When Is String Reconstruction Using de Bruijn Graphs Hard?

BEN BALS, Centrum Wiskunde en Informatica, Amsterdam, The Netherlands and Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
BENJAMIN ARAM BERENDSOHN, Max-Planck-Institut für Informatik, Saarbrücken, Germany
SEBASTIAAN VAN KRIEKEN, Delft University of Technology, Delft, The Netherlands
SOLON P. PISSIS, The Cyprus Institute, Nicosia, Cyprus
LEEN STOUGIE, Centrum Wiskunde en Informatica, Amsterdam, The Netherlands and Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
HILDE VERBEEK, Centrum Wiskunde en Informatica, Amsterdam, The Netherlands

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? One natural model uses a function that 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 Bruijn 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 $O(m \cdot w^{1.5} 4^w)$ time, where $m = |E|$ and w is the *maximum interval length* over all edges

S. P. Pissis is 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. L. Stougie is 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. H. Verbeek is supported by a Constance van Eeden Fellowship.
Authors' Contact Information: Ben Bals, Centrum Wiskunde en Informatica, Amsterdam, The Netherlands and Vrije Universiteit Amsterdam, Amsterdam, The Netherlands; e-mail: Ben.Justus.Bals@cwi.nl; Benjamin Aram Berendsohn, Max-Planck-Institut für Informatik, Saarbrücken, Germany; e-mail: bberends@mpi-inf.mpg.de; Sebastiaan van Krieken, Delft University of Technology, Delft, The Netherlands; e-mail: sebastiaanvk96@gmail.com; Solon P. Pissis (corresponding author), The Cyprus Institute, Nicosia, Cyprus; e-mail: s.pissis@cyi.ac.cy; Leen Stougie, Centrum Wiskunde en Informatica, Amsterdam, The Netherlands and Vrije Universiteit Amsterdam, Amsterdam, The Netherlands; e-mail: Leen.Stougie@cwi.nl; Hilde Verbeek, Centrum Wiskunde en Informatica, Amsterdam, The Netherlands; e-mail: Hilde.Verbeek@cwi.nl.



This work is licensed under Creative Commons Attribution International 4.0.

© 2026 Copyright held by the owner/author(s).
ACM 1549-6333/2026/10-ART36
https://doi.org/10.1145/3841640

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 $O(mw \cdot 2^{\frac{w(\log w+1)}{k-1}})$ 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.

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

Additional Key Words and Phrases: string algorithm, graph algorithm, de Bruijn graph, Eulerian trail

ACM Reference format:
Ben Bals, Benjamin Aram Berendsohn, Sebastiaan van Krieken, Solon P. Pissis, Leen Stougie, and Hilde Verbeek. 2026. When Is String Reconstruction Using de Bruijn Graphs Hard?. *ACM Trans. Algor.* 22, 4, Article 36 (October 2026), 27 pages.
https://doi.org/10.1145/3841640

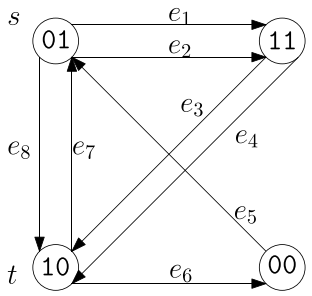
1 Introduction

One of the most important algorithmic tasks in bioinformatics is that of *genome assembly* (or *fragment assembly*) [11, 12, 27, 34]: 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 [30] has led to remarkable progress in the past three decades. This reduction is based on the notion of **de Bruijn graph (dBG)**.

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 dBG* 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 $\mathcal{S}$. Assuming the graph $G_{\mathcal{S},k}$ is Eulerian [30] (otherwise it can be made Eulerian [7, 8, 36]), a single Eulerian trail in $G_{\mathcal{S},k}$ represents a *candidate* string reconstruction; see Figure 1.

Bernardini et al. have introduced the complementary idea in data privacy [5] based on the *z-anonymity* privacy property [31, 33]. 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 six node-distinct Eulerian trails (and thus six distinct strings) corresponding to $\mathcal{S}$. Under the z-anonymity assumption, one cannot know which of the six 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



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

Fig. 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 $\mathcal{S} = \texttt{001}, \texttt{010}, \texttt{011}, \texttt{011}, \texttt{100}, \texttt{101}, \texttt{110}, \texttt{110}$, over the alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$, and $k = 3$.

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. [20] formalized this basic problem in the context of dBGs, it has applications in temporal graphs [10, 26, 28] and other networks [15, 24, 35]. We denote it here by diET, short for directed interval-respecting Eulerian trail (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. [4] then showed that diET is NP-complete, even on dBGs with $|\Sigma| = 4$. We settle the negative landscape by showing the following result:

Theorem 3.1. *The* diET *problem is* NP*-complete, even on dBGs with* $|\Sigma| = 2$.

Beyond theoretically interesting, Theorem 3.1 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. [5]. 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 [20]: 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. [20] showed an algorithm for diET$[w]$ working in $O(m^{2w+\log(2w)})$ time for any directed graph $G = (V, E)$, with $m = |E|$ and $w = O(1)$. Ben-Dor et al. [4] developed an algorithm to solve the problem in $O(m \cdot w^{1.5}4^w)$ time, for any w. Bumpus and Meeks [10] 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 4.3 (Theorem 11 of [4], Theorem 7 of [10]). *There is an $O(m \cdot w^{1.5}4^w)$-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 4.4. *There is an $O(m \cdot \lambda^{\frac{w}{k-1}+1})$-time algorithm solving the* diET$[w]$ *problem, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$ and the input is a dBG of order k over alphabet Σ, $|\Sigma| = O(1)$.*

Theorem 4.4 improves on the state of the art by roughly an exponent of $(\log w + 1)/(k-1)$. The existing algorithms [4, 10, 20] 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 4.4 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 $k = 31$ [25], 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 [1, 15, 19, 23] and the related **Time-Constrained Chinese Postman (TCCP)** problem [32, 35].

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 3.1) translates into inapproximability.

Corollary 3.5. *If* P ≠ NP, *there is no constant-factor polynomial-time approximation algorithm for the* dicET *problem, even on dBGs with interval cost functions.*

We show that the FPT algorithms underlying both Theorems 4.3 and 4.4 also work in the optimization version with an interval cost function, which we denote by dicET$[w]$.

