Eulerian if and only if $\mathcal{S}_2(W)$ is exactly $\widetilde{E}$, so finding a shortest Eulerian walk W in $\widetilde{G}$ corresponds to finding a shortest string over $\widetilde{V}$ such that $\mathcal{S}_2(W) = \widetilde{E}$, which is an instance of the SHORTEST $\mathcal{S}_2$- EQUIVALENT STRING problem. Finally, by Lemma 1, a solution to DCP on G can be obtained from an Eulerian walk W on $\widetilde{G}$ in constant time. □

3.2 Binary Alphabet

We reduce any instance of the DIRECTED CHINESE POSTMAN problem to an instance of SHORTEST $\mathcal{S}_k$-EQUIVALENT STRING over a binary alphabet $\Sigma = \{0, 1\}$.

For a given integer ℓ, we denote by 0^ℓ the string consisting of ℓ consecutive zeros. We assign to every $v \in V$ two binary strings v_A, v_B over $\{0, 1\}$ with the following properties:

1. For every $v \in V$, v_A and v_B are different from each other and from any w_A and w_B for $w \neq v$ in V, so that one can identify v by knowing only v_A or v_B.
2. All strings v_A and v_B start and end with a 1, and all have the same length ℓ.

Observation 1 The length ℓ can be chosen to be in $\mathcal{O}(\log |V|)$.

Proof We need two unique binary strings v_A, v_B for every $v \in V$. Since there exist 2^α distinct binary strings of length α, we seek the minimum length ℓ such that $2^{\ell-2} \geq 2|V|$, because we restrict to strings satisfying Property 2. The statement follows. □

Let $k = 3\ell$. Our reduction models both nodes and edges of G with appropriate string gadgets. The node gadgets are defined as length-k strings $s(v) := v_A \cdot 0^\ell \cdot v_B$, for every node $v \in V$. Let

$$S_0 := \{s(v) \mid v \in V\} \tag{1}$$

be the set of such gadgets. We model each edge $(u, v) \in E$ as a length-(7ℓ) string gadget $s(u) \cdot 0^\ell \cdot s(v)$. The edge gadget for (u, v) is schematically represented in Fig. 1. We define the set S of length-k strings input to SHORTEST $\mathcal{S}_k$-EQUIVALENT STRING as the set of all length-k *substrings* of the edge gadgets:

$$S := \cup_{(u,v)\in E} \mathsf{Substr}_k(s(u) \cdot 0^\ell \cdot s(v)). \tag{2}$$

Note that $S_0 \subseteq S$. Since $\ell = \mathcal{O}(\log |V|)$, each edge gadget has length $7\ell = \mathcal{O}(\log |V|)$ so it has $\mathcal{O}(\log |V|)$ substrings of length $k = 3\ell = \mathcal{O}(\log |V|)$. Therefore S contains $\mathcal{O}(|E| \log |V|)$ strings of length $\mathcal{O}(\log |V|)$.

Observation 2 All and only the occurrences of 0^ℓ within an edge gadget (i.e., within a string of S) start at position ℓ, 3ℓ, or 5ℓ of the gadget.

Proof Every string v_A or v_B starts and ends with a 1, so they cannot overlap an occurrence of 0^ℓ. Since we never concatenate two copies of 0^ℓ, the result follows. □

We now prove the correspondence between walks in G and strings having their length-k substrings in S. Lemma 2 is a first step in this direction: we show that given

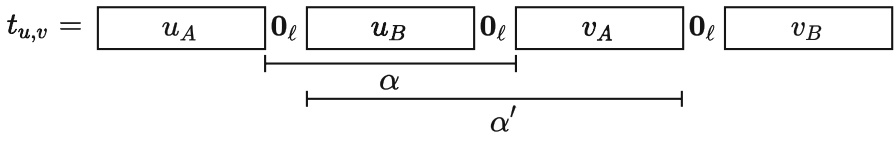

Fig. 1 The configuration described in the proof of Lemma 2

a string T with $\mathsf{Substr}_k(T) \subseteq S$, no two occurrences of node gadgets overlap in T, and the succession of such occurrences precisely corresponds to adjacency relations between nodes in G.

Lemma 2 *Let $G = (V, E)$ be a directed graph, ℓ be an integer, $k = 3\ell$, and let S be the set of length-k strings defined in (2). Let T be a string with $\mathsf{Substr}_k(T) \subseteq S$. If, for some $u, v \in V$, T has a substring $t_{u,v}$ such that (i) $t_{u,v}$ has $s(u)$ as a prefix, (ii) $t_{u,v}$ has $s(v)$ as a suffix, and (iii) $t_{u,v}$ has no other substrings of the form $s(w)$ for $w \in V$, then $t_{u,v} = s(u) \cdot 0^\ell \cdot s(v)$ and $(u, v) \in E$.*

Proof Let $t_{u,v}$ be a substring of T satisfying conditions (i)-(iii), so that $t_{u,v}[0..k-1] = s(u) = u_A \cdot 0^\ell \cdot u_B$ for some $u \in V$. Note that the length-k substring α starting at position ℓ of $t_{u,v}$ has 0^ℓ as a prefix. By the definition of S and Observation 2, every length-k string in S starting with 0^ℓ also ends with 0^ℓ, thus 0^ℓ is also a suffix of α (see Fig. 1). Let α' be the length-k substring starting at position 2ℓ of $t_{u,v}$. String α' has u_B as a prefix, and it has an occurrence of 0^ℓ at position ℓ (the suffix of α). Since $\alpha' \in S$, and since it has 0^ℓ in a central position, we know that α' is equal to $u_B \cdot 0^\ell \cdot w_A$ for some $w \in V$ such that $(u, w) \in E$. Let now β be the length-k substring of $t_{u,v}$ starting at position 3ℓ. We know that, since 0^ℓ is a prefix of β, it is also its length-ℓ suffix. Thus, the length-k substring β' starting at position 4ℓ of $t_{u,v}$ has $w_A \cdot 0^\ell$ as a prefix; by looking at S we find that $\beta' = s(w)$. Now, by definition of $t_{u,v}$, the string β' has to be $s(v)$. Thus, $w = v$, $(u, v) \in E$, and $t_{u,v} = s(u) \cdot 0^\ell \cdot s(v)$. □

Proposition 2 *Let $G = (V, E)$ be a directed graph, let S_0 and S be defined as in (1) and (2), and let $\mathcal{T}$ be the set of strings of length at least $k + 1$, having prefix and suffix in S_0 and such that $\mathsf{Substr}_k(T) \subseteq S$ for all $T \in \mathcal{T}$. The mapping*

$$\varphi : W = ((v_0, v_1), (v_1, v_2), \ldots, (v_{R-1}, v_R)) \mapsto T = s(v_0) \cdot 0^\ell \cdot s(v_1) \cdots 0^\ell \cdot s(v_R)$$

defines a bijection between the set of walks on G and $\mathcal{T}$. From any $T \in \mathcal{T}$, $\varphi^{-1}(T)$ can be computed in $\mathcal{O}(|T| + |V|)$ time. Moreover, given a walk W, the set of edges traversed by W is exactly the set of $(u, v) \in E$ such that $u_B \cdot 0^\ell \cdot v_A$ are substrings of $\varphi(W)$.

Proof The mapping is well defined and is injective by the definition of the gadgets. Consider an arbitrary string $T \in \mathcal{T}$. Let $u, v \in V$ be nodes such that $s(u) \in S_0$ is a prefix of T and $s(v) \in S_0$ is a suffix of T. Let t be the second leftmost occurrence of any string from S_0 in T (the prefix and suffix of T are in S_0, and since the length of T is at least $k + 1$, they do not coincide, thus there are at least two occurrences of strings from S_0 in T). We have $t = s(w)$ for some $w \in V$, and then T has a prefix $t_{u,w}$ satisfying conditions (i)-(iii) of Lemma 2. Then $(u, w) \in E$, and $t_{u,w} = s(u) \cdot 0^\ell \cdot s(w)$. By induction, we can find a sequence of nodes $u_0 = u, u_1 = w, \ldots, u_R = v$ such that $T = s(u_0) \cdot 0^\ell \cdots 0^\ell \cdot s(u_R)$, and such that $(u_i, u_{i+1}) \in E$ for every $i = 0, \ldots, R-1$. This gives us a walk $W = u_0, u_1, \ldots, u_R$ on G with $\varphi(W) = T$. The mapping φ is therefore surjective, hence it is a bijection. Its inverse map can be computed in $\mathcal{O}(|T|)$ time, by storing the encodings $s(v)$ for $v \in V$ in a dictionary in $\mathcal{O}(|V|)$ time, and linearly constructing the list of v from the occurrences of $s(v)$ in T. □

We are now ready to prove the main result of this section.

Theorem 6 *Any instance $G = (V, E)$ of* Directed Chinese Postman *with an output W can be reduced in $\mathcal{O}(|E|\log|V| + |W|)$ time to an instance of* Shortest $\mathcal{S}_k$- Equivalent String *with $n = |\mathcal{S}_k| = \mathcal{O}(|E|\log|V|)$ strings over the binary alphabet and $k = \mathcal{O}(\log|V|)$.*

Proof Let $G = (V, E)$ be a directed graph. We construct a graph $\widetilde{G}$ from G with bogus nodes a and b as in Lemma 1 and we construct sets S_0 and S on $\widetilde{G}$ as described at the beginning of this section. S contains $\mathcal{O}(|E|\log|V|)$ strings, each of them having length $\mathcal{O}(\log|V|)$ bits, and it can be constructed in $\mathcal{O}(|E|\log|V|)$ time by listing the binary encoding we assign to the nodes. We now prove that we can compute a shortest Eulerian walk in $\widetilde{G}$ from the output of Shortest $\mathcal{S}_k$-Equivalent String, and Lemma 1 concludes the proof.

Indeed, from the bijection in Proposition 2, every string in $\mathcal{T}$ gives rise to a unique walk in $\widetilde{G}$. Let T be a solution to Shortest $\mathcal{S}_k$-Equivalent String with $\mathcal{S}_k = S$. Since the node a does not have any ingoing edge in $\widetilde{G}$, by the definition of S, the length-$(k-1)$ prefix of $s(a)$ cannot be a suffix of any string in S. Since T contains every string in S, the string $s(a)$ must be a prefix of T. By symmetry, the string $s(b)$ is a suffix of T. Finally, since $s(a) \neq s(b)$ by definition of the node gadgets, T has length at least $k+1$ and therefore $T \in \mathcal{T}$.

