When Is String Reconstruction Using de Bruijn Graphs Hard?

Ben Bals
CWI, Amsterdam, The Netherlands
Vrije Universiteit Amsterdam, The Netherlands

Sebastiaan van Krieken
CWI, Amsterdam, The Netherlands
TU Delft, The Netherlands

Solon P. Pissis
CWI, Amsterdam, The Netherlands
Vrije Universiteit Amsterdam, The Netherlands

Leen Stougie
CWI, Amsterdam, The Netherlands
Vrije Universiteit Amsterdam, The Netherlands

Hilde Verbeek
CWI, Amsterdam, The Netherlands

Abstract

The reduction of the fragment assembly problem to (variations of) the classical Eulerian trail problem [Pevzner et al., PNAS 2001] has led to remarkable progress in genome assembly. This reduction employs the notion of *de Bruijn graph* $G = (V, E)$ of order k over an alphabet Σ. A single Eulerian trail in G represents a *candidate* genome reconstruction. Bernardini et al. have also introduced the complementary idea in data privacy [ALENEX 2020] based on z-anonymity. Let S be a private string that we would like to release, preventing, however, its full reconstruction. For a privacy threshold $z > 0$, we compute the largest k for which there exist at least z Eulerian trails in the order-k de Bruijn graph of S, and release a string S' obtained via a random Eulerian trail.

The pressing question is: How hard is it to reconstruct a best string from a de Bruijn graph given a function that models domain knowledge? Such a function maps every length-k string to an *interval of positions* where it may occur in the reconstructed string. By the above reduction to de Bruijn graphs, the latter function translates into a function c mapping every edge to an interval where it may occur in an Eulerian trail. This gives rise to the following basic problem on graphs:

Given an instance (G, c), can we efficiently compute an Eulerian trail respecting c?

Hannenhalli et al. [CABIOS 1996] formalized this problem and showed that it is NP-complete. Ben-Dor et al. [J. Comput. Biol. 2002] showed that it is NP-complete, even on de Bruin graphs with $|\Sigma| = 4$. In this work, we settle the lower-bound side of this problem by showing that finding a c-respecting Eulerian trail in de Bruijn graphs over alphabets of size 2 is NP-complete.

We then shift our focus to *parametrization* aiming to capture the quality of our domain knowledge in the complexity. Ben-Dor et al. developed an algorithm to solve the problem on de Bruijn graphs in $\mathcal{O}(m \cdot w^{1.5} 4^w)$ time, where $m = |E|$ and w is the *maximum interval length* over all edges in E. Bumpus and Meeks [Algorithmica 2023] later rediscovered the same algorithm on temporal graphs, which highlights the relevance of this problem in other contexts. Our central contribution is showing how combinatorial insights lead to *exponential-time* improvements over the state-of-the-art algorithm. In particular, for the important class of de Bruijn graphs, we develop an algorithm parametrized by $w(\log w + 1)/(k-1)$: for a de Bruijn graph of order k, it runs in $\mathcal{O}(mw \cdot 2^{\frac{w(\log w+1)}{k-1}})$ time. Our result improves on the state of the art by roughly an exponent of $(\log w + 1)/(k - 1)$. The existing algorithms have a natural interpretation for string reconstruction: when for each length-k string, we know a small range of positions it must lie in, string reconstruction can be solved in linear time. Our improved algorithm shows that *it is enough when the range of positions is small relative to k.*

We then generalize both the existing and our novel FPT algorithm by allowing the cost at every position of an interval to vary. In this optimization version, our hardness result translates into inapproximability and the FPT algorithms work with a slight extension. Surprisingly, even in this more general setting, we extend the FPT algorithms to count and enumerate the min-cost Eulerian trails. The counting result has direct applications in the data privacy framework of Bernardini et al.

2012 ACM Subject Classification Theory of computation → Pattern matching; Theory of computation → Graph algorithms analysis

Keywords and phrases string algorithm, graph algorithm, de Bruijn graph, Eulerian trail

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.53

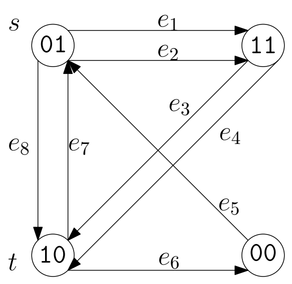
Related Version *Full Version*: `https://arxiv.org/abs/2508.03433`

Funding *Solon P. Pissis*: 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.
Leen Stougie: Supported in part by the Netherlands Organisation for Scientific Research (NWO) through project OCENW.GROOT.2019.015 "Optimization for and with Machine Learning (OPTIMAL)" and Gravitation-project NETWORKS-024.002.003.
Hilde Verbeek: Supported by a Constance van Eeden Fellowship.

1 Introduction

One of the most important algorithmic tasks in bioinformatics is that of *genome assembly* (or *fragment assembly*) [30, 8, 23, 9]: the process of taking a large number of short DNA fragments and putting them back together to create a representation of the original chromosomes from which the DNA originated. The textbook reduction of the fragment assembly problem to (variations of) the classical Eulerian trail problem [26] has led to remarkable progress in the past three decades. This reduction is based on the notion of *de Bruijn graph* (dBG, in short).

Let $S = S[1] \dots S[|S|] = S[1 \,.\,.\, |S|]$ be a *string* of length $|S|$ over an *alphabet* Σ. We fix a collection $\mathcal{S}$ of strings over Σ and define the *order-k de Bruijn graph* of $\mathcal{S}$ as a directed graph, denoted by $G_{\mathcal{S},k} = (V, E)$, where V is the set of length-$(k-1)$ substrings of the strings in $\mathcal{S}$ and E has an edge (u, v) if and only if $u[1] \cdot v = u \cdot v[k-1]$ and $u[1] \cdot v$ occurs in some string S of $\mathcal{S}$. In applications, we often consider a de Bruijn *multigraph* where the multiplicity of an edge is exactly the total number of these occurrences in the strings in $\mathcal{S}$. Then, a single Eulerian trail in $G_{\mathcal{S},k}$ represents a *candidate* string reconstruction [26]; see Figure 1.



Eulerian trails	String reconstructions
$e_1e_3e_6e_5e_2e_4e_7e_8$	0110011010
$e_1e_3e_7e_2e_4e_6e_5e_8$	0110110010
$e_1e_3e_7e_8e_6e_5e_2e_4$	0110100110
$e_8e_6e_5e_1e_3e_7e_2e_4$	0100110110
$e_8e_7e_1e_3e_6e_5e_2e_4$	0101100110
$e_1e_3e_6e_5e_8e_7e_2e_4$	0110010110

Figure 1 The de Bruijn multigraph $G_{\mathcal{S},k} = (V, E)$ (left), the set of node-distinct Eulerian trails from s to t (middle), and the corresponding set of string reconstructions (right) for the string collection $\mathcal{S} = 001, 010, 011, 011, 100, 101, 110, 110$, over the alphabet $\Sigma = \{0, 1\}$, and $k = 3$.

Bernardini et al. have introduced the complementary idea in data privacy [4] based on the *z-anonymity* privacy property [27, 29]. Let S be a *private* string that we would like to release for data analysis, preventing, however, its full reconstruction. For a privacy threshold $z > 0$, we compute the largest k for which there exist at least z node-distinct Eulerian trails in the order-k dBG of $\mathcal{S} = \{S\}$, and release a string S' obtained via a random Eulerian trail. In Figure 1, we have 6 node-distinct Eulerian trails (and thus 6 distinct strings) corresponding to $\mathcal{S}$. Under the z-anonymity assumption, one cannot know which of the 6 strings is S unless they can rely on some additional information about S, such as on *domain knowledge*.