Corollary 4.12. *Given a* dicET$[w]$ *instance, there is an* $O(m \cdot w^{1.5}4^w)$*-time algorithm finding a min-cost Eulerian trail in* G*. On a dBG of order* k *over alphabet* Σ, $|\Sigma| = O(1)$*, we can solve this problem in* $O(m \cdot \lambda^{\frac{w}{k-1}+1})$ *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 5.1. *Given a* dicET$[w]$ *instance, we can count the number of min-cost Eulerian trails in* $O(m \cdot w^{1.5}4^w)$ *time. On a dBG of order* k *over alphabet* Σ, $|\Sigma| = O(1)$*, we can solve this problem in* $O(m \cdot \lambda^{\frac{w}{k-1}+1})$ *time, where* $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.

We can also enumerate these trails in the same time as for counting (Theorem 5.1) 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 [9]. 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 [4, 20], this highly structured setting requires intricate techniques to obtain an (elementary) reduction from the directed Hamiltonian path problem [22]. 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 [22] 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 [14] 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* [2, 10, 16–18, 26, 28]: 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 [10] and by Marino and Silva [26]. 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 [26] 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 [1, 15, 19, 23] (and the related TCCP problem [32, 35]). 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. We also show slight extensions to the algorithms of Ben-Dor et al. and Bumpus and Meeks in Appendix A. Finally, we discuss two extensions: (1) how most of our algorithms extend to multigraphs (Section 6); and (2) how hardness and algorithms are affected if we allow both edges with an interval and edges without (Section 7).

Conference Version. This is an extended version of an article [3] that was published in the *Proceedings of the 33rd Annual European Symposium on Algorithms (ESA 2025).*

2 Preliminaries

For $a, b \in \mathbb{N}$, let $[a, b] := \{x \in \mathbb{N} \mid a \leq x \leq b\}$, $[a, b) := \{x \in \mathbb{N} \mid a \leq x < b\}$, and $[b] := [1, b]$. Further, let $\mathcal{I}_b := \{[i, j] \mid 1 \leq i \leq j \leq b\} \cup \{\emptyset\}$ 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 timesteps (e.g., the second edge is associated with the timestep 2). This vocabulary borrowed from temporal graphs makes the discussion more intuitive.

Definition 2.1. 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 timestep 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 $c(e, t) := \infty$.

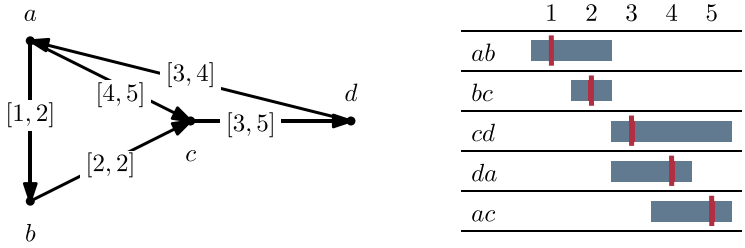


Fig. 2. On the left is the input graph G: every edge is labeled with the timesteps at which it is available. On the right is a table illustrating the interval every edge is available: *abcdac* is an Eulerian trail; the edge usages corresponding to this trail are indicated with red vertical lines.

When the relevant graph $G = (V, E)$ is clear from context, we set $n := |V|$ and $m := |E|$. 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 $c\colon E \times [m] \to \mathbb{Z} \cup \{\infty\}$, $c_{\text{budget}} \in \mathbb{N}$ interval function c' s.t. $c(e, t) = \infty$ iff $t \notin c'(e)$

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

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 [21]. 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.

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

Definition 2.2. We call an edge $e \in E$ *available* at $t \in [m]$, if $t \in c(e)$. For a timestep $t \in [m]$, 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 timestep 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; see Section 3.1. In the undirected case, we consider general graphs, which we believe is nonetheless insightful regarding what makes this problem class hard; see Section 3.2.

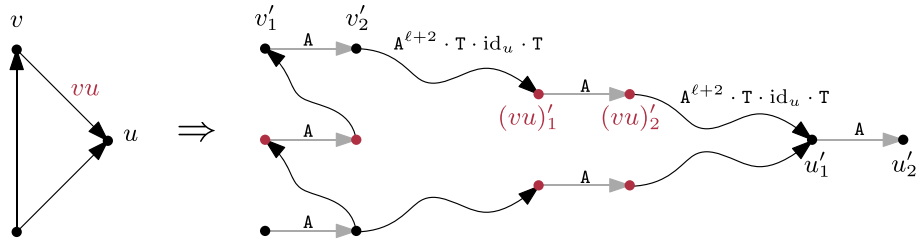


Fig. 3. A directed graph G (left) and the corresponding part of graph G' (right). The nodes in G' that correspond to nodes in G are colored black; the nodes in G' that correspond to edges in G are colored red. Inner edges in G' are colored gray and squiggly lines indicate *paths* of length $2\ell + 4$.

3.1 Directed Graphs

We first settle the computational complexity of diET.

Theorem 3.1. The diET problem is NP-complete, even on dBGs with $|\Sigma| = 2$.

We reduce from the directed Hamiltonian path problem, which is NP-complete [22]. 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 $\Sigma := \{\mathsf{A}, \mathsf{T}\}$ 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 (i.e., the graph with all possible edges) 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. To avoid awkward edge casing later, assume that $|V|$ is a power of two (i.e., that we did not need to round in the choice of ℓ). Note that, by the choice of ℓ and Σ, G' has $|\Sigma|^{k-1} = 2^{4\ell+10} = O(|V|^4)$ nodes. We will pay special attention to some of the nodes in G', to 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:

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

where A^i is the string of i 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 $v'_1 \xrightarrow{\mathsf{A}} v'_2$ 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:

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

Again, we have an edge $e'_1 \xrightarrow{\mathsf{A}} e'_2$ 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 $\mathsf{A}^{\ell+2} \cdot \mathsf{T} \cdot \mathrm{id}_u \cdot \mathsf{T}$ (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.