Now, by Proposition 2 we can get in $\mathcal{O}(|T| + |V|)$ time a unique walk $W = v_0, \ldots, v_R$ such that $T = s(v_0) \cdot 0^\ell \cdot s(v_1) \cdots s(v_R)$. The edges traversed by W are exactly edges $(u, v) \in \widetilde{E}$ such that $u_B \cdot 0^\ell \cdot v_A$ is a substring of T. But since T contains every string from S, and since $u_B \cdot 0^\ell \cdot v_A \in S$ for every $(u, v) \in \widetilde{E}$, the walk W traverses every edge in $\widetilde{G}$ and is therefore Eulerian. Furthermore, it is minimal, as otherwise, some shorter Eulerian walk W' would give rise to $T' = \varphi(W')$ shorter than T. But then T' would be a shorter string with $\mathsf{Substr}_k(T') = S$. It follows that $W = \varphi^{-1}(T)$ is a shortest Eulerian walk of $\widetilde{G}$. By looking at the bijection of Proposition 2, we get $|T| = \mathcal{O}(|W|k) = \mathcal{O}(|W|\log|V|)$ bits, so the total reduction time is $\mathcal{O}(|E|\log|V| + |W|)$. □

4 Combinatorial Bounds

The main goal of this section is to prove Theorem 2; in particular, to provide bounds on the quantities $||\mathcal{W}_1||$ and $||\mathcal{W}_z||$ for a directed graph $G = (V, E)$. We recall that is the sum of the lengths of the shortest Eulerian walks of Section 4.1, we prove the upper bound $||\mathcal{W}_1|| \leq |V| \cdot |E|$, and show that it is asymptotically tight, even for the restricted class of dBGs constructed over a constant-sized alphabet (Section 4.1.1). In Section 4.2, we prove the bound $||\mathcal{W}_z|| \leq z|V| \cdot |E| + z^2|V|$, and show that it is asymptotically tight as well. In Section 4.3, we employ Theorem 2 and our other tightness results to prove Theorem 3. In some special cases, as described in Section 4.4, the latter bound can be reduced to $z|V| \cdot |E| + 2z\sqrt{z}$.

Firstly, let us observe that there are three types of graphs to consider.

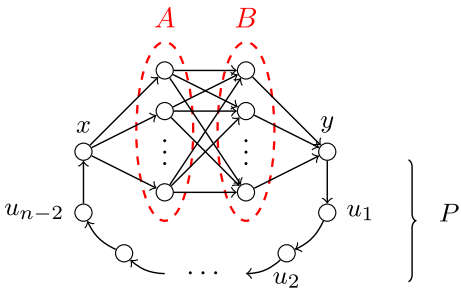

Fig. 2 A directed graph where the shortest Eulerian walk has length $\Omega(|V||E|) = \Omega(|V|^3)$

Observation 3 Every directed graph has either one, none, or infinitely many distinct Eulerian walks.

Proof Any directed path has exactly one Eulerian walk. It is easily seen that any other directed acyclic graph has none, and that any directed graph with two or more Eulerian walks must have a directed cycle[2]: this cycle may be repeated any number of times, yielding infinitely many distinct Eulerian walks. □

As the first two types of graphs are trivial with respect to this analysis, in the rest of the section we focus on graphs having infinitely many distinct Eulerian walks.

4.1 Length of a Shortest Eulerian Walk

Lemma 3 *For any directed graph $G = (V, E)$, we have $||\mathcal{W}_1|| \leq |V| \cdot |E|$.*

Proof Let us first assume that G is strongly connected. Letting $e_1, \ldots, e_{|E|}$ be the edges of E, we inductively build a walk that traverses all these edges in the given order and has length at most $|V| \cdot |E|$. The walk starts as a single edge e_1. Inductively, given a walk W' that includes $e_1, \ldots, e_{h-1}$ and traverses at most $|V| \cdot (h-1)$ edges for any $h \in [2, |E|]$, there is a walk W connecting the head of e_{h-1} to the tail of e_h, since G is strongly connected. Moreover, W traverses at most $|V| - 1$ edges: indeed, if W goes through the same node twice, we can remove the circuit that it forms doing so. As a result, $W'We_h$ is a walk that crosses $e_1, \ldots, e_h$ and traverses at most $|V| \cdot h$ edges. The claim follows when $h = |E|$, noting that a shortest walk cannot be longer than the one we have just described inductively.

Consider the general case of G and $\mathcal{W}_1 = \{W\}$. The walk W induces a topological order on the strongly connected components (SCCs) $G_1, G_2, \ldots, G_k$ of G, as otherwise the walk cannot traverse all the edges, and these SCCs form a chain graph of the form $G_1 \to G_2 \to \cdots \to G_k$ (see [13]). Since each $G_i = (V_i, E_i)$ is strongly connected, we can apply the above argument, so that W traverses at most $|V_i| \cdot |E_i|$ edges there. Overall, since $|E| = k - 1 + \sum_{i=1}^{k} |E_i|$, W traverses at most $k - 1 + \sum_{i=1}^{k} |V_i||E_i| \leq k - 1 + |V| \cdot \sum_{i=1}^{k} |E_i| \leq |V| \cdot |E|$ edges. □

In Lemma 4, we show that the bound of Lemma 3 is asymptotically tight by providing an example of a graph G with $||\mathcal{W}_1|| = \Omega(|V| \cdot |E|)$. With the same idea, we

[2] Note that some directed graphs with cycles may still allow no Eulerian walks.

will show that this can be done as well using de Bruijn graphs (see Section 4.1.1), hence proving Theorem 3.

Lemma 4 *There is an infinite family of directed graphs, such that each graph* $G = (V, E)$ *satisfies* $||\mathcal{W}_1|| = \Omega(|V|^3)$. *In particular,* $||\mathcal{W}_1|| = \Theta(|V| \cdot |E|)$.

Proof For any given $n \geq 3$, we construct a graph G attaining the stated bound as follows (see Fig. 2): we start from two sets A, B of n nodes each, and the set of n^2 edges $F = \{(a, b) \mid a \in A, b \in B\}$ (i.e., A, B induce a complete directed bipartite graph with edges directed from A to B). We then add two additional nodes x, y and add edges (x, a) for all $a \in A$, and edges (b, y) for all $b \in B$. We further connect y to x by adding $n - 2$ new nodes $u_1, \ldots, u_{n-2}$ and all edges of $P = \{(y, u_1)\} \cup \{(u_{n-2}, x)\} \cup \{(u_i, u_{i+1}) \mid i = 1, \ldots, n-3\}$. In total, this graph has $|V| = 3n$ nodes and $|E| = n^2 + 3n - 1 = \Theta(|V|^2)$ edges.

Let us now see why the length of a shortest Eulerian walk W of G is $\Omega(|V| \cdot |E|)$. By definition, $\mathcal{W}_1$ must traverse all the n^2 edges in the set of edges F. Without loss of generality, let $(a_1, b_1), \ldots, (a_{n^2}, b_{n^2})$ denote the order in which W traverses the edges in F. Let $i < n^2$, and consider the point where W has just traversed (a_i, b_i) for the first time: to reach the next (a_{i+1}, b_{i+1}), the walk must necessarily traverse the whole P. This means that W needs to traverse all $n - 1$ edges of P once for each $i = 1, \ldots, n^2 - 1$. Furthermore, it must also traverse at least once all the remaining $2n$ edges of the form (x, a) for all $a \in A$, and (y, b) for all $b \in B$. This leads to a total length of at least $(n-1)(n^2-1) + 2n = n^3 - n^2 + n + 1 = \Theta(|V| \cdot |E|)$ which for these graphs is $\Omega(|V|^3)$, proving the claim. □

We note that, while this lower bound is enough to prove the tightness of the bounds for the SHORTEST EULERIAN WALK problem, it is not enough for the SHORTEST $\mathcal{S}_k$-EQUIVALENT STRING problem. For the latter, we need to provide an infinite family of *de Bruijn graphs* over a constant-sized alphabet, which we show in Section 4.1.1.

4.1.1 Length of a Shortest Eulerian Walk on de Bruijn Graphs

Lemma 5 *There is an infinite family of de Bruijn graphs, such that each graph* $G = (V, E)$ *satisfies* $||\mathcal{W}_1|| = \Omega(|V| \cdot |E|)$.

Proof Let us fix $k \geq 3$. Consider the alphabets $\Sigma = \{0, 1, a, b\}$ and $\Sigma' = \{2, 3, 4, 5\}$ and the set of strings $\mathcal{S}_1 = a^k \cdot \{0, 1\}^{k-2} \cdot b^k$. Let us define the string $S_2 = a^k \cdot S_{\Sigma'} \cdot b^k$, where $S_{\Sigma'}$ is obtained by concatenating alternatively D distinct strings of length $\lfloor \frac{k}{2} \rfloor$ over the alphabet Σ', and a symbol $\$ \notin \Sigma \cup \Sigma'$ between each of those strings, for a parameter D to be determined later. For instance, given $k = 4$ and $D = 3$, $S_2 = aaaa \cdot 23\$35\$42 \cdot bbbb$ could be such a string. Note that $|S_2| = D(\lfloor \frac{k}{2} \rfloor + 1) - 1 + 2k$. We consider the graph $G = (V, E)$ defined as the dBG of the set $\mathcal{S} = \bigcup_{S \in \mathcal{S}_1 \cup \{S_2\}} \mathsf{Substr}_k(S)$. Figure 3 shows a construction of G.

Observe that each length-k substring of S_2 is distinct: each string $T \in \mathsf{Substr}_k(S_2)$ containing the letter a or b is uniquely determined by the number of occurrences of such letter, and every other string contains at least one occurrence of a string of the type $\$ \cdot s$ or $s \cdot \$$, where s is a string of length $\lfloor \frac{k}{2} \rfloor$ occurring at a single position in

S_2, thus uniquely determining the starting position of the occurrence of T in S_2. In particular, this means that $|\mathsf{Substr}_k(S_2)| = |S_2| - k + 1 = D(\lfloor \frac{k}{2} \rfloor + 1) + k$. To compute $|\bigcup_{S \in \mathcal{S}_1} \mathsf{Substr}_k(S)|$, observe that it contains the set $a^{k-i} \cdot \{0,1\}^i$ and the set $\{0,1\}^i \cdot b^{k-i}$ for each $0 \leq i \leq k-2$, as well as the set $a \cdot \{0,1\}^{k-2} \cdot b$, and that those sets are all disjoint. Hence, we obtain $|\bigcup_{S \in \mathcal{S}_1} \mathsf{Substr}_k(S)| = 2\sum_{i=0}^{k-2} 2^i + 2^{k-2} = 5(2^{k-2}) - 2$. Now, since $(\bigcup_{S \in \mathcal{S}_1} \mathsf{Substr}_k S) \cap \mathsf{Substr}_k(S_2) = \{a^k, b^k\}$ we obtain $|E| = |\mathcal{S}| = 5(2^{k-2}) + |S_2| - k - 3 = 5(2^{k-2}) + D(\lfloor \frac{k}{2} \rfloor + 1) + k - 4$. Observe as well (see Fig. 3) that after removing one edge with a label in $\mathsf{Substr}_k(S_2)$, and every