The pressing question arising from these applications is thus: How hard is it to reconstruct a best string from $G_{\mathcal{S},k}$ given a function modeling domain knowledge? This function maps every length-k string to an *interval of positions* where it may occur in the reconstructed string. By the above reduction, this translates into a function c mapping every edge to an interval where it may occur in an Eulerian trail, raising the following basic graph problem:

Given an instance (G, c), can we efficiently compute an Eulerian trail respecting c?

Although Hannenhalli et al. [16] formalized this basic problem in the context of dBGs, it has applications in temporal graphs [7, 22, 24] and other networks [11, 31, 20]. We denote it here by diET (see Section 2 for a formal definition). Hannenhalli et al. showed that diET is NP-complete when each node of G has in- and out-degree at most two. Even if their motivating application was fragment assembly and thus based on dBGs, their result was shown on general directed graphs. Ben-Dor et al. [3] then showed that diET is NP-complete, even on dBGs with $|\Sigma| = 4$. We settle the negative landscape by showing the following result:

▶ **Theorem 3 (*).** *The* diET *problem is* NP*-complete, even on de Bruijn graphs with* $|\Sigma| = 2$.

Beyond theoretically interesting, Theorem 3 shows that *in general* it is indeed hard to reconstruct a private (binary) string from a given dBG, thus providing theoretical justification for the privacy framework introduced by Bernardini et al. [4]. Given these negative results, we shift our focus to *parametrization* aiming to capture the quality of our domain knowledge in the complexity. This leads to algorithms that are efficient if the intervals are small [16]: every interval length is bounded by a natural number w, which we term the *interval width*. We denote this parametrized version of diET by diET$[w]$. Hannenhalli et al. [16] showed an algorithm for diET$[w]$ working in $\mathcal{O}(m^{2w+\log(2w)})$ time for any directed graph $G = (V, E)$, with $m = |E|$ and $w = \mathcal{O}(1)$. Ben-Dor et al. [3] developed an algorithm to solve the problem on dBGs in $\mathcal{O}(m \cdot w^{1.5} 4^w)$ time, for any w. Bumpus and Meeks [7] later rediscovered the same algorithm on temporal graphs, which highlights the relevance of the diET problem in other contexts. We may thus summarize the state of the art in the following statement.

▶ **Theorem 10** (Theorem 11 of [3], Theorem 7 of [7])**.** *There is an $\mathcal{O}(m \cdot w^{1.5} 4^w)$-time algorithm solving the* diET$[w]$ *problem. Therefore,* diET$[w]$ *is in* FPT.

We observe that the state-of-the-art algorithm does not exploit the dBG structure. Our central contribution is developing an algorithm for dBGs parametrized by $w(\log w+1)/(k-1)$:

▶ **Theorem 11.** *Let G be a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$. There is an $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$-time algorithm solving the* diET$[w]$ *problem, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

Theorem 11 improves on the state of the art by roughly an exponent of $(\log w+1)/(k-1)$. The existing algorithms [16, 3, 7] all have a natural interpretation for string reconstruction: when for each length-k substring (*k-mer*), we know a small range of positions it must lie in, string reconstruction can be solved in linear time. Theorem 11 shows that *it is enough when* *the range of positions is small relative to the order k* of the dBG. In particular, we show that for dBGs it is sufficient if $w \log w/(k-1)$ is relatively small, which significantly extends the practical applicability of our technique. For instance, in bioinformatics, it is standard to use $k = 31$ [21] and then we have $|\Sigma| = 4$ (the size of the DNA alphabet), which implies an exponential speedup by $\sqrt[30]{\cdot}$. Our approach of improving the FPT algorithms for diET based on combinatorial insights into the structure of the instances suggests further research into closely-related problems; e.g., the *Hierarchical Chinese Postman* (HCP) problem [15, 19, 11, 1] and the related *Time-Constrained Chinese Postman* (TCCP) problem [31, 28].

We then generalize the above results by allowing the cost at every position of an interval to vary. We denote this problem here by dicET (see Section 2 for a formal definition). In this setting, our hardness result (Theorem 3) translates into inapproximability.

▶ **Corollary 7 (*).** *If* P $\neq$ NP *there is no constant-factor polynomial-time approximation algorithm for the* dicET *problem, even on de Bruijn graphs with interval cost functions.*

We show that the FPT algorithms underlying both Theorems 10 and 11 also work in the optimization version with an interval cost function, which we denote by dicET$[w]$.

▶ **Corollary 20 (*).** *Given a* dicET$[w]$ *instance, there is an $\mathcal{O}(m \cdot w^{1.5}4^w)$-time algorithm finding a min-cost Eulerian trail in G. On a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$, we can solve this problem in $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ time, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

Surprisingly, even in this more general setting, we show how to extend our FPT techniques to count the number of min-cost Eulerian trails. We show the following result.

▶ **Theorem 21.** *Given a* dicET$[w]$ *instance, we can count the number of min-cost Eulerian trails in $\mathcal{O}(m \cdot w^{1.5}4^w)$ time. On a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$, we can solve this problem in $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ time, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

We can also enumerate these trails in the same time as for counting (Theorem 21) plus time that is linear in the size of the output. Notably, all of our algorithmic results translate from directed to undirected graphs with the same complexities. It is easy to verify that none of the proofs depend on the directedness of the graph. For simplicity, we focus our discussion on directed graphs. Particularly our result for counting Eulerian trails in undirected graphs is surprising given that the problem is #P-complete in the standard setting [6]. We also show that most of our algorithmic results generalize to multigraphs. Finally, we show that the undirected version of dicET, which we denote by uicET, is also NP-complete.

Overview of Hardness Techniques. We consider dBGs over alphabets of size two. Unlike [16, 3], this highly structured setting requires intricate techniques to obtain an (elementary) reduction from the directed Hamiltonian path problem [18]. For our instances, we harness the fact that the shortest path between any two nodes in a complete dBG is unique and has a meaningful string interpretation. The structure of the reduction makes it directly translatable to the optimization setting implying inapproximability. Similarly, we reduce the undirected Hamiltonian path problem [18] to finding a min-cost c-respecting Eulerian trail in an undirected graph. This reduction requires a different approach than the directed case to ensure that critical parts of the graph can only be traversed in the desired order. We assign different costs at even and odd time steps to certain edges to achieve that goal.

Overview of Algorithmic Techniques. We apply tools from parametrized algorithms [10] to solve dicET$[w]$ efficiently. Our techniques can be viewed as a careful combination of searching in a well-bounded state space and dynamic programming. We further combine approaches from graph and string algorithms to enhance the vanilla version of these tools with combinatorial insights into dBGs and obtain exponential-time improvements. The robustness of these tools allows us to generalize our algorithm for the decision version of dicET$[w]$ to both the optimization and the counting versions. The counting result in particular relies on the fact that the state space compactly captures all possible Eulerian trails, by excluding impossible trails at the construction level, and representing the possible ones efficiently.

Other Related Work. Our work is closely related to exploring *temporal graphs* [24, 13, 14, 12, 2, 7, 22]: graphs where every edge is available at an arbitrary subset of the time steps. Most relevant to our work are perhaps the works by Bumpus and Meeks [7] and by Marino and Silva [22]. The former proved that deciding whether a temporal graph has an Eulerian trail is NP-complete even if each edge appears at most r times, for every fixed $r \geq 3$. They also apply similar parametrized tools to interval-membership-width, a temporal graph parameter related to our interval width, but they do not consider directed graphs, costs, or counting. Marino and Silva [22] showed, for undirected graphs, that, if the edges of a temporal graph (G, λ) with lifetime τ are always available during the lifetime, then deciding whether (G, λ) has an Eulerian trail is NP-complete even if $\tau = 2$. They also showed that this problem is in XP when parametrized by $\tau + \mathsf{tw}(G)$, where $\mathsf{tw}(G)$ is the treewidth of G.