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 $\mathsf{A}^{\ell+3}$ starting at position $2\ell+5$ (i.e., after the first occurrence of $\mathrm{id}_v \cdot \mathsf{T}$). Any one edge on a path to $(vu)'_1$ shifts this substring left by one letter. By the strategic placement of T around the id parts, the substring $\mathsf{A}^{\ell+3}$ occurs precisely at positions 1 and $2\ell+6$ in $\mathrm{id}_{(vu)'_1}$. Thus, the closest occurrence of $\mathsf{A}^{\ell+3}$ 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 timesteps 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, and the value of τ is chosen exactly based on the sizes of the gadgets in G' corresponding 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 $c(v'_1v'_2) := [1, \tau]$, for all $v \in V$. For any non-edge $e \notin E$, we set $c(e'_1e'_2) := [\tau+1, m']$. 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 3.2. *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

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

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 timesteps 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 $\mathsf{A}^{\ell+3}$ 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 $(v_i)'_2 \overset{\mathsf{A}^{\ell+2}\cdot\mathsf{T}\cdot\mathrm{id}_{v_{i+1}}\cdot\mathsf{T}}{\rightsquigarrow} (v_iv_{i+1})'_1$ (or analogously of the type $(v_iv_{i+1})'_2 \overset{\mathsf{A}^{\ell+2}\cdot\mathsf{T}\cdot\mathrm{id}_{v_{i+1}}\cdot\mathsf{T}}{\rightsquigarrow} (v_{i+1})'_1$) 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 $r_1 \cdot \mathsf{A}^{\ell+3} \cdot \mathsf{T} \cdot \mathrm{id}_{v_i} \cdot \mathsf{T} \cdot r_2$, where r_1 is a suffix of $\mathsf{A}^{\ell+2} \cdot \mathsf{T} \cdot \mathrm{id}_{v_i} \cdot \mathsf{T}$ and r_2 is a prefix of $\mathsf{A}^{\ell+3} \cdot \mathsf{T} \cdot \mathrm{id}_{v_{i+1}} \cdot \mathsf{T}$. Consider that therefore the string corresponding to x contains the substring $\mathsf{A}^{\ell+3} \cdot \mathsf{T} \cdot \mathrm{id}_{v_i} \cdot \mathsf{T}$ 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, $P' := P'_1P'_2$ is an Eulerian trail in G'. As the timesteps 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 timesteps 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 3.3. *If there is a c-respecting Eulerian trail in G', G has a Hamiltonian path.*

Proof. Let P' be a c-respecting Eulerian trail in G'. If P' has the exact structure as in the proof of Lemma 3.2, the result is immediate. As we cannot (without further proof) make such assumptions upon the structure of P', this direction needs a slightly more careful argument. Yet, by our choice of the interval function we obtain the needed structural properties.

First, let $v_1, \ldots, v_n$ be the nodes from V in the order that the edges $(v_i)'_1 \to (v_i)'_2$ appear on P'. Since P' respects c, all of these inner edges for nodes must have timesteps at most τ. For any $i \in [n-1]$, there must be a subtrail from $(v_i)'_2$ to $(v_{i+1})'_1$. By looking at their strings, we see that the longest suffix of $\mathrm{id}_{(v_i)'_2}$ that is a prefix of $\mathrm{id}_{(v_{i+1})'_1}$, is just A. Thus, the path

$$(v_i)'_2 \overset{\mathtt{A}^{\ell+2}\cdot\mathtt{T}\cdot\mathrm{id}_{v_{i+1}}\cdot\mathtt{T}}{\rightsquigarrow} (v_iv_{i+1})'_1 \xrightarrow{\mathtt{A}} (v_iv_{i+1})'_2 \overset{\mathtt{A}^{\ell+2}\cdot\mathtt{T}\cdot\mathrm{id}_{v_{i+1}}\cdot\mathtt{T}}{\rightsquigarrow} (v_{i+1})'_1$$

is the unique shortest path between these two nodes. This path has length $2(2\ell + 4) + 1$. As there are n edges of type $(v_i)'_1 \to (v_i)'_2$, we must have $n - 1$ such connecting paths between them. As $n + (n-1) \cdot (2(2\ell + 4) + 1) = \tau$, we just have enough time to traverse $(v_n)'_1 \to (v_n)'_2$ in time (that is before timestep τ), if we assume the intermediate subtrails as the shortest possible, so since P' respects c, that must be precisely the structure of the length-τ prefix of P'. Therefore, the inner edges for nodes appear on P' connected to each other by a path containing an inner edge for edges. Since such inner edges for (potential) edges are available only if the corresponding edge in G exists, for all $i \in [n-1]$, we have that $v_iv_{i+1} \in E$. Thus, $P := v_1 \ldots v_n$ is a Hamiltonian path in G, as desired. □

Lemmas 3.2 and 3.3 allow us to deduce Theorem 3.1.

Theorem 3.1. *The* diET *problem is* NP*-complete, even on dBGs with $|\Sigma| = 2$.*

Proof. As each node in G' is associated with a string of length $4\lceil \log |V| \rceil + 10$, it has at most $2^{4\log|V|+11} = 2048\,|V|^4$ nodes. Thus, the construction of G' is clearly possible in polynomial time, and the statement follows from Lemmas 3.2 and 3.3.

To see that the problem is in NP consider the canonical certificate that encodes the desired trail as a sequence of nodes. □

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 3.4 was first shown by Hannenhalli et al. [20].

Corollary 3.4 ([20]). *The* diET *problem remains* NP*-hard even on 2-regular graphs (that is, on graphs of which every node has in- and out-degree 2).*

Proof. Notice that the construction for Theorem 3.1 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 3.1 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 3.5. *If* P ≠ NP, *there is no constant-factor polynomial-time approximation algorithm for the* dicET *problem, even on dBGs with interval cost functions.*

Proof. Assume that there is an approximation algorithm that solves this variant of the problem with a constant factor $c \in \mathbb{R}^+$. Now, clearly, if there is a zero-cost Eulerian trail in the constructed instance, the approximation algorithm must also output zero. And conversely, if there is no such zero-cost Eulerian trail then the approximation algorithm must output some non-zero answer. Therefore, a constant-factor polynomial-time approximation algorithm for dicET implies the existence of a polynomial-time algorithm for diET (i.e., the decision variant), which does not exist unless P = NP. □

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 lets all costs outside the original interval be $c_{\text{budget}} + 1$.

3.2 Undirected Graphs

Although we can find an Eulerian trail in an undirected graph in linear time by Hierholzer's algorithm [21], counting the number of Eulerian trails in an undirected graph is #P-complete [9]. 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. In this section, we show that uicET is NP-complete and inapproximable.