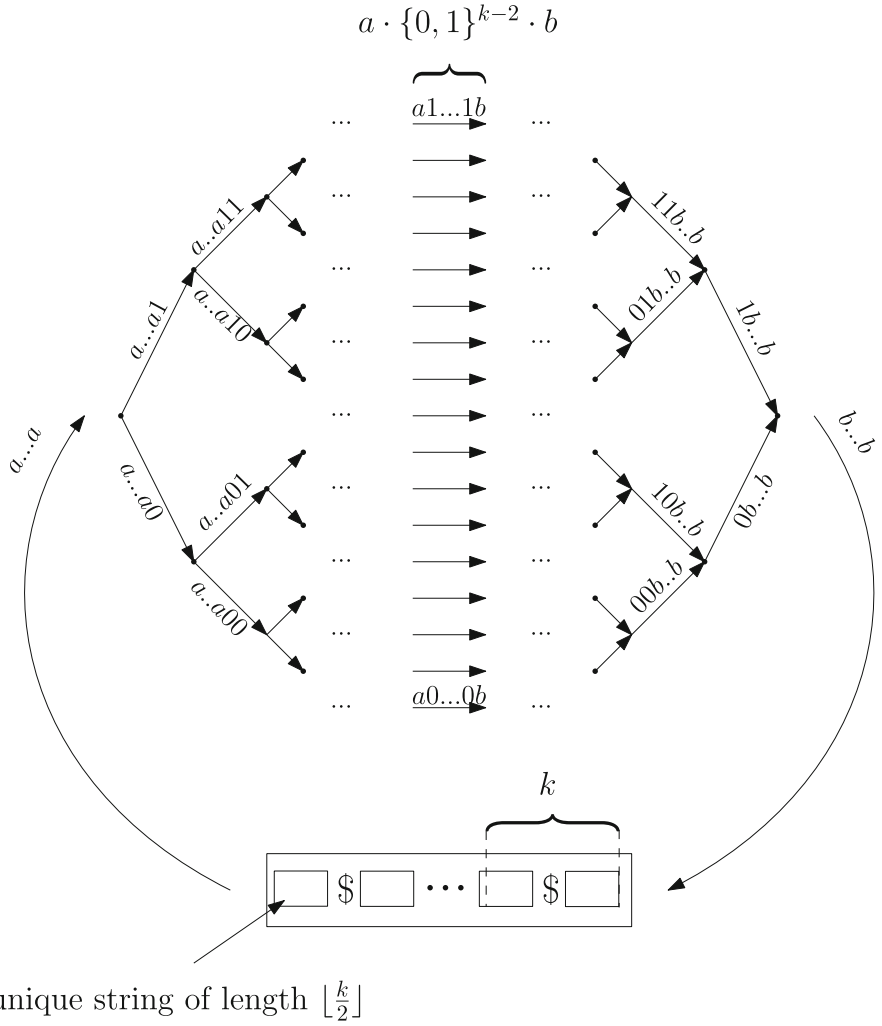

Fig. 3 Example of a de Bruijn graph $G = (V, E)$ satisfying $||\mathcal{W}_1|| = \Omega(|V||E|)$, constructed as in the proof of Lemma 5

edge with label in $a \cdot \{0,1\}^{k-2} \cdot b$, one obtains two disjoint trees that partition V. We deduce $|V| = |E| - 2^{k-2} + 1$.

To compute $||\mathcal{W}_1||$, observe that each of the 2^{k-2} edges with label in $a \cdot \{0,1\}^{k-2} \cdot b$ has to be visited, and each time one of such edges is visited, to proceed to the next one the whole part of the graph corresponding to $\mathsf{Substr}_k(S_2)$ must be traversed, plus a length-k path in each subtree corresponding to strings in the alphabet $\{0,1,a\}$ and $\{0,1,b\}$. Namely, the shortest path between any pair of the 2^{k-2} central edges has length $3k + D(\lfloor \frac{k}{2} \rfloor + 1) - 2$. This implies that $||\mathcal{W}_1|| \geq (2^{k-2} - 1)(3k + D(\lfloor \frac{k}{2} \rfloor + 1) - 2)$.

Since $|\Sigma'| = 4$, then one can generate at least $4^{\lfloor k/2 \rfloor}$ unique strings of length $\lfloor k/2 \rfloor$ on Σ', meaning we can set $D = \Theta(\frac{4^{\frac{k}{2}}}{k}) = \Theta(\frac{2^k}{k})$, from which we deduce $||\mathcal{W}_1|| = \Theta(2^{2k})$. On the other hand, one has $|E| = 5(2^{k-2}) + |S_2| - k - 3 = 5(2^{k-2}) + D(\lfloor \frac{k}{2} \rfloor + 1) + k - 4$, so that $|E| > 5(2^k)$ and $|E| = \Theta(2^k)$. Finally, we obtain $|V||E| = |E|(|E| - 2^{k-2} + 1) = \Theta(2^{2k})$. □

Observe how the restricted case of dBGs on a constant-sized alphabet gives us the same $\Omega(|V| \cdot |E|)$ lower bound as general graphs, proven in Lemma 4. On the other hand, differently from the general case, it is impossible to prove a $\Omega(|V|^3)$ lower bound for dBGs with a constant-size alphabet: as the outdegree is bounded by $|\Sigma|$, a constant alphabet implies $|E| = O(|V|)$, and thus $|V| \cdot |E| = \mathcal{O}(|V|^2)$. It remains open to determine if the lower bound for general $|\Sigma|$ falls closer to $\Omega(|V^2|)$ or to $\Omega(|V^3|)$.

4.2 Total Length of z Shortest Eulerian Walks

Lemma 6 *For any directed graph $G = (V, E)$, if $c_1 \leq \ldots \leq c_z$ are the lengths of the walks in $\mathcal{W}_z$, then $c_i \leq |V|(i - 1 + |E|)$, for all $i \in [1, z]$, and we have $||\mathcal{W}_z|| \leq z|V| \cdot |E| + z^2|V|$.*

Proof First observe that if G is acyclic, then it has at most one Eulerian walk, and by Lemma 3 its length is at most $|V| \cdot |E|$, so the claim holds for $z = 1$.

We can then assume that G has a cycle, and let C be the shortest one. Given an Eulerian walk W, the length of the next shortest walk is always bounded by $|W| + |C|$. In fact, let us consider a walk W' defined starting from W, to which we add a tour of cycle C when W traverses a node of C for the first time. The walk W' is Eulerian as it contains W, and has a length of exactly $|W| + |C|$, which thus bounds the length of the next shortest walk after W by $|W| + |C| \leq |W| + |V|$.

Now we can prove by a simple induction that the i-th shortest length for an Eulerian walk is at most $||\mathcal{W}_1|| + (i - 1)|V| \leq |V|(i - 1 + |E|)$, where the base case $i = 1$ is trivial (and bounded by Lemma 3) and the inductive step holds as the length of the shortest longer walk increases by at most $|V|$. Hence, $||\mathcal{W}_z|| \leq \sum_{i=1}^{z} (||\mathcal{W}_1|| + (i - 1)|V|) \leq z|V||E| + z^2|V|$. □

In Lemma 7 we show that the bound of Lemma 6 is asymptotically tight.

Lemma 7 *There exist infinite families of directed graphs, such that each graph $G = (V, E)$ satisfies either $||\mathcal{W}_z|| = \Omega(z|V| \cdot |E|)$ or $||\mathcal{W}_z|| = \Omega(z^2|V|)$, respectively.*

Furthermore, a family can be chosen to consist of de Bruijn graphs such that $||\mathcal{W}_z|| = \Omega(z|V| \cdot |E|)$ *or such that* $||\mathcal{W}_z|| = \Omega(z^2|V|)$.

Proof As clearly the i-th shortest Eulerian walk, for any i, is not shorter than the shortest one, we have $||\mathcal{W}_z|| \geq z \cdot ||\mathcal{W}_1||$, so the first family follows from Lemma 4 (shown in Fig. 2). As for the second, simply consider a directed cycle such that one of its nodes has two additional pendant edges, one incoming and the other outgoing. More formally, the directed cycle $(v_1, v_2), \ldots, (v_{|V|-2}, v_1)$ of length $|V| - 2$, plus two nodes x, y and the edges (x, v_1) and (v_1, y). Any Eulerian walk must start from the source node x, traverse the whole cycle at least once (say, $i > 0$ times), and then follow the edge (v_1, y). As different walks must traverse the cycle a different number of times, it follows that the i-th shortest Eulerian walk has length $2 + i(|V| - 2)$, so $||\mathcal{W}_z|| = \sum_{i=\{1,\ldots,z\}}(2 + i(|V| - 2)) = \Omega(z^2|V|)$.

Observe that each of these infinite families contains an infinite number of de Bruijn graphs: for the first bound, this clearly follows from Lemma 5. For the second, observe that we can obtain a simple cycle by taking the set $\mathsf{Substr}_k(S)$ for a string S such that $S[1..k] = S[|S| - k + 1, |S|]$, and S has no other repeating length-$(k-1)$ substring. This can be done by choosing S to be a de Bruijn sequence of order $k-1$ over a binary alphabet, to obtain a cycle of length 2^{k-1}. For the two extra edges, one can simply add a bogus letter \$, pick a node v having label ℓ and add the two strings $\$ \cdot \ell$ and $\ell \cdot \$$. □

We can conclude from Lemma 7 that a better worst-case bound than $\Omega(z|V| \cdot |E| + z^2|V|)$ is not possible, so Lemmas 3, 4, 6, 7 yield directly Theorem 2.

4.3 Total Length of z Shortest Equivalent Strings

Let us now prove Theorem 3, which infers upper bounds on the total length of a solution to z-SHORTEST $\mathcal{S}_k$-EQUIVALENT STRINGS. We use the following observation:

Observation 4 Let $\mathcal{S}_k$ be a set of n strings each of length k, such that there exists a string T with $\mathsf{Substr}_k(T) = \mathcal{S}_k$. Let $G = (V, E)$ be the dBG of order k of $\mathcal{S}_k$. It holds (i) $|E| = n$; and (ii) $|V| \leq n + 1$.

Proof (i) It follows from the definition of dBG: each edge corresponds to a string from $\mathcal{S}_k$ and reciprocally, if $S \in \mathcal{S}_k$ there is an edge between $S[0..k-2]$ and $S[1..k-1]$, which corresponds to S. Therefore $|E| = |\mathcal{S}_k| = n$.

(ii) Since there exists T with $\mathsf{Substr}_k(T) = \mathcal{S}_k$, we know that there is a walk W traversing every edge in G. It follows that all $e \in E$ (except possibly for the first and last edges traversed by W) are such that the head of e coincides with the tail of some $e' \in E$, and symmetrically, the tail of e coincides with the head of some $e'' \in E$. We conclude that there cannot be more than $|E| + 1$ distinct nodes, thus $|V| \leq |E| + 1 = n + 1$.

□

Let $\mathcal{S}_k$ be a set of n strings each of length k; we know that if $G = (V, E)$ is the dBG of order k of $\mathcal{S}_k$, a (shortest) string T such that $\mathsf{Substr}_k(T) = \mathcal{S}_k$ has length $k-1+|W|$, where W is a (shortest) Eulerian walk on G. From Theorem 2 and Observation 4 we

get that $||\mathcal{W}_1|| \leq |V| \cdot |E| \leq n^2 + n$ and so $||\mathcal{T}_1|| \leq k - 1 + n^2 + n = \mathcal{O}(k + n^2)$. Using again Theorem 2 and Observation 4, we get that $||\mathcal{W}_z|| \leq z|V| \cdot |E| + z^2|V|$ and so $||\mathcal{T}_z|| \leq z(k-1+n^2+n) + z^2(n+1) = \mathcal{O}(zk + zn^2 + z^2n)$. Lemmas 5 and 7 give the tightness of those bounds. We have arrived at Theorem 3.