Our work is also closely related to the HCP problem [15, 19, 11, 1] (and the related TCCP problem [31, 28]). In HCP, we are given an edge-weighted undirected graph $G = (V, E)$, a partition $\mathcal{P}$ of E into k classes, and a partial order $\prec$ on $\mathcal{P}$, and we are asked to find a least-weight closed walk traversing each edge in E at least once such that each edge e in a class E' is traversed only after all edges in all classes $E'' \prec E'$ are traversed. The main differences to our problem are two: (1) In HCP, there is a partition on the edges and then a partial order between the classes, whereas we have an arbitrary interval per edge; and (2) in HCP, one must traverse each edge at least once, whereas we have to do this exactly once.

Paper Organization. We begin by formalizing diET and dicET in Section 2. We present our hardness and inapproximability results in Section 3. We complement these negative results with positive algorithmic results for diET$[w]$ in Section 4. In Section 5, we extend our algorithms to counting Eulerian trails and discuss the relationship to the data privacy applications. In the full version, we discuss how most of our algorithms extend to multigraphs. *The formal statements, with proofs or details deferred to the full version, are marked with* *.

2 Preliminaries

For $a, b \in \mathbb{N}$, let $[a, b] \coloneqq \{x \in \mathbb{N} \mid a \leq x \leq b\}$, $[a, b) \coloneqq \{x \in \mathbb{N} \mid a \leq x < b\}$, and $[b] \coloneqq [1, b]$. Further let $\mathcal{I}_b \coloneqq \{[i, j] \mid 1 \leq i \leq j \leq b\} \cup \{\{\}\}$ be the set of intervals on $[b]$.

We imagine a trail as "walking" through a graph and thus refer to the ranks of the edges as happening at certain time steps (e.g., the second edge is associated with the time step 2). This vocabulary borrowed from temporal graphs makes the discussion more intuitive.

▶ **Definition 1.** *Let $G = (V, E)$ be a directed or undirected graph with $m = |E|$. We call a function $c \colon E \to \mathcal{I}_m$ an* interval function *for G. We extend this notion to an* interval cost function *where, for each $e \in E$, each time step from the interval $c(e)$ is associated with a cost from $\mathbb{Z}$. As an abuse of notation, we write $c(e, t)$ for this cost. If $t \notin c(e)$, we set $c(e, t) \coloneqq \infty$.*

When the relevant graph $G = (V, E)$ is clear from context, we set $n \coloneqq |V|$ and $m \coloneqq |E|$. We next formalize the main problems in scope on directed graphs.

▶ **Problem.** Eulerian Trails in Digraphs with Interval functions (diET)

Given: Directed graph $G = (V, E)$, interval function $c\colon E \to \mathcal{I}_m$

Decide: Is there an Eulerian trail $P = e_1 \dots e_m$ in G such that for all $t \in [m]$, $t \in c(e_t)$?

Similarly, we consider the version with interval *cost* functions.

▶ **Problem.** Eulerian Trails in Digraphs with Interval Cost functions (dicET)

Given: Directed graph $G = (V, E)$, interval cost function $c\colon E \times [m] \to \mathbb{Z} \cup \{\infty\}$, $c_{\text{budget}} \in \mathbb{N}$

Decide: Is there an Eulerian trail $P = e_1 \dots e_m$ in G such that $\sum_{t=1}^{m} c(e_t, t) \le c_{\text{budget}}$?

In addition, we consider the uicET problem as the natural *undirected* version of dicET.

Note that in all cases, we allow setting every interval to the whole $[m]$. In the case without costs, this yields the classical Eulerian trail problem [17]. In the case with costs, this yields a min-cost Eulerian trail problem without any interval restrictions, which is a natural special case we will call out when considering it to be particularly relevant. See Figure 2 for an example of a graph with an interval function and the corresponding edge availabilities.

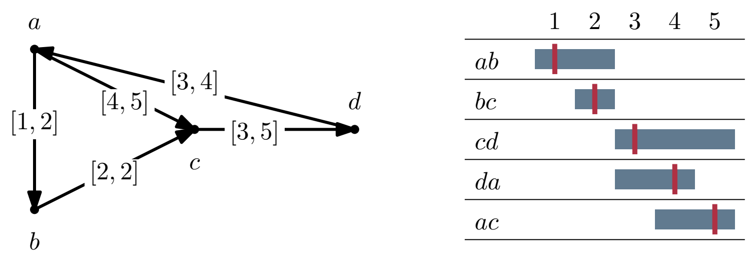


Figure 2 On the left is the input graph G: every edge is labeled with the time steps at which it is available. On the right is a table illustrating the interval every edge is available: *abcdac* is an Eulerian trail; the edge usages corresponding to this trail are indicated with red vertical lines.

The following definition makes our arguments based on an interval function more legible.

▶ **Definition 2.** *We call an edge $e \in E$* available *at $t \in [m]$, if $t \in c(e)$. For a time step $t \in [m]$, we write $E(t)$ to denote the set of edges available at t. Given an interval $T \subseteq [m]$, we similarly write $E(T)$ for the set of edges available at at least one time step from T.*

3 NP-hardness and Inapproximability

In this section, we consider both the directed and undirected case from the hardness perspective. Since in the directed case, dBGs are a particularly relevant application, we give strong results, even when restricting the instances to this graph class. We are the first to show the hardness for alphabets of all sizes (except $|\Sigma| = 1$, where the problem is trivial). In particular, we settle the question for the important case of binary alphabets. In the undirected case, we consider general graphs, which we believe is nonetheless insightful regarding what makes this problem class hard.

We first settle the computational complexity of diET.

▶ **Theorem 3** (*). *The* diET *problem is* NP*-complete, even on de Bruijn graphs with $|\Sigma| = 2$.*

We reduce from the directed Hamiltonian path problem, which is NP-complete [18]. Let $G = (V, E)$ be the directed graph in which we want to find a Hamiltonian path.

Construction of the dBG. We consider any two-letter alphabet; for example, here we will consider the subset $\Sigma := \{\mathtt{A}, \mathtt{T}\}$ of the DNA alphabet. We set $\ell := \lceil \log_2(|V|) \rceil$ and construct a diET instance $(G' = (V', E'), c)$ as the complete dBG on the set of nodes each labeled by a string of length $k - 1 = 4\ell + 10$, as well as an interval function c that we will describe later. Note that, by the choice of ℓ and Σ, G' has $|\Sigma|^{k-1} = 2^{4\ell+10} = \mathcal{O}(|V|^4)$ nodes. We will pay special attention to some of the nodes in G', which we assign an interpretation in terms of G.