We reduce from the undirected Hamiltonian path problem, which is famously known to be NP-complete [22]. Let $G = (V, E)$ be any undirected graph and let $n = |V| \geq 3$. We will construct a uicET instance $(G' = (V', E'), c)$ as follows. Let $V' = (V \times \{1, 2\}) \cup \{x\}$ where $x \notin V$ is a new node. Thus, we duplicate every node v of G into two nodes (which we denote by v'_1 and v'_2). Let $E' = \binom{V'}{2}$, that is, let G' be the complete graph on V'. Set $m' := |E'|$.

As we state in Theorem 3.6 below, we show the result even when all intervals are complete (i.e., $[m']$). For the cost, intuitively, we categorize every edge as *cheap* (cost zero) or *expensive* (cost non-zero) based on two factors: (1) whether we want it to be taken as part of the Hamiltonian path; and (2) whether we want the edge to be taken at an even or odd timestamp.

In our construction, a Hamiltonian path in G corresponds to a sequence of cheap edges in G' that alternates between taking *inner* edges (i.e., ones between v'_1 and v'_2) and *external* edges (i.e., ones connecting different nodes from G). In G', this cheap path can then be extended to a cheap Eulerian trail. The construction will similarly ensure that every cheap Eulerian trail in G' must have a prefix that corresponds to a Hamiltonian path in G. For $v, w \in V$, $i, j \in \{1, 2\}$ and $t \in [|E|]$, an edge $\{v'_i, w'_j\}$ has cost zero at t if and only if:

(1) it is an inner edge (i.e., $v = w$), t is even and $t < 2n + 1$;
(2) there is an edge $\{v, w\} \in E$, t is odd and $t \in [3, 2n + 1)$; or
(3) it is not an inner edge (i.e., $v \neq w$) and $t > 2n + 1$.

Regarding the special node x, set all adjacent edges to always have cost zero, that is $c(\{x, v'_i\}, t) := 0$. All other costs are set to 1. The construction from a Hamiltonian path to an Eulerian trail is illustrated in Figure 4.

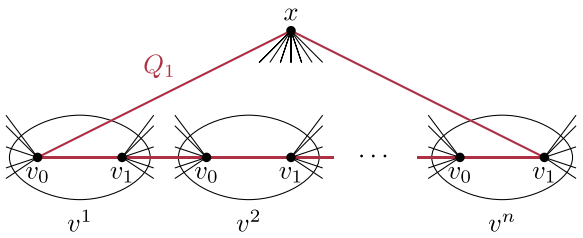


Fig. 4. The zero-cost cycle $Q_1 \subseteq E'$ constructed from a Hamiltonian path in G.

Theorem 3.6. *The* uicET *problem is* NP*-complete, even when all edges are always available.*

Proof. To prove the validity of this reduction, first assume that G has a Hamiltonian path $v^1v^2 \dots v^n$. Then

$$Q_1 := xv_1'^1v_2'^1v_1'^2v_2'^2 \dots v_1'^nv_2'^nx$$

is a cycle in G' of cost zero as all edges are on or before timestamp $2n+1$ and are (1) inner edges for a node from V and at an even timestamp; (2) between nodes corresponding to two different nodes in G sharing an edge and at an odd timestamp; or (3) adjacent to x.

Observe that G' is Eulerian as it is the complete graph on an odd number of nodes. Also, $G' - Q_1$ remains connected as every node, except $v_1'^1$ and $v_2'^n$, has an edge to x. Since $n \geq 3$, the two remaining nodes have at least two edges in $G' - Q_1$, thus at least one to a node that has an edge to x. Thus, $G' - Q_1$ is still Eulerian. Let Q_2 be an Eulerian cycle in $G' - Q_1$. Without loss of generality, assume that Q_2 starts at x. Then, $Q := Q_1Q_2$ is an Eulerian trail in G'. In Q, all edges from Q_2 have timestamp at least $2n+2$ and none of them are inner edges (as all of the inner edges are in Q_1). Thus, Q has a total cost of 0, and we have $(G', c, 0) \in$ uicET as desired.

For the other direction, now assume $(G', c, 0) \in$ uicET. Thus, there is an Eulerian trail Q in G' of cost 0. Therefore, all edges in Q must have cost 0. Let Q_1 be the first $2n+1$ edges of Q. By definition of c the first and last edge in Q_1 must be adjacent to x (otherwise they would not have cost 0). As the second and $2n$-th edge must be an inner edge to have cost 0, we can deduce that the first and last node in Q_1 are x. Similarly, in order for the n inner edges to have cost zero, they must occupy all even timestamps before $2n+1$, thus there are n inner edges in Q_1. Every edge of odd timestamp must be an edge between nodes corresponding to different nodes in G that share an edge in G. Thus, Q_1 must have the form

$$Q_1 = xv_\star'^1v_\star'^1 \dots v_\star'^nv_\star'^nx,$$

where $\star$ may be 1 or 2. Also since there must be n different inner edges in Q_1, for each $v \in V$, the nodes v_1 and v_2 must appear exactly once and consecutively in Q_1. Therefore, $v^1 \dots v^n$ is a Hamiltonian path in G, as desired.

Our reduction shows that uicET is NP-complete. □

Note that our reduction implies inapproximability analogously to Corollary 3.5.

Corollary 3.7. *If* P $\neq$ NP*, there is no constant-factor polynomial-time approximation algorithm for* uicET*, even when all intervals are set to* $[m]$.

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 2.1) 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 timestep 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 4.1. 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 [10]. This notion asks how many edges in the graph are relevant to the decision at a specific timestep 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 4.2, we show that this parameter is related to the length of the longest interval in our setting (Definition 4.1). A special case of this observation also underlies the FPT algorithm of Ben-Dor et al. [4]. 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 [4, 10] with the properties of dBGs and their underlying string interpretation. By utilizing a significantly more general setting, Bumpus and Meeks [10] have shown that the algorithm by Ben-Dor et al. [4] 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 4.1, the definition of the parametrized problem diET$[w]$ is canonical. We first consider simple graphs, deferring the discussion of multigraphs to Section 6.