4.4 Total Length of z Shortest Eulerian Walks in Graphs Without Sources and Sinks

While the upper bound on the total length $||W_z||$ of the z shortest Eulerian walks described in the previous section is tight, in this section we describe how graphs respecting relatively weak constraints allow for a smaller upper bound. Namely, we will show that $||W_z|| = \mathcal{O}(z|V||E| + z\sqrt{z})$ for any directed graph G satisfying the following conditions:

1. G admits an Eulerian walk W;
2. G contains neither a source nor a sink;
3. G is not a simple cycle.[3]

Intuitively, under such conditions, one may extend W by just one edge at a time from either end, in multiple combinations, without the beginning or the end of the walk becoming "stuck" in a source or a sink, respectively. However, the tricky part is ensuring that different extensions always lead to distinct walks.

To do so, let us further characterize this class of graphs.

Lemma 8 *Let $G = (V, E)$ be a directed graph allowing an Eulerian walk, and W a shortest Eulerian walk on G. The following conditions are equivalent:*

1. *The graph contains neither a source nor a sink, and is not a simple cycle*
2. *There is a non-Eulerian cycle[4] containing $s = I(W)$, and a non-Eulerian cycle (possibly the same) containing $t = L(W)$.*

Proof Let W be an Eulerian walk in a graph $G = (V, E)$ containing neither a source nor a sink. Let s, t be respectively the starting and ending nodes of W. We first prove that (1) implies (2). By construction, the node s admits at least one outgoing and one ingoing edge; let e_o (resp. e_i) the first outgoing (resp. ingoing) edge from s that is visited by W. By definition, the first sequence of edges visited by W starting with e_o and ending with e_i is equal to a circuit containing s, which gives us a cycle containing s by removing any possible node repetition. By a symmetrical argument, the node t is also contained in a cycle. Assume that at least one of those cycles, call it C, visits every edge in G (namely, if it is Eulerian), and assume without loss of generality that it is the cycle containing s. Then, if a node v exists with an outdegree of at least 2, it must be visited at least twice (once for each outgoing edge). But this implies the existence of two sub-cycles C_1 and C_2, where C_1 is the cycle traversed between the first and the last visit of v, and where C_2 is obtained from C by skipping C_1. By construction, one of the two cycles, C_1 or C_2, must contain s, hence there exists a

[3] A graph is a simple cycle if it is isomorphic to the graph C_n for some n, namely if its nodes can be numbered $v_1, .., v_n$ such that (v_i, v_j) with is an edge if and only if $j = i + 1 \mod n$.

[4] I.e., that does not traverse every edge of the graph.

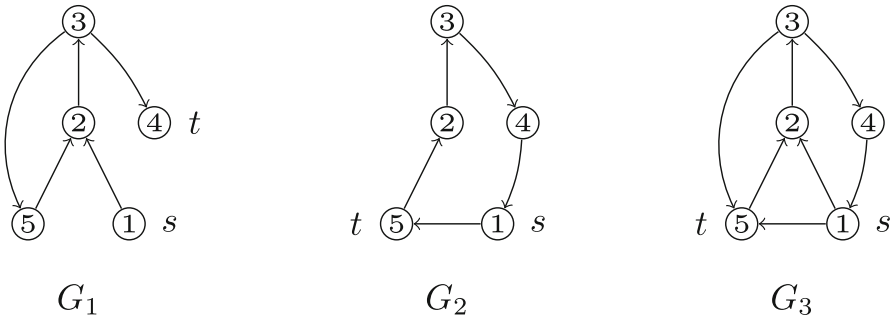

Fig. 4 Left and center: graphs that do *not* satisfy the conditions of Lemma 8; the shortest Eulerian walk of G_1 is 1235234, and any new walk requires repeating the entire cycle 235. A shortest walk of G_2 for the given s, t is 1523415, however it is easy to see that adding multiples of 5 edges either in the front or the back of such walk yields the same trail (i.e., in the notation of Lemma 9, $W(0,5) = W(5,0) = 152341523415$). Right: a graph G_3 satisfying the conditions of Lemma 8. A shortest Eulerian walk for G is $W = 123415235$, where only edge $(2,3)$ is repeated twice. Let us illustrate Lemma 9 on G_3: let $C_1 = 12341$ and $C_2 = 5235$ be the two cycles containing $s = 1$ and $t = 5$ respectively; for $i = 2$, $j = 3$ we have $S(2) = 341$ and $T(4) = 52352$, producing Eulerian walk $W(i,j) = 341234152352352$

cycle containing s which does not visit every edge. If every node has outdegree 1, then it also has indegree 1 (since it is Eulerian, one can use Proposition 1), and G is a simple cycle, a contradiction. Conversely, given a graph G containing an Eulerian walk W and respecting condition (2), observe how it cannot contain either a source or a sink, since only the first and last nodes of W (i.e., s and t) may potentially be a source and a sink, but they are not, because they are both contained in cycles. Furthermore, if those cycles do not visit every edge, then G contains two distinct cycles, and so it is not a simple cycle itself. □

Remark 1 The conditions given in Lemma 8 do not necessarily hold for dBGs in general. First, note that a dBG is a simple cycle if and only if it corresponds to the length-k substrings of a circular string, such that no length-k substring appears twice. Furthermore, a dBG for a set $\mathcal{S}_k$ of length-k strings has a source for each $X = X[0]\dots X[k-1] \in \mathcal{S}_k$ such that $X[0]\dots X[k-2]$ is not a suffix of any other $Y \in \mathcal{S}_k$ (symmetrically for sinks and prefixes)

Thus, a dBG for a set $\mathcal{S}_k$ of length-k strings satisfies the conditions of the lemma if and only if:

1. for each $X \in \mathcal{S}_k$, there is at least one $P \in \mathcal{S}_k$ such that $P[1]\dots P[k-1] = X[0]\dots X[k-2]$ (X is not a source);
2. for each $X \in \mathcal{S}_k$, there is at least one $S \in \mathcal{S}_k$ such that $X[1]\dots X[k-1] = S[0]\dots S[k-2]$ (X is not a sink);
3. there is at least one $X \in \mathcal{S}_k$ for which either condition 1 or 2 is satisfied for at least two other P or S in $\mathcal{S}_k$ (it has either indegree or outdegree of at least 2).

Lemma 9 *Let $G = (V, E)$ be a graph admitting at least one Eulerian walk, and satisfying the equivalent conditions of Lemma 8. Let W be a shortest Eulerian walk for G, which starts at node s and ends at node t. Let (C_1, C_2) be a pair of non-Eulerian*

cycles containing respectively s and t (note that one might have $C_1 = C_2$), and given two integers i, j let $S(i)$ be the sequences of i adjacent edges from C_1 ending at node s, $T(j)$ the sequence of j adjacent edges from C_2 starting at node t, and let $W(i, j)$ be the walk obtained by concatenating $S(i) \cdot W \cdot T(j)$ (see Fig. 4*). The mapping $(i, j) \mapsto W(i, j)$ is injective, i.e., two distinct pairs $(i, j), (i', j')$ correspond to distinct walks.*

Proof Assume that for two pairs $(i, j), (i'j')$ it holds $W(i, j) = W(i', j')$; we show that in this case, $(i, j) = (i'j')$. By construction, $W(i, j)$ is obtained adding i edges before s (from C_1) and j after t (from C_2), so $|W(i, j)| = |W| + i + j = |W(i', j')| = |W| + i' + j'$, and $i + j = i' + j'$. Moreover, let e be the earliest edge in W that does not appear in C_1 (e exists because C_1 is not Eulerian), and say e it the h-th node of W. Again by construction, the first occurrence of e in $W(i, j)$ must appear at position $i + h$, and since $W(i, j) = W(i', j')$, the same is true for $W(i', j')$. It follows that $W(i', j')$ was built adding i edges before s, so $i = i'$. Finally, as $i + j = i' + j'$, we also have $j = j'$, meaning $(i, j) = (i'j')$. □

We are now ready to prove the main result of the section:

Theorem 7 *Let G be a graph containing no sources, no sinks, and that is not a simple cycle, and let z be an integer. Then $||W_z|| = \mathcal{O}(z|V||E| + z\sqrt{z})$.*

Proof We use Lemma 8 to construct different walks of a given length. For a given n, we have $n + 1$ different pairs of nonnegative integers (i, j) such that $i + j = n$, giving us n distinct walks $W(i, j)$ of length $||W|| + n$, where W is a shortest Eulerian walk for G. For a given z, let a be the smallest integer such that $z \leq \sum_{i=0}^{a} i$: we have $a = \mathcal{O}(\sqrt{z})$. The set of all $W(i, j)$ with $i + j \leq a$ gives us at least z distinct Eulerian walks, each of length $|W| + i + j$. Hence we obtain $||W_z|| \leq \sum_{n \leq a} (n + |W|)(n + 1) = \mathcal{O}(a^3 + a^2|W|) = \mathcal{O}(z\sqrt{z} + z|V||E|)$ (using Theorem 2). □

Observe that the conditions from Lemma 8 are quite general, and in particular, are satisfied by all strongly connected graphs, except simple cycles.

Corollary 1 *Let G be a strongly connected graph that is not a simple cycle, and let z be an integer. Then $||W_z|| = \mathcal{O}(z|V||E| + z\sqrt{z})$.*

5 Listing z Shortest Eulerian Walks

The main goal of this section is to prove Theorem 1, and to provide an efficient algorithm for solving z-SHORTEST EULERIAN WALKS. In Section 5.1, we start by providing the state-of-the-art results for listing z best flows in a directed graph; the underlying algorithms form the main computational routine of our algorithm, which is provided in detail next. Finally, in Section 5.2, we employ Theorem 1 and Theorem 3 to prove Theorem 4; namely, to provide an efficient algorithm for solving z-SHORTEST $\mathcal{S}_k$-EQUIVALENT STRINGS.

5.1 Main Algorithm

We start by recalling the definition of flow.

Definition 1 (Flow) Let $G = (V, E)$ be a directed graph and $\delta : V \to \mathbb{Z}$ be a function called the *supply*. Let m (resp. M) be a function $E \to \mathbb{N} \cup \{+\infty\}$ called *minimal* (resp. *maximal*) capacity. A *flow* on G with supply δ, minimal capacity m and maximal capacity M is a function $f : E \to \mathbb{N}$ such that:

$$\forall e \in E,\ m(e) \leq f(e) \leq M(e)$$
$$\forall v \in V,\ \sum_{e=(v,w)\in E} f(e) - \sum_{e=(w,v)\in E} f(e) = \delta(v)$$

We denote this set of flows by $\mathcal{F}(G, \delta, m, M)$.