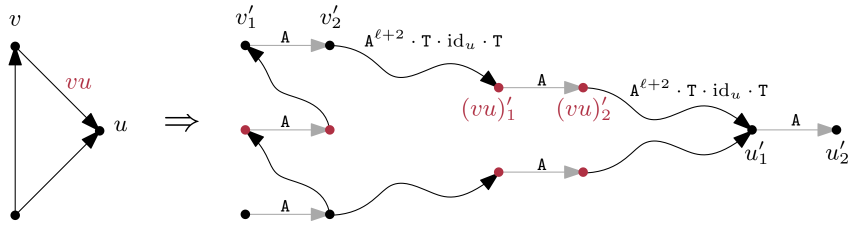
For each $v \in V$, let $\mathrm{id}_v \in \Sigma^\ell$ be a unique identifier among V. By the choice of ℓ, this numbering is always possible. Similarly, let $\mathrm{id}_{v'} \in \Sigma^{4\ell+10}$ be the string associated with a node $v' \in V'$. With each node $v \in V$, we associate two nodes v'_1 and v'_2 in V' such that:

$$\mathrm{id}_{v'_1} := \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_v \cdot \mathtt{T} \cdot \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_v \cdot \mathtt{T} \text{ and } \mathrm{id}_{v'_2} := \mathtt{A}^{\ell+2} \cdot \mathtt{T} \cdot \mathrm{id}_v \cdot \mathtt{T} \cdot \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_v \cdot \mathtt{T} \cdot \mathtt{A},$$

where $\mathtt{A}^i$ is the string of i $\mathtt{A}$'s. We use the notation $a \xrightarrow{x} b$ to denote a directed edge ab labeled x, where x is the letter used to obtain b from a in the dBG. We extend this notation to $a \overset{X}{\rightsquigarrow} b$ to denote the path from a to b using string X. Observe that by the choice of the underlying strings, we have an edge $v'_1 \xrightarrow{\mathtt{A}} v'_2$ in the dBG. We call this the *inner edge* for v. With each edge $e = vu \in E$, we associate two nodes e'_1 and e'_2 such that:

$$\mathrm{id}_{e'_1} := \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_v \cdot \mathtt{T} \cdot \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_u \cdot \mathtt{T} \text{ and } \mathrm{id}_{e'_2} := \mathtt{A}^{\ell+2} \cdot \mathtt{T} \cdot \mathrm{id}_v \cdot \mathtt{T} \cdot \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_u \cdot \mathtt{T} \cdot \mathtt{A}.$$

Again, we have an edge $e'_1 \xrightarrow{\mathtt{A}} e'_2$ in the dBG. We call this the *inner edge* for e. For every other string of length $4\ell + 10$ over Σ a node v' exists in V', but it is not associated with a node $v \in V$ or an edge $e \in E$. See Figure 3 for an illustration of this construction. A crucial property that we will later utilize is that for every pair of nodes $v, u \in V$, such that $vu \in E$, there is a unique shortest path of length $2\ell + 4$ in G' between v'_2 and $(vu)'_1$; this path is associated with the string $\mathtt{A}^{\ell+2} \cdot \mathtt{T} \cdot \mathrm{id}_u \cdot \mathtt{T}$ (see Figure 3). The analogous property holds for $(vu)'_2$ and u'_1: there is a unique shortest path of length $2\ell + 4$ in G' from $(vu)'_2$ to u'_1.



v
vu
u
$\Rightarrow$
v'_1
v'_2
A
$\mathtt{A}^{\ell+2} \cdot \mathtt{T} \cdot \mathrm{id}_u \cdot \mathtt{T}$
$(vu)'_1$
$(vu)'_2$
$\mathtt{A}^{\ell+2} \cdot \mathtt{T} \cdot \mathrm{id}_u \cdot \mathtt{T}$
u'_1
u'_2

Figure 3 A directed graph G (left) and the corresponding part of graph G' (right). The nodes in G' that correspond to nodes in G are colored black; the nodes in G' that correspond to edges in G are colored red. Inner edges in G' are colored gray and squiggly lines indicate *paths* of length $2\ell + 4$.

It is straightforward to verify that the claimed path (from v'_2 to $(vu)'_1$) exists. To see that this is the shortest possible, notice that $\mathrm{id}_{v'_2}$ contains the substring $\mathtt{A}^{\ell+3}$ starting at position $2\ell + 5$ (i.e., after the first occurrence of $\mathrm{id}_v \cdot \mathtt{T}$). Any one edge on a path to $(vu)'_1$ shifts this substring left by one letter. By the strategic placement of $\mathtt{T}$ around the id parts, the substring $\mathtt{A}^{\ell+3}$ occurs precisely at positions 1 and $2\ell + 6$ in $\mathrm{id}_{(vu)'_1}$. Thus, the closest occurrence of $\mathtt{A}^{\ell+3}$ left of position $2\ell + 6$ in $\mathrm{id}_{(vu)'_1}$ starts at position 1 (i.e., requires a left shift by at least $2\ell + 4$ letters). Therefore, any path from v'_2 to $(vu)'_1$ must contain at least $2\ell + 4$ edges. The argument is analogous for the path from $(vu)'_2$ to u'_1.

Construction of the Interval Function. Let us set $\tau := 2n - 1 + 2(n - 1)(2\ell + 4)$. We dub the time steps 1 to τ *early* and all others *late*. The intuition is that the first τ edges on any c-respecting Eulerian trail in G' will correspond to a Hamiltonian path in G. For ease of notation, we set $m' := |\Sigma|^k = 2^{4\ell+11}$ to be the number of edges in G'.

For the interval function c, we set $c(v_1'v_2') := [1, \tau]$, for all $v \in V$. For any non-edge $e \notin E$ we set $c(e_1'e_2') := [\tau + 1, m']$. All other edges in G' have the full interval $[m']$. This also holds for all inner edges for the edges in E. Note that with these choices, the inner edges for nodes $v \in V$ are only available early, and for a node pair $v, u \in V$, the edge $(vu)_1' \to (vu)_2'$ is available early only if $v \to u$ is an edge in E.

▶ **Lemma 4.** *If G has a Hamiltonian path, there is a c-respecting Eulerian trail in (G', c).*

Proof. Let $P = v_1 \dots v_n$ be a Hamiltonian path in G. Then for the Eulerian trail in G', set

$$P_1' := (v_1)_1' \xrightarrow{\mathtt{A}} (v_1)_2' \overset{\mathtt{A}^{\ell+2}\cdot\mathtt{T}\cdot\mathrm{id}_{v_2}\cdot\mathtt{T}}{\rightsquigarrow} (v_1v_2)_1' \xrightarrow{\mathtt{A}} (v_1v_2)_2' \rightsquigarrow \cdots \rightsquigarrow (v_n)_1' \xrightarrow{\mathtt{A}} (v_n)_2'.$$

Notice that P_1' traverses all inner edges for nodes and that P_1' has length exactly τ. Thus all inner edges for nodes are available by construction at their time steps on P_1'. Similarly, for any $i \in [n-1]$, we have that $v_iv_{i+1} \in E$ since P is a Hamiltonian path in G. Thus, the inner edges for edges are available on P_1' as well. Finally, all other edges on P_1' have their $\mathtt{A}^{\ell+3}$ substring in such positions that they do not fall under any of the edge classes whose interval is restricted, thus they are available for the full interval $[m']$. We conclude that P_1' respects c.

As we want P_1' to be the prefix of an Eulerian trail, we must ensure that no edge repeats in P_1'. The nodes of types v_1', v_2', e_1', and e_2' in G', for some $v \in V$ or $e \in E$, each appears at most once on P_1' by construction, and thus so must their inner edges. Therefore, if there is an edge xy in G', for two nodes x and y, that appears twice on P_1', it must be on some subpath of type $(v_i)_2' \overset{\mathtt{A}^{\ell+2}\cdot\mathtt{T}\cdot\mathrm{id}_{v_{i+i}}\cdot\mathtt{T}}{\rightsquigarrow} (v_iv_{i+1})_1'$ (or analogously of the type $(v_iv_{i+1})_2' \overset{\mathtt{A}^{\ell+2}\cdot\mathtt{T}\cdot\mathrm{id}_{v_{i+1}}\cdot\mathtt{T}}{\rightsquigarrow} (v_{i+1})_1'$) as well as at some other part of P_1'. Let us analyze the string corresponding to node x. As x appears on the path from $(v_i)_2'$ to $(v_iv_{i+1})_1'$, it must have the form $r_1 \cdot \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_{v_i} \cdot \mathtt{T} \cdot r_2$, where r_1 is a suffix of $\mathtt{A}^{\ell+2} \cdot \mathtt{T} \cdot \mathrm{id}_{v_i} \cdot \mathtt{T}$ and r_2 is a prefix of $\mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_{v_{i+1}} \cdot \mathtt{T}$. Consider that therefore the string corresponding to x contains the substring $\mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_{v_i} \cdot \mathtt{T}$ somewhere in the middle. Since, however, our id's are unique, x may only appear on the subpath corresponding to an out-edge of v_i in G and a Hamiltonian path can only include one out-edge per node. Finally, no edge can repeat within the same such subpath as this is a shortest path. The above discussion contradicts our assumption that xy repeats on P_1'. Therefore, we conclude that all edges on P_1' are unique.