▶ **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 4.2) generalizes [4, 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 timesteps, a similar bound applies to the set of edges available at an individual timestep. This insight implies that there are relatively few choices on which edge to select at a specific timestep in an Eulerian trail.

Lemma 4.2. *If (G, c, w) is a YES-instance of* diET$[w]$ *and T is an interval in $[m]$, then $|E(T)| \leq |T| + 2w - 2$.*

Proof. Suppose $T = [t, t+p-1]$, so $|T| = p$. The available intervals for all edges in $E(T)$ will be sub-intervals of $[t-w+1, t+p-1+w-1]$, as each edge is available for at most w time steps. This interval has length $(t+p-1+w-1) - (t-w+1) + 1 = p + 2w - 2$. As all edges in $E(T)$ are

only available within this interval, we must be able to traverse them within $p + 2w - 2$ time steps, bounding their number. □

In particular, Lemma 4.2 implies that at any specific timestep (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 interval membership width at most $2w - 1$. The condition of this lemma is easy to check in time $O(mw)$ by testing if the number of edges available at each timestep is at most $2w - 1$. If this is not the case for any timestep, 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 4.3 (Theorem 11 of [4], Theorem 7 of [10]). *There is an $O(m \cdot w^{1.5}4^w)$-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 [10]. 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 $\sqrt{w}$) than the version by Ben-Dor et al. For the sake of completeness, we provide a proof with these modifications in Appendix A.

Specializing the parametrized technique of [4, 10] 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 4.4 *There is an $O(m \cdot \lambda^{\frac{w}{k-1}+1})$-time algorithm solving the* diET$[w]$ *problem, where $\lambda := \min(|\Sigma|^{k-1}, 2w - 1)$ and the input is a dBG of order k over alphabet Σ, $|\Sigma| = O(1)$.*

Since λ is defined as a minimum, the algorithm underlying Theorem 4.4 always runs in time $O(mw \cdot 2^{\frac{w(\log w+1)}{k-1}})$, as shown by the following lemma.

Lemma 4.5. *For any $w, k \in \mathbb{N}^+$, $(2w-1)^{\frac{w}{k-1}+1} = O(w \cdot 2^{\frac{w(\log w+1)}{k-1}})$.*

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

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

□

Notice that the exponential part is higher than for the $O(mw^{1.5} \cdot 4^w)$-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 $O(mw \cdot |\Sigma|^w)$ time. Thus, for alphabets of size at most 4, in that 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 3.1 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 $O(m \cdot \lambda^{\frac{w}{k-1}+1})$ 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 timestep t. We show the construction of graph $H = (V', E')$ and start by defining the construction of the nodes in V'.

The algorithms in [4, 10] 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 timestep t, and some information about the edges of the partial trail. Both [4] and [10] 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 4.2 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 $\binom{2w-1}{w} = O(4^w/\sqrt{w})$; see Appendix A. 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 4.4. For ease of notation, we also set $\ell := \lceil \min(w+1, t)/(k-1) \rceil$ when t is clear from the context. Intuitively, ℓ is the number of nodes required to uniquely represent a walk of length $\min(w, t)$ in G. As we may assume $w \geq k-1 \geq 1$, we have $\ell \leq \min(w, t)/(k-1) + 1$. By id_v we denote the string of length $k-1$ associated with the node $v \in V$; note that, we may abuse this notation and denote by id_W the string of length $k-1+|W|$ for a trail $W = e_1 \dots e_{|W|}$ in G.

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

- $t \in [0, m]$ is a timestep; 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 $\alpha[(i-1)(k-1)+1\,..\,i(k-1)]$ of α has an incoming edge from the node labeled $\alpha[(i-1)(k-1)\,..\,i(k-1)-1]$, available at timestep $t - w + (i-1)(k-1)$.

In order to avoid edge casing, we say that the incoming-edge requirement is fulfilled vacuously for timestep 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 4.6. 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 $v_\ell, \dots, v_1 \in V$, such that v_i has an incoming edge labeled $\mathrm{id}_{v_i}[k-1]$ available at timestep $t - (i-1) \cdot (k-1)$. Also, the slightly complicated condition of the existence of certain edges will be natural as we will construct H in a DFS manner.

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 timesteps. 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 5 for an illustration of this construction.

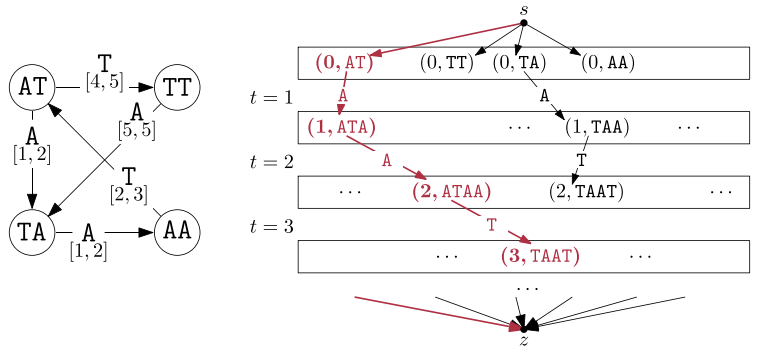


Fig. 5. The input dBG G (on the left) and the construction of graph H (on the right) for $k = 3$, $|\Sigma| = 2$, $w = 2$, and thus $w + k - 1 = 4$. For instance, for $t = 2$, $\ell = 2$, and $i = 2$, we add node $(t, \alpha) = (2, \mathtt{ATAA})$ to layer $t = 2$ in H, because the node of G corresponding to $\alpha[(i-1)(k-1)+1\,..\,i(k-1)] = \mathtt{AA}$ has an incoming edge labeled $\alpha[i(k-1)] = \mathtt{A}$ available at timestep $t - w + (i-1)(k-1) = 2$. A c-respecting Eulerian trail in G is indicated in red in H.

Lemma 4.7. *Graph H has $O(m \cdot \lambda^{\frac{w}{k-1}+1})$ nodes and $O(m\,|\Sigma| \cdot \lambda^{\frac{w}{k-1}+1})$ edges, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$. Graph H can be constructed in time linear in its maximum size.*

Proof. First, 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. Second, by Lemma 4.2, there are at most $2w-1$ edges available at any timestep $t > 0$, thus there can be at most that many nodes with at least one incoming edge available at t. Thus, by Remark 4.6, 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 wth 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 $O(1)$ 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 5 that illustrates the properties stated in the lemma.