In general, we can equip a graph $G = (V, E)$ with a *cost* function $C : E \to \mathbb{N}$. Given a flow $f \in \mathcal{F}(G, \delta, m, M)$ for some supply function δ, and minimal (resp. maximal) capacities function m (resp. M), the *cost of* f is $C(f) = \sum_{e\in E} C(e)f(e)$. Let us now formally define the problem of listing z best flows (i.e., flows of minimal cost) from the set $\mathcal{F}(G, \delta, m, M)$ of all feasible flows (with respect to the given conditions) in a directed graph.

z- BEST FLOWS
Input: A directed graph $G = (V, E)$, a supply function δ, a minimal capacity function m, a maximal capacity function M, a cost function C (all taking finite values), a min-cost flow f, and an integer $z > 0$.
Output: A list F of z flows in $\mathcal{F}(G, \delta, m, M)$, such that for every flow f' not in F, $C(f') \geq C(F[i])$, for all $i \in [0, z-1]$, and ordered such that $C(F[0]) \leq \cdots \leq C(F[z-1])$; or FAIL if that is not possible.

A min-cost flow f can be computed for any G using one of the following results.

Lemma 10 ([4, 5, 45, 46]) *Given any directed graph $G = (V, E)$, computing a min-cost flow takes $\mathcal{O}((|E|\log|V|)(|E|+|V|\log|V|))$ time or $\mathcal{O}(|E||V|\log|V|\log K|V|)$ time, where K is the maximum cost of any edge of G.*[5]

The following recent result for the z- BEST FLOWS problem is known [47]; see also [48, 49].

Lemma 11 ([47]) *The z- BEST FLOWS problem can be solved in $\mathcal{O}(z|V|^3)$ time or in $\mathcal{O}(z(|E||V| + |V|^2\log|V|))$ time.*

Main Idea We list Eulerian walks as follows: we construct semi-Eulerian multigraphs G_μ, which are extensions of G obtained by duplicating some edges. Recall that each Eulerian walk in G can be seen as a trail in the semi-Eulerian extension G_μ for

[5] We do not use the most recent $|E|^{1+o(1)}$-time algorithm [6] for computing the min-cost flow, since Lemma 11 (for solving the z- BEST FLOWS problem) anyway has an $\mathcal{O}(|E||V|)$ term in the time complexity. Moreover, as we have shown in Lemma 4, $||\mathcal{W}_1|| = \Omega(|V| \cdot |E|)$, as well as $||\mathcal{W}_1|| = \Omega(|V|^3)$.

some multiplicity function μ such that, for every edge e, $\mu(e) = \mathsf{mult}_e(W)$. We will therefore treat Eulerian walks on G and Eulerian trails on G_μ as if they were the same objects, and use the fact that these semi-Eulerian extensions of G form a partition of the sets of Eulerian walks on G.

Our algorithm has two steps: first, listing the extensions, and then, listing the trails in each extension, using the algorithm given in [13]. For the first part, we need the following:

Proposition 3 *Let $G = (V, E)$ be a directed graph and let $M = \mathbb{N}^E$ be the set of all possible multiplicity functions on G. We have the following set equality:*[6]

$$EW(G) = \bigsqcup_{\mu \in M} ET(G_\mu) \tag{3}$$

Proof An Eulerian walk W on G is an Eulerian trail on the semi-Eulerian multigraph obtained by taking as many copies of each edge as the number of times W traverses it. □

To list semi-Eulerian extensions, we will use flows. Indeed, for Eulerian extensions, the balancing conditions (Proposition 1) are precisely a flow problem.

For any directed graph $G = (V, E)$ and for $v \in V$, we write $\mathbb{1}_v$ for the indicator function on V of v; namely, for every $u \in V$, $\mathbb{1}_v(u) = 1 \iff u = v$. We define:

- For every $v, w \in V$, a supply function $\delta_{v,w} : u \mapsto \mathbb{1}_v(u) - \mathbb{1}_w(u)$. If $v = w$, this is the null function, and if $v \neq w$, the function has value 1 on v, -1 on w, and 0 on any other node.
- For every $n \in \mathbb{N} \cup \{\infty\}$, a function c_n on E constantly equal to n.

Observation 5 An extension G_μ of a directed weakly connected graph G with multiplicities μ is semi-Eulerian if and only if

$$\mu \in \bigcup_{(v,w) \in V^2} \mathcal{F}(G, \delta_{v,w}, c_1, c_\infty)$$

Proof This is a reformulation of the balancing conditions (Proposition 1). □

The semi-Eulerian extensions of G correspond exactly to flows on G with supply function of the form $\delta_{v,w}$, with $(v, w) \in V^2$. However, we would like to treat all cases by solving a single flow problem in order to avoid solving $\mathcal{O}(|V|^2)$ of them separately.

Let $G = (V, E)$ be a directed graph, and $G_{s,t} = (V_{s,t}, E_{s,t})$ be an extension of G with $V_{s,t} = V \cup \{s, t\}$ for some $s, t \notin V$ and $E_{s,t} = E \cup (\bigcup_{v \in V}(s, v)) \cup (\bigcup_{v \in V}(v, t))$. That is, we add two bogus nodes, s and t, respectively connected to, and from, all nodes of G. We will compute flows on $G_{s,t}$ by defining a maximal capacity function equal to c_∞ on every edge, a minimal capacity function m_c with $m_c|_E = c_1$ and

[6] We use squared cup to underline that the sets $ET(G_\mu)$ are pairwise disjoint.

$m_c|_{(E_{s,t}\setminus E)} = c_0$ (i.e., one on all edges of the original graph, and zero on new bogus edges), and a supply function $\delta_{s,t}$.

Because we set the *minimal* capacity to be 1 on every edge in E, and since the flow can enter and exit G from any node via the edges from s to every $v \in V$, there is almost an equivalence between flows in $\mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty)$ and semi-Eulerian extensions of G, except for the following detail: a flow also contains information about the starting and ending point of some trail in the extension, as it specifies which nodes are connected to the bogus nodes. So multiple flows that differ only in the edges they use to connect to the bogus nodes correspond to the same extension of G: this happens when the extension is in fact Eulerian.

Proposition 4 *Let $G = (V, E)$ be a directed graph and let $f \in \mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty)$. There is a unique $i \in V$ such that $f(s, i) = 1$, and a unique $o \in V$ such that $f(o, t) = 1$. The flow f takes value 0 on every other edge with tail s or head t.*

Proof The node s has supply 1. Since there are no ingoing edges for s, the outgoing flow has to be 1, so only one node i has $f(s, i) = 1$. The node t has supply -1. Since there are no outgoing edges for t, the ingoing flow has to be 1, so only one node o has $f(o, t) = 1$. □

We call $i = \mathbf{en}(f)$ the *entrance* of f and $o = \mathbf{ex}(f)$ its *exit*. Note that $\mathbf{en}(f)$ and $\mathbf{ex}(f)$ are not necessarily distinct. We can now formally describe the relation between the flows in $\mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty)$ and the walks in G. We will make use of a function $\mathsf{WalkToFlow}$ that takes a walk on G and returns a specific flow on $G_{s,t}$, as defined in the following proposition.

Proposition 5 *Let $G = (V, E)$ be a directed weakly connected graph and $G_{s,t}$ the graph extended with the additional nodes s, t, as defined above. For each Eulerian walk W on G with multiplicity function μ, there is a unique flow $\mathsf{WalkToFlow}(W) \in \mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty)$ such that (i) $\mathsf{WalkToFlow}(W)|_E = \mu$, (ii) $\mathbf{en}(\mathsf{WalkToFlow}(W)) = I(W)$ and (iii) $\mathbf{ex}(\mathsf{WalkToFlow}(W)) = L(W)$. For every multiplicity function μ, we then have the following partition:*

$$ET(G_\mu) = \bigsqcup_{\substack{f \in \mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty) \\ f|_E = \mu}} \mathsf{WalkToFlow}^{-1}(f)$$

In particular, the sets $\mathsf{WalkToFlow}^{-1}(f)$ for $f \in \mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty)$ are nonempty and pairwise disjoint.

Proof Let μ be a multiplicity function such that an Eulerian walk W with multiplicity μ exists on G, or in other terms, that G_μ is semi-Eulerian. From Observation 5 we know that μ is a flow in $\mathcal{F}(G, \delta_{v,w}, c_1, c_\infty)$ for some nodes $v, w \in V$, namely $v = I(W)$,

$w = L(W)$. It is easy to verify that conditions (i)-(iii) uniquely define the following flow:

$$\mathsf{WalkToFlow}(W) : e \mapsto \begin{cases} \mu(e) & \text{if } e \in E \\ 1 & \text{if } e = (s, I(W)) \text{ or } e = (L(W), t) \\ 0 & \text{otherwise} \end{cases}$$

Now, for every flow $f_0 \in \mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty)$, the set $\mathsf{WalkToFlow}^{-1}(f_0)$ is the set of Eulerian trails in $G_{(f_0)|_E}$ starting at $\mathbf{en}(f_0)$ and ending at $\mathbf{ex}(f_0)$. To get all the Eulerian trails having a given multiplicity μ, we need to consider all the flows agreeing with μ on G, regardless of their entrance or exit. Hence we have:

$$ET(G_\mu) = \bigsqcup_{\substack{f \in \mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty) \\ f|_E = \mu}} \mathsf{WalkToFlow}^{-1}(f)$$

The sets $\mathsf{WalkToFlow}^{-1}(f)$ for $f \in \mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty)$ are pairwise disjoint because they are defined as reciprocal sets for the mapping $\mathsf{WalkToFlow}$, and are nonempty. Indeed, for a flow $f \in \mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty)$, the balancing conditions hold in $G_{f|_E}$ (Observation 5). One can then choose an Eulerian trail W in $G_{f|_E}$ starting at $\mathbf{en}(f)$ and ending at $\mathbf{ex}(f)$. We then obtain a partition of $ET(G_\mu)$. □

To compute shortest walks, we need to assign costs to the flows, which are equal to the lengths of the corresponding walks. This is achieved by defining a cost function C_{walks} on $G_{s,t}$ such that $C_{\text{walks}}|_E = c_1$ (the function constantly equal to 1), and $C_{\text{walks}}|_{E_{s,t} \setminus E} = c_0$ (the function constantly equal to 0). Note that this definition coincides with the definition of the minimal capacity function m that we use, but we distinguish the two functions for clarity.

Observation 6 Let us equip graph $G_{s,t}$ with the cost function C_{walks}. For any Eulerian walk W on G, the total cost of the flow $\mathsf{WalkToFlow}(W)$ on $G_{s,t}$ is the length of W.

Proof By the definition of $\mathsf{WalkToFlow}$ (Proposition 5), for a walk W of length ℓ we have

$$\sum_{e \in E_{s,t}} \mathsf{WalkToFlow}(W)(e) C(e) = \sum_{e \in E} \mathsf{WalkToFlow}(W)(e) = \ell$$

□

Proposition 6 *Let $G = (V, E)$ be a directed weakly connected graph. It holds*

$$EW(G) = \bigsqcup_{f \in \mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty)} \mathsf{WalkToFlow}^{-1}(f)$$

Proof It follows directly from Propositions 3 and 5. □

Let $G = (V, E)$ be a directed graph, $z \geq 1$ be an integer, and $A \in \mathbb{N} \cup \{\infty\}$. By $\mathcal{F}_{z,A}$ we denote the $\min(z, |\mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_A)|)$ minimal cost flows (with cost function C_{walks} and an arbitrary but fixed order on flows having the same total cost) in $\mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_A)$.