We are left to show that we can complete P_1' to traverse every edge of G' exactly once. Observe that since G' is a complete dBG, the in- and out-degree of every node is $|\Sigma| = 2$. Therefore, in $G' - P_1'$ (which we define as G' without the edges in P_1') the in- and out-degree of every node is the same except for $(v_1)_1'$ (where the out-degree is one less than the in-degree) and for $(v_n)_2'$ (where the out-degree is one more than the in-degree). Also notice that $G' - P_1'$ remains weakly connected (as any strongly connected 2-regular directed graph remains weakly connected if a path is removed). By these properties, there is an Eulerian trail P_2' in $G' - P_1'$. In particular, it must start at $(v_n)_2'$ and end at $(v_1)_1'$. Hence $P' := P_1'P_2'$ is an Eulerian trail in G'. As the time steps of all edges from P_1' remain the same, that prefix of P' still respects c. For the suffix of P' defined by P_2', observe that all edges are used at time steps later than τ and that none of the edges are inner edges for nodes. Thus, all edges in the P_2' part of P' are also available under c. We conclude that P' is an Eulerian trail in (G', c). ◀

Similar ideas allow us to prove the other direction, yielding the following lemma.

▶ **Lemma 5 (*).** *If there is a c-respecting Eulerian trail in G', G has a Hamiltonian path.*

Lemma 4 and Lemma 5 allow us to deduce Theorem 3.

Consequences. The construction we used is quite restrictive in that it uses only two letters to construct the dBG and is within the framework of interval functions without costs. This allows us to deduce the following two corollaries, narrowing down the hardness landscape of the general problem even further. Corollary 6 was first shown by Hannenhalli et al. [16].

▶ **Corollary 6** ([16]). *The* diET *problem remains* NP*-hard even on 2-regular graphs.*

Proof. Notice that the construction for Theorem 3 uses only the letters $\{\mathtt{A}, \mathtt{T}\}$ and thus the complete dBG (over that alphabet) is 2-regular. ◀

Our hardness reduction also directly implies that dicET is NP-hard: we set the costs of every edge to 0 within its interval and to 1 outside its interval. We also set $c_{\text{budget}} := 0$. As our proof of Theorem 3 relies only on distinguishing zero and non-zero cost instances, we deduce the following inapproximability result, which is to the best of our knowledge new.

▶ **Corollary 7** (*). *If* $\mathsf{P} \neq \mathsf{NP}$ *there is no constant-factor polynomial-time approximation algorithm for the* dicET *problem, even on de Bruijn graphs with interval cost functions.*

Interestingly, the problem remains hard and inapproximable, even when restricting dicET to instances where every interval is the full interval $[m]$ (effectively dropping the interval requirement and only considering edge costs). This reduction simply extends, for each edge, its interval and let all costs outside the original interval be $c_{\text{budget}} + 1$.

Although we can find an Eulerian trail in an undirected graph in linear time by Hierholzer's algorithm [17], counting the number of Eulerian trails in an undirected graph is #P-complete [6]. Given this complexity landscape and our results on directed graphs, it is natural to ask about the complexity of deciding if an Eulerian trail of at most some fixed cost exists. We show that this problem is NP-complete and inapproximable in the full version.

4 FPT by Interval Width

After these strong negative results, it is now natural to ask whether there is a restricted setting in which we can efficiently solve diET. Interval functions (see Definition 1) are a restricted class of functions that are natural to the string reconstruction application. In this setting, it is equally natural to ask if more precise domain knowledge leads to more efficient algorithms. Providing tighter intervals for the time step each edge is to be used reduces the combinatorial complexity as there are fewer valid combinations to consider. This observation also translates to string reconstruction, where we might expect that reconstructing a string is easier when the location of every k-mer is narrowed down to a small interval.

▶ **Definition 8.** *Given a* diET *instance* $((V, E), c)$ *with interval function* c*, we define the instance's* interval width w *as the length of the longest interval of any edge* $e \in E$ *under* c*.*

Interestingly, this concept is closely related to the notion of *interval-membership width* (imw, in short) studied by Bumpus and Meeks in the context of temporal graphs [7]. This notion asks how many edges in the graph are relevant to the decision at a specific time step t by looking at the set of edges which have been available at or before t *and* will also be available at or after t. The imw is the largest size of such a set for any t. In Lemma 9, we show that this parameter is related to the length of the longest interval in our setting (Definition 8). A special case of this observation also underlies the FPT algorithm of Ben-Dor et al [3]. This relates the length of time any single edge is available to the total number of edges relevant at a given time. With this observation, we unveil the equivalence of the algorithms by Bumpus and Meeks and by Ben-Dor et al.

For dBGs of order k, we severely speed up these algorithms by roughly an exponent of $(\log w + 1)/(k-1)$. In fact, we show that on dBGs, the problem is not only in FPT by w, but also by $w \log w/k$; namely, we show that it suffices if w is small compared to k to make the problem tractable. We achieve this by extending the techniques in [3, 7] with the properties of dBGs and their underlying string interpretation. By utilizing a significantly more general setting, Bumpus and Meeks [7] have shown that the algorithm by Ben-Dor et al. [3] applies to a significantly broader graph class than dBGs. We observe that this leaves plenty of dBG-specific structure unexploited to speed up the task. Finally, we show how to extend both the existing and our novel approach to find min-cost Eulerian trails.

Considering Definition 8, the definition of the parametrized problem diET$[w]$ is canonical. We first consider simple graphs, deferring the discussion of multigraphs to the full version.

▶ **Problem.** diET$[w]$

Parameter:	$w \in \mathbb{N}^+$
Given:	Directed graph $G = (V, E)$, interval function $c\colon E \to \mathcal{I}_m$ with interval width w
Decide:	Is there an Eulerian trail $P = e_1 \dots e_m$ such that for all $t \in [m]$, $t \in c(e_t)$?

The following structural insight (Lemma 9) generalizes [3, Lemma 8] and underpins their algorithm and our reduction to the Bumpus and Meeks version of the FPT algorithm: Since any edge is available only at a bounded number of time steps, a similar bound applies to the set of edges available at an individual time step. This insight implies that there are relatively few choices on which edge to select at a specific time step in an Eulerian trail.

▶ **Lemma 9 (*).** *If (G, c, w) is a YES-instance of* diET$[w]$ *and T is an interval in $[m]$, then $|E(T)| \leq |T| + 2w - 2$.*

In particular, Lemma 9 implies that at any specific time step (i.e., an interval of length 1), there are at most $2w-1$ edges available. Thus, we have that any graph with interval width w has imw at most $2w-1$. The condition of this lemma is easy to check in time $\mathcal{O}(mw)$ by testing if the number of edges available at each time step is at most $2w-1$. If this is not the case for any time step, we deduce that (G, c, w) is a NO-instance for diET$[w]$. We will henceforth assume that the condition has been checked and deduce the following result.

▶ **Theorem 10** (Theorem 11 of [3], Theorem 7 of [7])**.** *There is an $\mathcal{O}(m \cdot w^{1.5}4^w)$-time algorithm solving the* diET$[w]$ *problem. Therefore,* diET$[w]$ *is in* FPT.