Lemma 4.8. *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 $W = e_1 \ldots e_t$ 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 timestep* t'*;*
(3) *the last* $\min(w, t)$ *edges of* W *correspond to* α*; namely, they start at the node labeled* $\alpha[1\,..\,k-1]$ *and then follow the edges corresponding to* $\alpha[k\,..\,|\alpha|]$*.*

Proof. Let v_1 be the last node encoded by α under Remark 4.6.

We will use induction over the layers of H, that is, over t in the statement of the lemma. For the base case, consider the nodes $(0, \alpha)$ that have a direct edge from s. There $|\alpha| = k - 1$, that is, they encode a single node and no edge, thus Condition 3 is fulfilled. Given that we are looking at timestep 0, the trail W is empty, so it trivially exists and vacuously fulfills Condition s 1 and 2.

For the induction step, we start by proving that the existence of a path from s to (t, α') implies the three listed conditions. Since H is arranged in layers, (t, α') must have an incoming edge from some $(t-1, \alpha)$ such that $(t-1, \alpha)$ is reachable from s. By the induction hypothesis (and Condition 3), there exists a valid trail W that ends in the node u_1 corresponding to the last $k-1$ letters in α after $t-1$ timesteps. We will show that the trail W' obtained by appending the outgoing edge from u_1 labeled $\alpha'[|\alpha'|]$ (the last letter of α') to W fulfills the given conditions. This edge exists since (t, α') is a node in H. Let (u_1, v_1) be that edge. Note that $\mathrm{id}_{v_1} = \alpha[|\alpha| - k + 2\,..\,|\alpha|]$ (i.e., v_1 is precisely the node encoded by the last $k-1$ letters of α').

By the construction of the nodes of H, the edge $u_1 \xrightarrow{x} v_1$ is available at t. For all other edges on W', the analogous is true by the induction hypothesis. Thus, Condition 2 holds for W'.

For Condition 1, we show that (u_1, v_1) is not in W. All other edges are unique by the induction hypothesis. By the definition of w, the edge (u_1, v_1) can thus only be available after $t - w + 1$. Thus, if it were to appear on W, it must appear as one of the $w-1$ last edges. Then, the string $\mathrm{id}_{u_1} \cdot x$ is a substring of α', which contradicts our construction of the edges in H, as there would then be no edge from $(t-1, \alpha)$ to (t, α'). Thus, (u_1, v_1) does not appear in W and Condition 1 holds.

Condition 3 follows since $\mathrm{id}_W[|\mathrm{id}_W| - w - k + 2\,..\,|\mathrm{id}_W|] \cdot x = \alpha \cdot x = \mathrm{id}_W[|\mathrm{id}_W| - w - k + 2] \cdot \mathrm{id}_{W'}[|\mathrm{id}_{W'}| - w - k + 2\,..\,|\mathrm{id}_{W'}|] = \mathrm{id}_W[|\mathrm{id}_W| - w - k + 2] \cdot \alpha'$ and thus $\alpha' = \mathrm{id}_{W'}[|\mathrm{id}_{W'}| - w - k + 2\,..\,|\mathrm{id}_{W'}|]$, which is the required property.

The other direction of the implication holds by verifying that such a trail fits the construction of H. □

From Lemma 4.8, we can deduce the following crucial property, which both proves the correctness of our algorithm (i.e., Theorem 4.4) and underpins our extensions.

Lemma 4.9. *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 timestep* i*.*

Proof. By construction of H, there are edges from all nodes in layer m to node z. Lemma 4.8 tells us that there is a path from s to (m, α) if and only if there exists a trail W of m edges in G such that:

(1) every edge occurs at most once in W,
(2) for every $t \in [m]$, the tth edge is available at timestep t.

It follows immediately that W is an Eulerian trail, as it has length m. Since each (m, α) node has an edge going into z, it follows that there is an s-to-z path in H if and only if there exists a valid Eulerian trail in G. □

Lemmas 4.7 and 4.9 imply Theorem 4.4. We stress that, by Lemma 4.7, if we waive the assumption that $|\Sigma| = O(1)$, 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 4.3 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 4.10. *For any instance (G, c), where c is an interval function with width w, there is an $O(m \cdot w^{1.5}4^w)$-time algorithm finding a c-respecting Eulerian trail in G if it exists and correctly deciding non-existence otherwise. On a dBG of order k over alphabet Σ, $|\Sigma| = O(1)$, we can solve this problem in $O(m \cdot \lambda^{\frac{w}{k-1}+1})$ time, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

Proof. For general graphs, the result follows from Lemma A.5. The Eulerian trail W constructed in the proof of the lemma is easy to compute by taking the s-to-z path from H and replacing every edge $(t-1, u, S) \to (t, v, S')$ with the edge $u \to v \in G$.

For dBGs, the result follows from Lemma 4.9. The Eulerian trail W constructed in the proof of the lemma is easy to compute by taking the s-to-z path from H and replacing every edge $(t, u, \alpha) \to (t+1, v, \alpha')$ with the edge $u \to v \in G$. □

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

It is important to note that in many practical applications, especially in bioinformatics and data privacy (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 Section 6, but we restate the following main observation here.

Observation 6.1. *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 timestep 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 4.12. *Given a* dicET$[w]$ *instance, there is an $O(m \cdot w^{1.5}4^w)$-time algorithm finding a min-cost Eulerian trail in G. On a dBG of order k over alphabet Σ, $|\Sigma| = O(1)$, we can solve this problem in $O(m \cdot \lambda^{\frac{w}{k-1}+1})$ time, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

Proof. To see this is a corollary of Corollary 4.10, notice the important change when compared to interval functions: each timestep is associated with a cost.

For general graphs, observe that every Eulerian trail consisting only of available edges in G uniquely corresponds to an s-to-z path W in H by Lemma A.5. If we now modify the construction of H such that an edge $(t-1, v, S)$ to (t, u, S') has cost $c(vu, t)$, each such path $W = \{(v_i, i, S_i)\}_{i \in [0,m]}$ has cost precisely $\sum_{i \in [m]} c(v_{i-1}v_i, i)$, which by definition is the cost of the corresponding Eulerian trail. Thus, a min-cost Eulerian trail in G corresponds now to the min-cost s-to-z path in H. Note that since H is a DAG, we can find a shortest s-to-z path in graph-linear time, thus we do not

asymptotically increase the running time when compared to the unweighted version. Having addressed this issue, the desired result follows.