We now prove that we can list z shortest Eulerian walks in a directed graph G by computing only $\mathcal{F}_{z,\infty}$.

Proposition 7 *Let $G = (V, E)$ be a directed weakly connected graph. Let us write $EW_z(G)$ for the set containing the $\min(z, |EW(G)|)$ shortest Eulerian walks in G. It holds that:*

$$EW_z(G) \subseteq \bigcup_{f \in \mathcal{F}_{z,\infty}} ET(G_{(f_{|E})})$$

Proof We know from Observation 5 that $ET(G_\mu) \neq \emptyset$ if and only if $\mu = f_{|E}$ with $f \in \mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_A)$, and that, from Proposition 5, given $f, f' \in \mathcal{F}_{z,\infty}$ with $\mu = f_{|E} = f'_{|E}$, the sets $\mathsf{WalkToFlow}^{-1}(f)$ and $\mathsf{WalkToFlow}^{-1}(f')$ are disjoints sets of Eulerian trails in G_μ. Hence the set $M := \bigcup_{f \in \mathcal{F}_{z,\infty}} ET(G_{(f_{|E})})$ contains at least $|\mathcal{F}_{z,\infty}|$ distinct Eulerian walks on G. If $|\mathcal{F}_{z,\infty}| = z$, then M contains at least z Eulerian walks, which have minimal length from Observation 6. If $|\mathcal{F}_{z,\infty}| < z$, then $\mathcal{F}_{z,\infty} = \mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_A)$, and the result follows directly from Proposition 6 as $M = EW(G)$ in that case. □

From Proposition 7, we obtain the 2 steps of our algorithm for z-SHORTEST $\mathcal{S}_k$-EQUIVALENT STRINGS: we first compute the set $\mathcal{F}_{z,\infty}$: this can be done by finding a suitable integer M such that $\mathcal{F}_{z,M} = \mathcal{F}_{z,\infty}$ and using Lemma 11 to compute $\mathcal{F}_{z,M}$. The second step is to compute $ET(G_{(f_{|E})})$ for each $f \in \mathcal{F}_{z,M}$ (stopping either when z walks are found, or when all of them have been computed). This is done by using the algorithm from [13].

In order to compute flows in $\mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty)$ by means of existing algorithms, we need to define an equivalent problem having finite maximal capacity on each edge, as the known algorithms are not constructed for infinite maximal capacities.

Lemma 12 *Let $G = (V, E)$ be a directed graph and $z \geq 1$ be an integer. We have that $\mathcal{F}_{z,\infty} = \mathcal{F}_{z,|V|(z-1+|E|)}$.*

Proof Let $F = f_0, \ldots$ (resp. $F' = f'_0, \ldots, f'_N$) be the list of feasible flows for the maximal capacity function c_∞ (resp. $c_{|V|(z-1+|E|)}$), ordered by increasing cost and such that the order on flows having the same total cost is arbitrary but fixed. Note that the list F may be infinite. Each flow in the list F' is trivially in the list F. If $F = F'$ the lemma holds. Otherwise, let f_i be the minimal cost flow in F which is not in F', so that $f_j = f'_j$ for every $j < i$. If $i \leq z$, then by Proposition 6 there is at least one Eulerian walk W in $\mathsf{WalkToFlow}^{-1}(f_i)$, and this walk has length $C_{\text{walks}}(f_i)$. But from Lemma 6, the length of W is also at most $|V|(z - 1 + |E|)$, so $f_i \in F'$ (since the flows which appear strictly before in the list F' are less than z), which gives a contradiction. Therefore the first mismatch (if any) between F and F' has to be after the z first elements, and $\mathcal{F}_{z,\infty} = \mathcal{F}_{z,|V|(z-1+|E|)}$. □

Algorithm 1 EW-List($G = (V, E), z$).

1: **if** G is not weakly connected **then**
2: **return FAIL**
3: **end if**
4: $s \leftarrow |V| + 1$; $t \leftarrow |V| + 2$; $V_{s,t} \leftarrow V \cup \{s, t\}$ ▷ Add two bogus nodes
5: $E_{s,t} \leftarrow E \cup \{(s, v), (v, t) \mid v \in V\}$ ▷ Add $2|V|$ bogus edges
6: Construct $G_{s,t}, \delta_{s,t}, m_c, c_{|V|(z-1+|E|)}$, and C_{walks} accordingly
7: $f \leftarrow$ Best-Flow$(G_{s,t}, \delta_{s,t}, m_c, c_{|V|(z-1+|E|)}, C_{\text{walks}})$ ▷ **Lemma 10**
8: $F \leftarrow z$-Best-Flows$(G_{s,t}, \delta_{s,t}, m_c, c_{|V|(z-1+|E|)}, C_{\text{walks}}, f, z)$ ▷ **Lemma 11**
9: $F_s \leftarrow$ list$(f|_E$ **for** $f \in F)$ ▷ We keep only the values flows take on the initial graph
10: Remove duplicates in F_s if any ▷ Duplicates might arise from the loss of information
11: EW $\leftarrow \emptyset$; $i \leftarrow 0$;
12: **while** |EW| $< z$ **do**
13: EW $\leftarrow$ EW $\cup$ {ET-List$(G_{F_s[i]}, z - |\text{EW}|)$} ▷ List Eulerian trails on extended G [13]
14: $i \leftarrow i + 1$; ▷ Retrieve the next best flow
15: **if** $i = |F_s|$ **then** ▷ We have consumed all best flows
16: **break**;
17: **end if**
18: **end while**
19: **if** |EW| $\geq z$ **then**
20: **return EW**
21: **else**
22: **return FAIL**
23: **end if**

Lemma 13 *Given a directed graph $G = (V, E)$ and an integer z, Algorithm 1 terminates and solves the z-*Shortest Eulerian Walks *problem on G.*

Proof Algorithm 1 computes z flows with minimal cost in $\mathcal{F}(G_{s,t}, \delta_{s,t}, m_c, c_\infty)$ (for the cost function C_{walks}) as defined in Proposition 7, or all of them if there are less than z (Line 8), from Lemma 12. The set F_s is defined in Lines 9 and 10 as in Proposition 7; and in Line 13, the function ET-List computes, if they exist, the $z - |EW|$ shortest elements from $\text{EW}(G_{f|_E})$ for every $f \in F_s$ (where $|EW|$ is, at each step, the number of Eulerian walks already computed), so we know that no more than z Eulerian walks are computed in the end. The function is implemented by means of the algorithm provided in [13]. The correctness then directly follows from the equivalence proved in Proposition 7. □

Lemma 14 *Given a directed graph $G = (V, E)$ and an integer z, Algorithm 1 requires:*

- $\mathcal{O}(|E||V| \log^2 |V| + z|V|^3 + ||\mathcal{W}_z||)$ *time;*
- *or* $\mathcal{O}(|E||V| \log^2 |V| + z(|E||V| + |V|^2 \log |V|) + ||\mathcal{W}_z||)$ *time.*

Proof Computing a min-cost flow takes $\mathcal{O}(|E||V| \log^2 |V|)$ time by Lemma 10 because the edge costs are at most 1 on each edge. From there on, finding z flows with minimal cost (or all of them if there are less than z) takes $\mathcal{O}(z|V|^3)$ time or $\mathcal{O}(z(|E||V| + |V|^2 \log |V|))$ time by Lemma 11. Listing z Eulerian trails takes $\mathcal{O}(||\mathcal{W}_z||)$ total time by applying the algorithm of [13]. □

Lemmas 13 and 14 imply Theorem 1.

5.2 Listing z Shortest Equivalent Strings

Any instance of the z-SHORTEST $\mathcal{S}_k$-EQUIVALENT STRINGS problem can be reduced to some instance of the z-SHORTEST EULERIAN WALKS problem in linear time. In particular, this reduction consists in constructing the dBG of order k of $\mathcal{S}_k$ in $\mathcal{O}(nk)$ time [50]. The resultant dBG is the directed graph $G = (V, E)$ given as input to z-SHORTEST EULERIAN WALKS. By Observation 4, the total number of nodes in V is $\mathcal{O}(n)$ and the total number of edges in E is also $\mathcal{O}(n)$. Any Eulerian walk W in G corresponds to a string of length $k - 1 + |W|$: $k - 1$ is the length of the first node of the walk to which we concatenate one letter for each of the $|W|$ edges of the walk. Thus by employing Theorems 1 and 3 we obtain Theorem 4.

6 A Reverse-safe Text Index for Existential Queries

In this section, we provide a direct technical application of our algorithms in data privacy, following the reverse-safe data structure framework [10]. Before getting into the technical details, let us provide some context for non-experts. The reverse-safe data structure framework is inspired by the *z-anonymity* privacy property [51]. This property attempts to address the following problem: Given individual-specific (field-structured) data, produce a release of the data to guarantee that the individuals who are the subjects of the data cannot be re-identified, while the data remain practically useful [51]. Then, a release of data is said to have the z-anonymity property if the information for each individual contained in the release cannot be distinguished from that of at least $z - 1$ other individuals whose information also appears in the release.

The two standard operations for producing the release are: suppression; and generalization [52]. In *suppression*, certain values of the attributes are replaced by a special symbol $\star$, which represents any value in that attribute. In *generalization*, individual values of attributes are replaced with a broader category (for categorical attributes) or interval (for a numerical attribute). For instance, the value 18 in the numerical attribute `Age` can be replaced by the interval "[10, 30]". To reduce the information loss caused by suppression or generalization, z-anonymity algorithms aim to find the most specific categories and/or the shortest intervals that are enough for z-anonymity to be satisfied.

The authors of [10] injected the z-anonymity property into the design of data structures. A data structure $D(\cdot)$ is called *z-reverse-safe* if and only if the answers it stores correspond to at least z distinct inputs [10]. Intuitively, this is to ensure that the correct input cannot be distinguished, based on the answers, from at least $z - 1$ other inputs. This, for example, is useful in the context of genomic sequences corresponding to different individuals. Let us consider a text indexing data structure as an example. Given a *private* string T of length n over an alphabet Σ, we would like to construct $D(T)$, a z-reverse-safe data structure (z-RSDS), for some integer $z > 1$, answering existential and counting pattern matching queries: *Does string P occur in T? How many times does string P occur in T?* At the same time, we would like to prevent one from reconstructing T by asking such queries to the data structure. Furthermore, we would ideally like $D(T)$ to have the following properties:

1. *Small size.* The size of $D(T)$ is asymptotically no larger than $|T|$.

2. *Data utility.* $D(T)$ stores the *maximum possible* amount of answers.
3. *Fast queries.* $D(T)$ answers queries in (near-)optimal time.