Three remarks are in order regarding the version of this theorem by Bumpus and Meeks [7]. First, note that this rendering of the theorem requires a small modification, as we are interested in any Eulerian trail, not necessarily a cycle. This allows us to slash a factor of w. Second, their algorithm is for undirected graphs, but it naturally translates to directed graphs. Third, their version of the algorithm does not exclude a small number of irrelevant states and is thus slightly slower (by another factor of $\sqrt{w}$) than the version by Ben-Dor et al. For the sake of completeness, we provide a proof with these modifications in the full version.

Specializing the parametrized technique of [3, 7] to dBGs allows us to design an algorithm that achieves a significant speedup. Our improvements rely on exploiting the specific structure in the dBG graph class in which we are particularly interested in finding Eulerian trails.

▶ **Theorem 11.** *Let G be a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$. There is an $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$-time algorithm solving the* diET$[w]$ *problem, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

Since λ is defined as a minimum, the algorithm underlying Theorem 11 always runs in time $\mathcal{O}(mw \cdot 2^{\frac{w(\log w+1)}{k-1}})$, as shown by the following lemma.

▶ **Lemma 12.** *For any* $w, k \in \mathbb{N}^+$, $(2w-1)^{\frac{w}{k-1}+1} = \mathcal{O}(w \cdot 2^{\frac{w(\log w+1)}{k-1}})$.

Proof. Note that $(2w-1)^{\frac{w}{k-1}+1} = (2w-1) \cdot (2w-1)^{\frac{w}{k-1}}$. We first bound the exponential term. Then multiplying by $(2w-1)$ gives the claimed upper bound. We have that

$$(2w-1)^{\frac{w}{k-1}} = 2^{\frac{w\log(2w-1)}{k-1}} \leq 2^{\frac{w\log(2w)}{k-1}} = 2^{\frac{w(\log w+\log 2)}{k-1}} = 2^{\frac{w(\log w+1)}{k-1}}. \qquad ◀$$

Notice that the exponential part is higher than for the $\mathcal{O}(mw^{1.5} \cdot 4^w)$-time algorithm only if $w(\log w + 1)/(k-1) > 2w$, that is, if $w = \Omega(4^k)$; but in this case, since λ is defined by the minimum, our new algorithm takes $\mathcal{O}(mw \cdot |\Sigma|^w)$ time. Thus, for alphabets of size at most 4, in the extreme edge case, our new algorithm is at least a $\sqrt{w}$ factor faster than the state of the art. Note that since $n \leq |\Sigma|^{k-1}$, for alphabets of size at most 4, this extreme edge case can only occur if w is linear in n, where by our Theorem 3 the problem is not tractable. Specifically, in that case, the running time of both the state of the art and our novel approach become unrealistic.

We describe how to encode a diET[w] instance $(G = (V, E), c, w)$, where G is a dBG, using an auxiliary directed graph $H = (V', E')$ of size $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ in which there is a path between the designated source and target nodes if and only if the instance contains a c-respecting Eulerian trail. The graph H consists of a sequence of $m+1$ *layers* of nodes with edges only between consecutive layers. In this model, traversing an edge of H between layers $t-1$ and t means traversing a corresponding edge of G at time step t. We show the construction of graph $H = (V', E')$ and start by defining the construction of the nodes in V'.

The algorithms in [3, 7] rely on enumerating states, where a state is described by a node v of G (this is the end location of the current partial trail), a time step t, and some information about the edges of the partial trail. Both [3] and [7] achieve their FPT runtime by observing that it is unnecessary to encode the entire partial trail; instead, it suffices to consider the last w edges of it. They are even able to show that it is unnecessary to remember the order these edges are used on the trail. Applying Lemma 9 shows that, for each $v \in V$ and $t \in [m]$, it suffices to consider all subsets of a fixed size. These are at most $\binom{2w-1}{w} = \mathcal{O}(4^w/\sqrt{w})$; see the full version. We will use additional structural insights into dBGs to severely reduce the number of states that need to be considered in many relevant cases.

Henceforth, we set $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$ as in Theorem 11. For ease of notation, we also set $\ell := \lceil \min(w,t)/(k-1) + 1 \rceil$ when t is clear from the context. Intuitively, ℓ is the number of nodes required to uniquely represent a path of length $\min(w,t)$ in G. By id_v we denote the string of length $k-1$ associated with the node $v \in V$.

The Nodes of H. We add a node to V' for every tuple (t, α) such that:

- $t \in [0, m]$ is a time step; and
- $\alpha \in \Sigma^{\min(w,t)+k-1}$ is a string of length $\min(w,t) + k - 1$ over Σ, such that, for all $i \in [\ell]$, the node of G corresponding to the substring $\alpha[(i-1)(k-1)+1 \mathinner{.\,.} i(k-1)]$ of α has an incoming edge labeled $\alpha[i(k-1)]$, available at time step $t - w + (i-1)(k-1)$.

In order to avoid edge casing, we say that the incoming-edge requirement is fulfilled vacuously for time step 0: for layers $t \leq w$, we do not require the first encoded node to have an incoming edge. We also add to V' auxiliary source and target nodes, denoted by s and z, respectively.

▶ Remark 13. While this construction appears overtechnical at first glance, by the definition of a dBG $G = (V, E)$, the string α is equivalent to a sequence of nodes $v_\ell, \ldots, v_1 \in V$, such that v_i has an incoming edge labeled $\mathrm{id}_{v_i}[k-1]$ available at time step $t - (i-1) \cdot (k-1)$.

Intuitively, this construction relies on the insight that any path (of length r) in a dBG is fully described by the string (of length $r + k - 1$) it generates and that this string in turn is fully described by examining every $(k-1)$-th node on the path.

The Edges of H. For a node (t, α) in V', we now potentially create an outgoing edge for each letter $x \in \Sigma$ to $(t+1, \alpha[2\,.\,.\,|\alpha|] \cdot x)$, essentially deleting the leftmost letter of α and then appending x. However, we add the edge $(t, \alpha) \xrightarrow{x} (t+1, \alpha[2\,.\,.\,|\alpha|] \cdot x)$ to E' only if:

- both the head and tail node exist in V'; and
- $\alpha[|\alpha| - k + 2\,.\,.\,|\alpha|] \cdot x$ (the last $k-1$ letters of α appended with x) does not occur in α.

The first condition ensures that the head and tail nodes fulfill the condition that the encoded nodes have incident edges at the required time steps. The second condition ensures that the edge of G with label x that starts at the node v with $\mathrm{id}_v = \alpha[|\alpha| - k + 2\,.\,.\,|\alpha|]$ is not traversed more than once in the (Eulerian) trail we aim to construct.

We also add an edge from the source node s to all nodes in layer 0 (i.e., all nodes where $t = 0$), and from all nodes in layer m (i.e., all nodes where $t = m$), to the target node z.

See Figure 4 for an illustration of this construction.

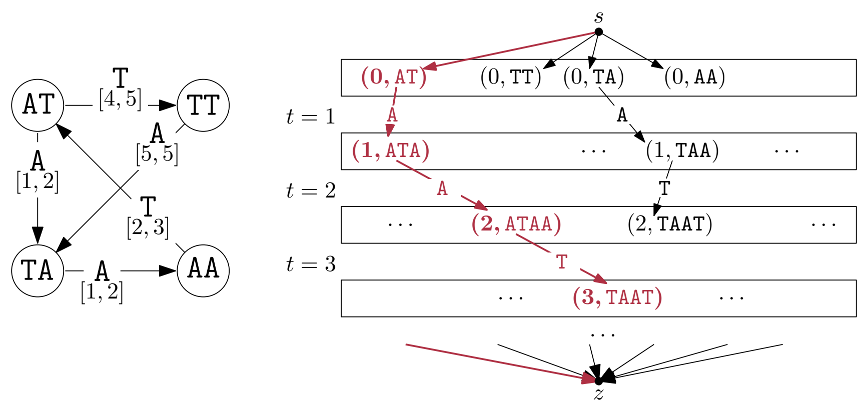


Figure 4 The input de Bruijn graph G (on the left) and the construction of graph H (on the right) for $k = 3$, $|\Sigma| = 2$, $w = 2$, and thus $w + k - 1 = 4$. For instance, for $t = 2$, $\ell = 2$, and $i = 2$, we add node $(t, \alpha) = (2, \texttt{ATAA})$ to layer $t = 2$ in H, because the node of G corresponding to $\alpha[(i-1)(k-1)+1\,.\,.\,i(k-1)] = \texttt{AA}$ has an incoming edge labeled $\alpha[i(k-1)] = \texttt{A}$ available at time step $t - w + (i-1)(k-1) = 2$. A c-respecting Eulerian trail in G is indicated in red in H.