For dBGs, every Eulerian trail consisting of only available edges in G uniquely corresponds to an s-to-z path W in H by Lemma 4.9. If we modify the construction of H such that an edge $(t-1, \alpha)$ to (t, α') has the cost associated with the last encoded edge e_i in G, each such path $W = \{(i, \alpha_i)\}_{i\in[0,m]}$ has cost $\sum_{i\in[m]} c(e_i, i)$, where e_i is the edge corresponding to the k-mer $\alpha_i[|\alpha_i|-k+1\,..\,|\alpha_i|]$, which by definition is the cost of the corresponding Eulerian trail. Thus, a min-cost Eulerian trail in G corresponds to the min-cost s-to-z path in H. Finally, since H is a DAG, we can find a shortest s-to-z path in linear time, thus asymptotically we retain the running time of the unweighted setting. □

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 timestep at which the edge is available.

Corollary 4.12 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 [5] and [6]. On the one hand, our hardness results (Theorem 3.1 and Corollary 3.5) 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 [5]. On the other hand, our algorithmic results (Theorem 4.4 and Corollary 4.12) show that, with sufficiently specific domain knowledge, one can reconstruct such a (best) string in linear time. To address this concern, the algorithms of [5, 6] rely on a counting routine to efficiently ensure that the released string has at least z possible reconstructions (this is called *z-anonymity*). To verify that the z-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 [5, 6].

Theorem 5.1. *Given a* dicET$[w]$ *instance, we can count the number of min-cost Eulerian trails in* $O(m \cdot w^{1.5}4^w)$ *time. On a dBG of order k over alphabet Σ, $|\Sigma| = O(1)$, we can solve this problem in* $O(m \cdot \lambda^{\frac{w}{k-1}+1})$ *time, where* $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.

Proof. Observe that by Lemma 4.9 and Corollary 4.12, 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 [29]. □

Remark 5.2. The algorithm underlying Theorem 5.1 can be trivially extended to enumerate all such Eulerian trails in additional time that is linear in the size of the output.

6 Extensions to Multigraphs

So far we have only considered simple graphs, but especially in the setting of string reconstruction, dBGs are often applied as multigraphs; see Figure 1. Notably, most of our algorithmic results translate to this important setting. First of all, we observe that the state-of-the-art algorithm by Ben-Dor et al. [4] and Bumpus and Meeks [10] does not use the fact that edges are uniquely defined by their endpoints. Thus, this algorithm, as well as the optimization and counting extensions we introduce, translate immediately to multigraphs.

For our novel algorithm exploiting the dBG structure, we must be more careful, as this approach crucially relies on identifying edges by their corresponding string from Σ^k (or equivalently by their associated letter and start- or end-node). Luckily, the interval availabilities add sufficient structure,

that we can use an exchange-style argument to overcome the challenges raised by this non-unique identification. This relies on the following:

Observation 6.1. *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 timestep 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* $\max c(e_t) \leq \max c(e')$.

Simply put, presented with a choice between two parallel edges that are both available at t, it suffices to greedily choose the edge whose interval ends earlier. This holds since switching any two parallel edges violating the condition again yields a c-respecting Eulerian trail.

Taking care that the bit-string which tracks used edges in our novel algorithm chooses which copy of a multi-edge to mark according to this rule (breaking ties consistently), the correctness of the algorithm on multigraphs follows. Unfortunately, this does not translate to the optimization variant as the exchange argument does not preserve the trail's cost, unless we enforce that parallel edges must have the same cost at any timestep they share.

Since by the above observation, Lemma A.5 translates to multigraphs saying that each s-to-z path in H corresponds to an Eulerian trail whose node sequence is unique, the counting algorithm also works in that setting. Depending on the application, this might even be the more interesting object to count. At least in the string reconstruction setting, counting Eulerian trails that yield different strings is the natural objective [5, 6, 13]. For example, in the data privacy application discussed in Section 5, different Eulerian trails which yield the same string reconstruction do not contribute to the privacy goal.

7 Mixing Free and Interval-Bound Edges

A natural extension to the model presented so far is to allow two types of edges: interval-bound edges (which are assigned an interval of timesteps as before) and free edges (which may appear in any position on the Eulerian trail). Note that this is equivalent to setting the interval of the free edges to $[1, m]$, which would then pose the same problem. Modeling free edges this way will lead to a width of m, making direct use of the above FPT algorithms running in exponential time and space in the number of edges in the input.

Intuitively, free edges award more freedom to an algorithm searching for an Eulerian trail so it would be surprising if they would actually necessitate such a strong deterioration in running time. Again intuitively, the special case where only very few edges are interval-bound and all others are free seems like it should admit an efficient algorithm. Unfortunately, both these intuitions are in fact false.

In particular, we show that on general directed graphs, computing an interval-respecting Eulerian trail is NP-hard on graphs with at least one interval-bound edge, even if the interval has length 1 (i.e., even if there is one edge with one specific time). Using a padding argument, this result can be extended to show that it is NP-hard to compute an interval-respecting Eulerian trail for any instance of the problem with a density $\gamma \in (0, 1)$ of interval-bound edges (i.e., with $\Theta(m^\gamma)$ interval-bound edges). This even holds when we require the present intervals to only contain a single number (i.e., for every interval-bound edge we are given the precise position it should be on the Eulerian trail). Together, these show that for all polynomial densities of free edges, the problem is hard.

This result is complemented by a simple adaptation of the FPT algorithms, showing that the problem can be solved in FPT-time by w of the interval-bound edges on general graphs if the number of free edges is poly-logarithmic. The analogous result holds for our dBG-specific FPT algorithm.

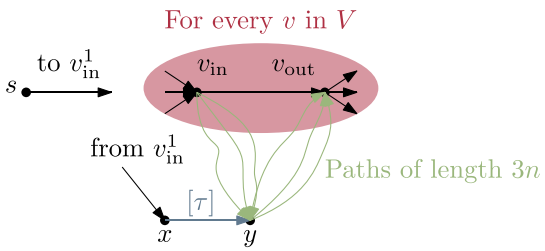


Fig. 6. The hardness construction for Theorem 7.3. The black in-edges for v_{in} correspond to the in-edges of v. Similarly, for the out-edges of v_{out}. For each node v, the number of green paths is chosen such that the nodes v_{in} and v_{out} have the same in as out-degree. The interval assigned to edge xy is $[\tau]$.