The authors of [10] presented an algorithm that constructs a z-RSDS for answering counting pattern matching queries: *How many times does string P occur in T?* Their algorithm runs in $\mathcal{O}(n^{\omega}\log k)$ time, where ω is the optimal exponent of matrix multiplication; it constructs a z-RSDS of size $\mathcal{O}(n)$, answering counting pattern matching queries of length $m \leq k$, for maximal k, in the optimal $\mathcal{O}(m)$ time. The high-level idea is as follows. For a privacy threshold $z > 0$, we compute (via exponential search) the largest k for which there exist *at least z distinct Eulerian trails* in the order-k dBG of T, and publish a string T' obtained via a random Eulerian trail in this dBG. This is correct because the strings corresponding to these trails form an *equivalence class*: every string in this equivalence class possesses exactly the same answers for counting pattern matching queries of length at most k. The authors left open [10, Final Remarks] the same question for existential pattern matching queries, which indeed seems more challenging. Intuitively, this is because strings of *very different length* can have the same answers for *existential* pattern matching queries of length at most k.

In response, we formalize the following auxiliary problem to capture the above-mentioned requirements for a z-RSDS answering existential pattern matching queries.

REVERSE- SAFE EXISTENTIAL QUERIES
Input: A string T of length n and an integer $z > 1$.
Output: The largest k such that there exist at least z distinct strings $T_1, \ldots, T_z$ with $\mathsf{Substr}_k(T_i) = \mathsf{Substr}_k(T)$, for all $i \in [1, z]$, and a shortest such witness string X; or report FAIL if this is not possible.

The above formalization is correct for existential pattern matching queries of length at most k because of the following simple yet crucial observation.

Observation 7 For any two strings S and T and an integer $k > 0$ with $\mathsf{Substr}_k(S) = \mathsf{Substr}_k(T)$, we have $\mathsf{Substr}_d(S) = \mathsf{Substr}_d(T)$, for all $d \in [1, k]$.

Let us denote by $\mathcal{S}_k(T)$ the set of strings having the same set of length-k substrings as string T, for each integer $k \leq n$. The following lemma holds.

Lemma 15 *For any string T of length n and for each $k < n$, it holds $\mathcal{S}_{(k+1)}(T) \subseteq \mathcal{S}_k(T)$. The mapping $k \mapsto |\mathcal{S}_k(T)|$ is then decreasing.*

Proof Let T be a string, $k < n$ an integer and $T_{k+1} \in \mathcal{S}_{(k+1)}(T)$. We state that $T_{k+1} \in \mathcal{S}_k(T)$, in other terms that the set of the length-k substrings in T_{k+1} is the set of the length-k substrings of T. Indeed, every length-k substring is the suffix or the prefix of a length-$(k+1)$ substring, and every length-k suffix or prefix of a length-$(k+1)$ substring is a length-k substring. Then, the set of the length-k substrings in a string is determined by its length-$(k+1)$ substrings and two strings having the same set of length-$(k+1)$ substrings have the same set of length-k substrings. □

Due to monotonicity on k (Lemma 15), we reduce the above problem to a logarithmic number of the following instances, similar to the solution in [10].

Theorem 8 *The* REVERSE- SAFE EXISTENTIAL QUERIES *problem can be solved in* $\mathcal{O}(zn^3 \log k)$ *time.*

Proof We solve $\mathcal{O}(\log k)$ instances of the z- SHORTEST $\mathcal{S}_k$- EQUIVALENT STRINGS problem (by Lemma 15); instead of listing strings (Line 13 in Algorithm 1), we count them using the BEST theorem [11] in $\mathcal{O}(n^3)$ time per extended G [10]. Note that we do not explicitly construct the multigraph, as its weighted version suffices to apply the BEST theorem. Finding a shortest witness walk W can be done in $\mathcal{O}(|W|)$ time [53, 54, 1B] by finding an Eulerian trail in a smallest extended G constructed by Algorithm 1. Since $|W| \leq n$, we can output a shortest witness string X, $|X| \leq n$, in $\mathcal{O}(n^3)$ time. □

Indeed, by constructing the suffix tree of string X output by REVERSE- SAFE EXISTENTIAL QUERIES and conceptually truncating it at string-depth k, we obtain $D(T)$. The suffix tree of X occupies $\mathcal{O}(|X|)$ space and it can be constructed in $\mathcal{O}(|X| \log |X|)$ time [55]. For any pattern P of length $m \leq k$, it can answer whether P occurs in T or not in the optimal $\mathcal{O}(m)$ time. Since $|X| \leq n$, we obtain the following result. Recall that the correctness of the constructed data structure follows from Observation 7.

Theorem 9 *Given any* $T \in \Sigma^n$ *and any integer* $z > 1$*, we can construct in* $\mathcal{O}(zn^3 \log k)$ *time an* $\mathcal{O}(n)$*-sized* z*-RSDS over* T *answering existential pattern matching queries in the optimal* $\mathcal{O}(m)$ *time for any pattern of length* $m \leq k$ *for maximal* k*, or report* FAIL *if this is not possible.*

Let us now comment on the quality of the constructed z-RSDS.

1 *Small size.* The size of $D(T)$ is asymptotically no larger than $|T|$, and, in particular, it can be much smaller due to the fact that string X is a shortest witness.
2 *Data utility.* Since k is maximal by construction, $D(T)$ stores the maximum possible number of answers.
3 *Fast queries.* $D(T)$ answers existential queries in optimal time.

7 Final Remarks

The z-SHORTEST $\mathcal{S}_k$-EQUIVALENT STRINGS problem seems quite a natural encoding problem for strings. Beyond data privacy (see Section 6), it may inspire other applications. In what follows, we give a high-level idea of how it may prove useful.

Data Compression Let $\mathcal{S}_k$ be a set of length-k strings, which we want to compact in a single longer string, preserving their substrings of length $1, \ldots, k-1$.[7] We can construct the dBG of order k of $\mathcal{S}_k$. If an Eulerian walk exists in this graph, it induces a string Z that compactly stores all and only the strings in $\mathcal{S}_k$ and their substrings of length $1, \ldots, k-1$. We can apply our algorithm underlying Theorem 4 with $z = 1$. In some application domains [15], compressing Z may require less space than compressing the set of strings $\mathcal{S}_k$ as the total number of letters in Z is significantly smaller than those in $\mathcal{S}_k$.

[7] Note that this problem differs from the shortest common superstring problem, as the latter problem may output a string containing substrings that do not appear in the input strings.

Bioinformatics Let $\mathcal{S}_k$ be a set of length-k strings obtained as the set of length-k substrings, for some integer k, occurring in a collection of reads generated by a sequencing experiment from a genome. Any Eulerian walk in the order-k dBG of $\mathcal{S}_k$ corresponds to a single genome reconstruction [16]. Our Algorithm 1 underlying Theorem 1 can be trivially adapted to list or count only the Eulerian walks whose length is *at most* ℓ, for any integer $\ell > 0$, without increasing its time complexity. This adapted algorithm can thus be applied to study the assembly complexity of specific genomes using reads collections [9].

Author Contributions All authors contributed equally to this work.

Funding GB was partially supported by the INdAM - GNCS Project CUP E53C24001950001; AC and RG were partially supported by MUR PRIN Project n. 2022TS4Y3N - EXPAND: scalable algorithms for EXPloratory Analyses of heterogeneous and dynamic Networked Data; RG was partially supported by NextGeneration EU programme PNRR ECS00000017 Tuscany Health Ecosystem Spoke 6 (CUP B63C2200068007 and I53C22000780001); EG was supported by MIUR project PRIN 2022 APML – 20229BCXNW25 and by the ALPACA project, that have received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956229; SPP was supported in part by the PANGAIA and ALPACA projects that have received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No 872539 and 956229, respectively; GP was supported by the Italian Ministry of Research, under the complementary actions to the NRRP "Fit4MedRob - Fit for Medical Robotics" Grant (# PNC0000007).

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on Strings. Cambridge University Press, Cambridge (2007)
2. Alanko, J.N., Biagi, E., Puglisi, S.J.: Longest common prefix arrays for succinct k-spectra. In: Nardini, F.M., Pisanti, N., Venturini, R. (eds.) String Processing and Information Retrieval, pp. 1–13. Springer, Cham (2023)
3. Edmonds, J.R., Johnson, E.L.: Matching, Euler tours and the Chinese Postman. Math. Program. **5**(1), 88–124 (1973). https://doi.org/10.1007/BF01580113
4. Orlin, J.B.: A polynomial time primal network simplex algorithm for minimum cost flows. Math. Program. **77**, 109–129 (1997). https://doi.org/10.1007/BF02614365