▶ **Lemma 14.** *Graph H has $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ nodes and $\mathcal{O}(m\,|\Sigma| \cdot \lambda^{\frac{w}{k-1}+1})$ edges, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$. Graph H can be constructed in time linear in its maximum size.*

Proof. Firstly, for any layer $t > 0$ in H, there can be at most $|\Sigma|^{w+k-1} = \left(|\Sigma|^{k-1}\right)^{\frac{w}{k-1}+1}$ choices for α. Secondly, by Lemma 9, there are at most $2w - 1$ edges available at any time step $t > 0$, thus there can be at most that many nodes with at least one incoming edge available at t. Thus, by Remark 13, there can be at most $(2w-1)^{\ell}$ such strings α.

By construction, there are precisely n nodes in layer 0. Counting s and z as well and by the choice of λ, there are at most $2 + n + m \cdot \lambda^{\frac{w}{k-1}+1}$ nodes in H. For the edges, observe that every node (except s) has out-degree at most $|\Sigma|$. Node s has out-degree n and node z has in-degree the size of the last layer, that is at most $\lambda^{\frac{w}{k-1}+1}$. This yields the bound on the number of edges. We next discuss how H can be efficiently constructed.

To construct H efficiently, we perform a DFS over its *implicit* representation. For each node (t, α), we store the last letter of α, and associate each incoming edge with the letter it removed from the string in the previous layer to obtain α. The only part that requires a little care is when our DFS is at some node (t, α) and has to decide whether to add an edge with letter $x \in \Sigma$. In such a case, we have to check the condition that the string $\alpha[|\alpha| - k + 2\,.\,.\,|\alpha|] \cdot x$ does not occur in α. Instead of storing the entire string α and performing the pattern matching explicitly, we use a single bit-string of length m during the DFS that maintains the set of w previously used edges (which are equivalent to the strings of length k that can appear in this form). This set can be maintained by correctly flipping the bits corresponding to the edges at the top and at the w-th position in the DFS stack every time the DFS moves forward or backward. In the word RAM model, each edge id is from $[m]$ and so it fits into one machine word. We therefore decide on the existence of each possible edge of H in $\mathcal{O}(1)$ time. To conclude, constructing H takes linear time in its maximum size. ◀

Formalizing the intuition behind the aim of the construction yields the following central lemma. See also Figure 4 that illustrates the properties stated in the lemma.

▶ **Lemma 15 (*).** *For any $t \in [m]$ and $\alpha \in \Sigma^{\min(w,t)+k-1}$, the graph H contains a path from s to the node labeled (t, α) if and only if there exists a trail $W = e_1 \dots e_t$ in G such that:*

1. *every edge from G occurs at most once in W;*
2. *for every $t' \in [t]$, the edge $e_{t'}$ is available at time step t';*
3. *the last w edges of W correspond to α; namely, they start at the node labeled $\alpha[1 \mathinner{.\,.} k-1]$ and then follow the edges corresponding to $\alpha[k \mathinner{.\,.} |\alpha|]$.*

From Lemma 15, we can deduce the following crucial property, which both proves the correctness of our algorithm (i.e., Theorem 11) and underpins our extensions.

▶ **Lemma 16 (*).** *Graph H contains an s-to-z path if and only if there exists an Eulerian trail $W = e_1 \dots e_m$ in G, such that for every $i \in [m]$, edge e_i is available at time step i.*

Lemma 14 and Lemma 16 imply Theorem 11. We stress that, by Lemma 14, if we waive the assumption that $|\Sigma| = \mathcal{O}(1)$, we get an algorithm that is slower only by a factor of $|\Sigma|$.

Generalizations. We now discuss how to exploit this construction to solve more general problems. These extensions work analogously on the algorithm underlying Theorem 10 for general directed graphs. As a first extension, we can use these algorithms to report a witness: a c-respecting Eulerian trail in G. The following result makes this precise.

▶ **Corollary 17 (*).** *For any instance (G, c), where c is an interval function with width w, there is an $\mathcal{O}(m \cdot w^{1.5} 4^w)$-time algorithm finding a c-respecting Eulerian trail in G if it exists and correctly deciding non-existence otherwise. On a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$, we can solve this problem in $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ time, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

For small $w \log w/(k-1)$ values, the algorithm on dBGs is highly efficient. At first glance, that is unintuitive, as even if for each pair of consecutive positions there are only two available k-mers (edges), there could already be $2^{m/2}$ possible orderings. Yet, our result shows that the additional structure of the graph allows us to efficiently select the correct ordering. The following remark captures the significant improvement over previous works which required w to be tractably small to efficiently solve the string reconstruction problem.

▶ Remark 18. Let $|\Sigma| = \mathcal{O}(1)$. For any $\frac{w \cdot \log w}{k-1} = \mathcal{O}(1)$, the diET$[w]$ problem on dBGs can be solved in $\mathcal{O}(mw)$ time. For any $w = \mathcal{O}(1)$, diET$[w]$ on dBGs can be solved in $\mathcal{O}(m)$ time.

It is important to note that in many practical applications, especially in bioinformatics and data privacy (see Figure 1), the dBGs considered are multigraphs (have parallel edges). For our FPT technique, the analysis above relies on the fact that an edge is uniquely determined by its endpoints, but with some extra care it can be adjusted to extend to multigraphs. We give a full discussion in the full version, but we restate the following main observation here.

▶ **Observation 19.** *If there is a c-respecting Eulerian trail in a de Bruijn multigraph $G = (V, E)$, with $m = |E|$, then there is a c-respecting Eulerian trail $W = e_1 \dots e_m$ such that for any time step t where the edge e_t has a parallel edge e' that is also available at t and appears after e_t on W, we have that $\max c(e_t) \leq \max c(e')$.*

Moreover, as mentioned above, we can generalize our technique to interval cost functions and retain the same running time, for both general directed graphs and for dBGs.

▶ **Corollary 20 (*).** *Given a* dicET[w] *instance, there is an $\mathcal{O}(m \cdot w^{1.5} 4^w)$-time algorithm finding a min-cost Eulerian trail in G. On a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$, we can solve this problem in $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ time, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

Note that in this setting, we can allow an edge to become temporarily unavailable by setting its cost to at least $c_{\text{budget}} + 1$. In such a case, the width of the edge's interval remains the difference between the last and the first time step at which the edge is available.

Corollary 20 directly implies that dicET parametrized by w is in FPT. On dBGs, dicET is also in FPT when parametrized by $w \log w/(k-1)$.

5 Counting Eulerian Trails with Applications in Data Privacy

Our results are of crucial relevance for the data privacy applications presented in [4] and [5]. On the one hand, our hardness results (Theorem 3 and Corollary 7) show that *in general* it is indeed hard to reconstruct a private (binary) string from a given dBG, thus providing theoretical justification for the privacy model introduced in [4]. On the other hand, our algorithmic results (Theorem 11 and Corollary 20) show that, with sufficiently specific domain knowledge, one can reconstruct such a (best) string in linear time. To address this concern, the algorithms of [4, 5] rely on a counting routine to efficiently ensure that the released string has at least κ possible reconstructions. To verify that the κ-anonymity property continues to hold even when reconstruction is aided by domain knowledge, we can directly employ the following result as the counting routine in the main algorithm of [4, 5].

▶ **Theorem 21.** *Given a* dicET[w] *instance, we can count the number of min-cost Eulerian trails in $\mathcal{O}(m \cdot w^{1.5} 4^w)$ time. On a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$, we can solve this problem in $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ time, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

Proof. Observe that by Lemma 16 and Corollary 20, there is a one-to-one correspondence between min-cost s-to-z paths in H and min-cost c-respecting Eulerian trails in G. Thus, we can apply the folklore linear-time shortest-path counting DAG dynamic program [25]. ◀

▶ Remark 22. The algorithm underlying Theorem 21 can be trivially extended to enumerate all such Eulerian trails in additional time that is linear in the size of the output.

References

1 Vsevolod A. Afanasev, René van Bevern, and Oxana Yu. Tsidulko. The hierarchical Chinese postman problem: The slightest disorder makes it hard, yet disconnectedness is manageable. *Oper. Res. Lett.*, 49(2):270–277, 2021. doi:10.1016/J.ORL.2021.01.017.

2 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Christoforos L. Raptopoulos. The temporal explorer who returns to the base. *J. Comput. Syst. Sci.*, 120:179–193, 2021. doi:10.1016/J.JCSS.2021.04.001.

3 Amir Ben-Dor, Itsik Pe'er, Ron Shamir, and Roded Sharan. On the complexity of positional sequencing by hybridization. *J. Comput. Biol.*, 8(4):361–371, 2002. doi:10.1089/106652701752236188.

4 Giulia Bernardini, Huiping Chen, Gabriele Fici, Grigorios Loukides, and Solon P. Pissis. Reverse-safe data structures for text indexing. In Guy E. Blelloch and Irene Finocchi, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020, Salt Lake City, UT, USA, January 5-6, 2020*, pages 199–213. SIAM, 2020. doi:10.1137/1.9781611976007.16.

5 Giulia Bernardini, Huiping Chen, Gabriele Fici, Grigorios Loukides, and Solon P. Pissis. Reverse-safe text indexing. *ACM J. Exp. Algorithmics*, 26:1.10:1–1.10:26, 2021. doi:10.1145/3461698.

6 Graham R. Brightwell and Peter Winkler. Counting Eulerian circuits is #P-complete. In Camil Demetrescu, Robert Sedgewick, and Roberto Tamassia, editors, *Proceedings of the Seventh Workshop on Algorithm Engineering and Experiments and the Second Workshop on Analytic Algorithmics and Combinatorics, ALENEX /ANALCO 2005, Vancouver, BC, Canada, 22 January 2005*, pages 259–262. SIAM, 2005. URL: http://www.siam.org/meetings/analco05/papers/09grbrightwell.pdf.

7 Benjamin Merlin Bumpus and Kitty Meeks. Edge exploration of temporal graphs. *Algorithmica*, 85(3):688–716, 2023. doi:10.1007/S00453-022-01018-7.

8 Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, and Alexandru I. Tomescu. Optimal omnitig listing for safe and complete contig assembly. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 29:1–29:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.CPM.2017.29.

9 Massimo Cairo, Romeo Rizzi, Alexandru I. Tomescu, and Elia C. Zirondelli. Genome assembly, from practice to theory: Safe, complete and *Linear-Time*. *ACM Trans. Algorithms*, 20(1):4:1–4:26, 2024. doi:10.1145/3632176.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Moshe Dror, Helman Stern, and Pierre Trudeau. Postman tour on a graph with precedence relation on arcs. *Networks*, 17(3):283–294, 1987. doi:10.1002/NET.3230170304.

12 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *J. Comput. Syst. Sci.*, 119:1–18, 2021. doi:10.1016/J.JCSS.2021.01.005.

13 Thomas Erlebach and Jakob T. Spooner. Faster exploration of degree-bounded temporal graphs. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 36:1–36:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.MFCS.2018.36.

14 Thomas Erlebach and Jakob T. Spooner. Non-strict temporal exploration. In Andréa Werneck Richa and Christian Scheideler, editors, *Structural Information and Communication Complexity - 27th International Colloquium, SIROCCO 2020, Paderborn, Germany, June 29 - July 1, 2020, Proceedings*, volume 12156 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2020. doi:10.1007/978-3-030-54921-3_8.

15 Gianpaolo Ghiani and Gennaro Improta. An algorithm for the hierarchical Chinese postman problem. *Oper. Res. Lett.*, 26(1):27–32, 2000. doi:10.1016/S0167-6377(99)00046-2.

16 Sridhar Hannenhalli, William Feldman, Herbert F. Lewis, Steven Skiena, and Pavel A. Pevzner. Positional sequencing by hybridization. *Comput. Appl. Biosci.*, 12(1):19–24, 1996. doi:10.1093/BIOINFORMATICS/12.1.19.

17 Carl Hierholzer and Chr Wiener. Über die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, 1873.

18 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

19 Peter Korteweg and Ton Volgenant. On the hierarchical Chinese Postman Problem with linear ordered classes. *Eur. J. Oper. Res.*, 169(1):41–52, 2006. doi:10.1016/J.EJOR.2004.06.003.

20 Orna Kupferman and Gal Vardi. Eulerian paths with regular constraints. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *MFCS*, volume 58 of *LIPIcs*, pages 62:1–62:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.62.

21 Antoine Limasset, Jean-François Flot, and Pierre Peterlongo. Toward perfect reads: self-correction of short reads via mapping on de Bruijn graphs. *Bioinform.*, 36(2):651, 2020. doi:10.1093/BIOINFORMATICS/BTZ548.

22 Andrea Marino and Ana Silva. Eulerian walks in temporal graphs. *Algorithmica*, 85(3):805–830, 2023. doi:10.1007/S00453-022-01021-Y.

23 Paul Medvedev. Modeling biological problems in computer science: a case study in genome assembly. *Briefings Bioinform.*, 20(4):1376–1383, 2019. doi:10.1093/BIB/BBY003.

24 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theor. Comput. Sci.*, 634:1–23, 2016. doi:10.1016/J.TCS.2016.04.006.

25 Matús Mihalák, Rastislav Srámek, and Peter Widmayer. Approximately counting approximately-shortest paths in directed acyclic graphs. *Theory Comput. Syst.*, 58(1):45–59, 2016. doi:10.1007/S00224-014-9571-7.

26 Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001. doi:10.1073/pnas.171285098.

27 Pierangela Samarati and Latanya Sweeney. Generalizing data to provide anonymity when disclosing information (abstract). In Alberto O. Mendelzon and Jan Paredaens, editors, *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*, page 188. ACM Press, 1998. doi:10.1145/275487.275508.

28 Jinghao Sun, Guozhen Tan, and Xianchao Meng. Graph transformation algorithm for the time dependent Chinese Postman Problem with Time Windows. In *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, pages 955–960, 2011. doi:10.1109/MEC.2011.6025623.

29 Latanya Sweeney. k-Anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.*, 10(5):557–570, 2002. doi:10.1142/S0218488502001648.

30 Alexandru I. Tomescu and Paul Medvedev. Safe and complete contig assembly through omnitigs. *J. Comput. Biol.*, 24(6):590–602, 2017. doi:10.1089/CMB.2016.0141.

31 Hsiao-Fan Wang and Yu-Pin Wen. Time-constrained Chinese postman problems. *Computers & Mathematics with Applications*, 44(3):375–387, 2002. doi:10.1016/S0898-1221(02)00156-6.