5. Tarjan, R.E.: Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. Math. Program. **77**, 169–177 (1997). https://doi.org/10.1007/BF02614369
6. Chen, L., Kyng, R., Liu, Y.P., Peng, R., Gutenberg, M.P., Sachdeva, S.: Almost-linear-time algorithms for maximum flow and minimum-cost flow. Commun. ACM **66**(12), 85–92 (2023). https://doi.org/10.1145/3610940
7. Hutchinson, J.P.: On words with prescribed overlapping subsequences. Util. Math. **7**, 241–250 (1975)
8. Hutchinson, J.P., Wilf, H.S.: On Eulerian circuits and words with prescribed adjacency patterns. J. Comb. Theory Ser. A **18**(1), 80–87 (1975). https://doi.org/10.1016/0097-3165(75)90068-0
9. Kingsford, C., Schatz, M.C., Pop, M.: Assembly complexity of prokaryotic genomes using short reads. BMC Bioinform. **11**, 21 (2010). https://doi.org/10.1186/1471-2105-11-21
10. Bernardini, G., Chen, H., Fici, G., Loukides, G., Pissis, S.P.: Reverse-safe text indexing. ACM J. Exp. Algorithmics **26** (2021). https://doi.org/10.1145/3461698
11. Aardenne-Ehrenfest, T., Bruijn, N.G.: Circuits and trees in oriented linear graphs. In: Gessel, I., Rota, G.-C. (eds.) Classic Papers in Combinatorics, pp. 149–163. Birkhäuser Boston, Boston, MA (1987). https://doi.org/10.1007/978-0-8176-4842-8_12
12. Karhumäki, J., Puzynina, S., Rao, M., Whiteland, M.A.: On cardinalities of k-abelian equivalence classes. Theor. Comput. Sci. **658**, 190–204 (2017). https://doi.org/10.1016/j.tcs.2016.06.010
13. Conte, A., Grossi, R., Loukides, G., Pisanti, N., Pissis, S.P., Punzi, G.: Beyond the best theorem: Fast assessment of eulerian trails. In: Bampis, E., Pagourtzis, A. (eds.) Fundamentals of Computation Theory, pp. 162–175. Springer, Cham (2021)
14. Bernardini, G., Chen, H., Fici, G., Loukides, G., Pissis, S.P.: Reverse-safe data structures for text indexing. In: Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX, pp. 199–213 (2020). https://doi.org/10.1137/1.9781611976007.16
15. Orlandi, A., Venturini, R.: Space-efficient substring occurrence estimation. Algorithmica **74**(1), 65–90 (2016). https://doi.org/10.1007/s00453-014-9936-y
16. Pevzner, P.A., Tang, H., Waterman, M.S.: An Eulerian path approach to DNA fragment assembly. Proc. Natl. Acad. Sci. **98**(17), 9748–9753 (2001). https://doi.org/10.1073/pnas.171285098
17. Bernardini, G., Marchetti-Spaccamela, A., Pissis, S.P., Stougie, L., Sweering, M.: Constructing strings avoiding forbidden substrings. In: 32nd Annual Symposium on Combinatorial Pattern Matching (CPM). LIPIcs, vol. 191, pp. 9–1918. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl (2021). https://doi.org/10.4230/LIPICS.CPM.2021.9
18. Bernardini, G., Conte, A., Gabory, E., Grossi, R., Loukides, G., Pissis, S.P., Punzi, G., Sweering, M.: On strings having the same length- k substrings. In: 33rd Annual Symposium on Combinatorial Pattern Matching (CPM). LIPIcs, vol. 223, pp. 16–11617. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl (2022). https://doi.org/10.4230/LIPICS.CPM.2022.16
19. Břinda, K., Baym, M., Kucherov, G.: Simplitigs as an efficient and scalable representation of de Bruijn graphs. Genome Biol. **22**, 1–24 (2021). https://doi.org/10.1186/s13059-021-02297-z
20. Schmidt, S.S., Alanko, J.N.: Eulertigs: Minimum plain text representation of k-mer sets without repetitions in linear time. Algorithms Mol. Biol. **18**(1), 5 (2023). https://doi.org/10.1186/S13015-023-00227-1
21. Rahman, A., Medevedev, P.: Representation of k-mer sets using spectrum-preserving string sets. J. Comput. Biol. **28**(4), 381–394 (2021). https://doi.org/10.1089/CMB.2020.0431
22. Schmidt, S., Khan, S., Alanko, J.N., Pibiri, G.E., Tomescu, A.I.: Matchtigs: Minimum plain text representation of k-mer sets. Genome Biol. **24**(1), 136 (2023). https://doi.org/10.1186/s13059-023-02968-z
23. Sladkỳ, O., Veselỳ, P., Břinda, K.: Masked superstrings as a unified framework for textual k-mer set representations. bioRxiv, 2023–02 (2023)
24. Bernardini, G., Chen, H., Gørtz, I.L., Krogh, C., Loukides, G., Pissis, S.P., Stougie, L., Sweering, M.: Connecting de bruijn graphs. In: 35th Annual Symposium on Combinatorial Pattern Matching (CPM). LIPIcs, vol. 296, pp. 6–1616. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl (2024). https://doi.org/10.4230/LIPICS.CPM.2024.6
25. Bernardini, G., Chen, H., Loukides, G., Pissis, S.P., Stougie, L., Sweering, M.: Making de Bruijn graphs Eulerian. In: 33rd Annual Symposium on Combinatorial Pattern Matching (CPM). LIPIcs, vol. 223, pp. 12–11218. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl (2022). https://doi.org/10.4230/LIPIcs.CPM.2022.12
26. Rossignolo, E., Comin, M.: Enhanced compression of k-mer sets with counters via de bruijn graphs. J. Comput. Biol. **31**(6), 524–538 (2024). https://doi.org/10.1089/CMB.2024.0530

27. Bernardini, G., Liu, C., Loukides, G., Marchetti-Spaccamela, A., Pissis, S.P., Stougie, L., Sweering, M.: Missing value replacement in strings and applications. Data Min. Knowl. Discov. **39**(2), 12 (2025). https://doi.org/10.1007/S10618-024-01074-3
28. Simon, I.: Piecewise testable events. In: Automata Theory and Formal Languages, 2nd GI Conference. Lecture Notes in Computer Science, vol. 33, pp. 214–222. Springer, Berlin, Heidelberg (1975). https://doi.org/10.1007/3-540-07407-4_23
29. Lothaire, M.: Combinatorics on Words, 2nd edn. Cambridge Mathematical Library. Cambridge University Press, Cambridge (1997). https://doi.org/10.1017/CBO9780511566097
30. Karandikar, P., Schnoebelen, P.: The height of piecewise-testable languages with applications in logical complexity. In: 25th EACSL Annual Conference on Computer Science Logic. LIPIcs, vol. 62, pp. 37–13722. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl (2016). https://doi.org/10.4230/LIPIcs.CSL.2016.37
31. Karandikar, P., Kufleitner, M., Schnoebelen, P.: On the index of Simon's congruence for piecewise testability. Inf. Process. Lett. **115**(4), 515–519 (2015). https://doi.org/10.1016/j.ipl.2014.11.008
32. Karandikar, P., Schnoebelen, P.: The height of piecewise-testable languages and the complexity of the logic of subwords. Log. Methods Comput. Sci. **15**(2) (2019). https://doi.org/10.23638/LMCS-15(2:6)2019
33. Barker, L., Fleischmann, P., Harwardt, K., Manea, F., Nowotka, D.: Scattered factor-universality of words. In: Developments in Language Theory - 24th International Conference. Lecture Notes in Computer Science, vol. 12086, pp. 14–28. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-48516-0_2
34. Kim, S., Han, Y., Ko, S., Salomaa, K.: On the Simon's congruence neighborhood of languages. In: 27th International Conference on Developments in Language Theory (DLT). Lecture Notes in Computer Science, vol. 13911, pp. 168–181. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-33264-7_14
35. Hébrard, J.: An algorithm for distinguishing efficiently bit-strings by their subsequences. Theor. Comput. Sci. **82**(1), 35–49 (1991). https://doi.org/10.1016/0304-3975(91)90170-7
36. Garel, E.: Minimal separators of two words. In: 4th Annual Symposium on Combinatorial Pattern Matching (CPM). Lecture Notes in Computer Science, vol. 684, pp. 35–53. Springer, Berlin, Heidelberg (1993). https://doi.org/10.1007/BFb0029795
37. Tronícek, Z.: Common subsequence automaton. In: Implementation and Application of Automata, 7th International Conference. Lecture Notes in Computer Science, vol. 2608, pp. 270–275. Springer, Berlin, Heidelberg (2002). https://doi.org/10.1007/3-540-44977-9_28
38. Crochemore, M., Melichar, B., Troníček, Z.: Directed acyclic subsequence graph - overview. J. Discrete Algorithms **1**(3–4), 255–280 (2003). https://doi.org/10.1016/S1570-8667(03)00029-7
39. Fleischer, L., Kufleitner, M.: Testing Simon's congruence. In: 43rd International Symposium on Mathematical Foundations of Computer Science. LIPIcs, vol. 117, pp. 62–16213. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl (2018). https://doi.org/10.4230/LIPIcs.MFCS.2018.62
40. Gawrychowski, P., Kosche, M., Koß, T., Manea, F., Siemer, S.: Efficiently testing Simon's congruence. In: 38th International Symposium on Theoretical Aspects of Computer Science (STAC). LIPIcs, vol. 187, pp. 34–13418. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl (2021). https://doi.org/10.4230/LIPIcs.STACS.2021.34
41. Kim, S., Ko, S., Han, Y.: Simon's congruence pattern matching. In: 33rd International Symposium on Algorithms and Computation (ISAAC). LIPIcs, vol. 248, pp. 60–16017. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl (2022). https://doi.org/10.4230/LIPICS.ISAAC.2022.60
42. Fleischmann, P., Kim, S., Koß, T., Manea, F., Nowotka, D., Siemer, S., Wiedenhöft, M.: Matching patterns with variables under Simon's congruence. In: Reachability Problems - 17th International Conference (RP). Lecture Notes in Computer Science, vol. 14235, pp. 155–170. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-45286-4_12
43. Kim, S., Ko, S., Han, Y.: Simon's congruence pattern matching. Theor. Comput. Sci. **994**, 114478 (2024). https://doi.org/10.1016/J.TCS.2024.114478
44. Euler, L.: Solutio problematis ad geometriam situs pertinentis. Euler Archive - All Works **53**, 15 (1741)
45. Goldberg, A.V., Tarjan, R.E.: Solving minimum-cost flow problems by successive approximation. In: Aho, A.V. (ed.) Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC), pp. 7–18. ACM, New York (1987). https://doi.org/10.1145/28395.28397
46. Orlin, J.B.: A faster strongly polynomial minimum cost flow algorithm. Oper. Res. **41**(2), 338–350 (1993). https://doi.org/10.1287/opre.41.2.338

47. Könen, D., Schmidt, D.R., Spisla, C.: Finding all minimum cost flows and a faster algorithm for the K best flow problem. Discret. Appl. Math. **321**, 333–349 (2022). https://doi.org/10.1016/J.DAM.2022.07.007
48. Hamacher, H.W.: A note on K best network flows. Ann. Oper. Res. **57**(1), 65–72 (1995). https://doi.org/10.1007/BF02099691
49. Sedeño-Noda, A., Espino-Martín, J.J.: On the K best integer network flows. Comput. Oper. Res. **40**(2), 616–626 (2013). https://doi.org/10.1016/j.cor.2012.08.014
50. Cazaux, B., Lecroq, T., Rivals, E.: Linking indexing data structures to de Bruijn graphs: Construction and update. J. Comput. Syst. Sci. **104**, 165–183 (2019). https://doi.org/10.1016/j.jcss.2016.06.008
51. Sweeney, L.: k-Anonymity: A model for protecting privacy. Int. J. Uncertain. Fuzziness Knowl. Based Syst. **10**(5), 557–570 (2002). https://doi.org/10.1142/S0218488502001648
52. Sweeney, L.: Achieving k-Anonymity privacy protection using generalization and suppression. Int. J. Uncertain. Fuzziness Knowl. Based Syst. **10**(5), 571–588 (2002). https://doi.org/10.1142/S021848850200165X
53. Hierholzer, C., Wiener, C.: Über die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. Math. Ann. **6**(1), 30–32 (1873)
54. Biggs, N.L., Lloyd, E.K., Wilson, R.J.: Graph Theory 1736-1936. Clarendon Press, Oxford (1976). Chap. 1B
55. Weiner, P.: Linear pattern matching algorithms. In: 14th Annual Symposium on Switching and Automata Theory, pp. 1–11. IEEE Computer Society, Iowa (1973). https://doi.org/10.1109/SWAT.1973.13

Authors and Affiliations

Giulia Bernardini[1] · Alessio Conte[2] · Esteban Gabory[3] · Roberto Grossi[2] · Grigorios Loukides[4] · Solon P. Pissis[5] · Giulia Punzi[2] · Michelle Sweering[5]

✉ Solon P. Pissis
solon.pissis@cwi.nl

Giulia Bernardini
giulia.bernardini@unimi.it

Alessio Conte
alessio.conte@unipi.it

Esteban Gabory
esteban.gabory@unipa.it

Roberto Grossi
roberto.grossi@unipi.it

Grigorios Loukides
grigorios.loukides@kcl.ac.uk

Giulia Punzi
giulia.punzi@unipi.it

Michelle Sweering
michelle.sweering@gmail.com

1 University of Milan, Milan, Italy

2 University of Pisa, Pisa, Italy

3 University of Palermo, Palermo, Italy

4 King's College London, London, UK

5 CWI, Amsterdam, The Netherlands