Together, these results almost complete the complexity landscape with regard to the proportion of free and interval-bound edges. In particular, if the number of free edges is poly-logarithmic, the problem can be solved in FPT time. If the proportion of free edges is any polynomial in the graph size, the problem is NP-complete, even if we only allow intervals of size one. This only leaves the problem open for the fairly esoteric class of density regimes that are super-polylogarithmic but sub-polynomial.

Formally, consider a (directed) graph $G = (V, E)$ and a function $c\colon E \to \mathcal{I}_m \cup F$, where the value F indicates that an edge is free. An Eulerian trail $T = e_1, \ldots, e_m$ respects c if for all e_i s.t. $c(e_i) \neq F$ we have $i \in c(e_i)$. Write E_F for the free edges in E. The parametrized problem is denoted by $\text{diET}^F[w]$.

7.1 Hardness

For our hardness results, we will reduce from the directed Hamiltonian cycle problem. Let $G = (V, E)$ be the input to the reduction and set $n := |V|$. Fix an arbitrary node v^1. Conceptually, this is the first node on any Hamiltonian cycle we consider. As any cycle can be rotated, this arbitrary choice is harmless.

Create G' as follows (and as shown in Figure 6),

- for every $v \in V$, create nodes v_{in} and v_{out} together with an edge $v_{\text{in}} \to v_{\text{out}}$,
- for every edge $u \to v$ in E, create $u_{\text{out}} \to v_{\text{in}}$,
- create a new node s and an edge $s \to v^1_{\text{in}}$,
- create new nodes x and y as well as edges $v^1_{\text{in}} \to x$ and $x \to y$,
- for every $v \in V$, create $\delta^-(v) - 1$ many directed paths $\{P^i_v\}_{i \in [\delta^-(v)-1]}$ of length $3n$ from v_{in} to y, where $\delta^-(v)$ is the in-degree of v. On each path the internal nodes are nodes in G' that are not in G and that only appear on the path,
- similarly, for every $v \in V$, create $\delta^+(v) - 1$ many directed paths $\{Q^i_v\}_{i \in [\delta^+(v)-1]}$ of length $3n$ from y to v_{out}, where $\delta^+(v)$ is the out-degree of v.

For ease of notation, write $\mathcal{P}$ and $\mathcal{Q}$, respectively, for all the paths created in the two last points. Choose the interval function c such that the only edge that receives an interval is the edge $x \to y$, which is labelled with an interval only containing $\tau := 2n + 3$.

Intuitively, any Hamiltonian cycle in G will translate to a path in G' that starts at s and then passes through the copies of the nodes in the order of the Hamiltonian cycle, before returning to v^1. From there it takes the edges $v^1 \to x \to y$, passing the latter at the exactly right time. The rest of the gadgets then ensure this can be completed into an Eulerian path. The length of the paths $\mathcal{P}$ and $\mathcal{Q}$ ensures that they cannot appear before τ, helping us enforce that any Eulerian trail follows the

form just described, thus ensuring that an Eulerian trail exists in G' if and only if a Hamiltonian cycle exists in G. Let us make this argument formal.

First, observe that G' has size polynomial in that of G. This follows from simply counting the number of vertices on the paths $\mathcal{P}$ and $\mathcal{Q}$, which dominate the size. There are $O(n^3)$ of them. Therefore, G' can be constructed in polynomial time.

Now, let us prove that the reduction works.

Lemma 7.1. *If there is a Hamiltonian cycle in G, then there is a c-respecting Eulerian trail in G'.*

Proof. Without loss of generality, let $C = v^1 \dots v^n v^1$ be a Hamiltonian cycle in G. Consider the following prefix of an Eulerian trail

$$T_1 := s \to v^1_{\text{in}} \to v^1_{\text{out}} \to v^2_{\text{in}} \to \cdots \to v^n_{\text{out}} \to v^1_{\text{in}} \to x \to y. \tag{1}$$

First, observe this is a trail in G'. In particular, the required edges $v^i_{\text{out}} \to v^{i+1}_{\text{in}}$ exist as C is a Hamiltonian cycle in G. Also, $x \to y$ appears on T_1 at position τ, as demanded by our interval function. It remains to see that T_1 can be completed to an Eulerian trail. This follows as in the graph $G' - T_1$, all nodes have the same in- as out-degree by the construction of the paths $\mathcal{P}$ and $\mathcal{Q}$. As $x \to y$ is at the correct position and no other edges have an associated interval, the completed trail is c-respecting. □

Lemma 7.2. *If there is a c-respecting Eulerian trail in G', then there is a Hamiltonian cycle in G.*

Proof. Let T be a c-respecting Eulerian trail in G' and write T_1 for the prefix of T up to and including the edge $x \to y$. The first node on T must be s as it is the unique node such that its out-degree is one more than its in-degree.

The core observation is now that none of the edges from the paths $\mathcal{P}$ and $\mathcal{Q}$ can appear on T_1. This holds as if one such edge appeared on T_1, then there is an edge from $\mathcal{P}$ or $\mathcal{Q}$ that is connected to $x \to y$ with a path of length less than $|T_1| = \tau$. By construction, this is impossible in G'.

Thus, there must be a sequence of nodes $v^1, \dots, v^n$ from G such that T_1 has the form

$$T_1 := s \to v^1_{\text{in}} \to v^1_{\text{out}} \to v^2_{\text{in}} \to \cdots \to v^n_{\text{out}} \to v^1_{\text{in}} \to x \to y. \tag{2}$$

Compare this to the previous lemma. The nodes $v^1, \dots, v^n$ must be without repetition as from any v^i_{in} there is exactly one out-edge (namely $v^i_{\text{in}} \to v^i_{\text{out}}$) that is not on the paths $\mathcal{P}$. Thus, only this edge can appear on T_1 and thus every node v^i_{in} can only appear once on T_1. The edges $v^i \to v^{i+1}$ exist in G as T_1 witnesses that the edges $v^i_{\text{out}} \to v^{i+1}_{\text{in}}$ exist in G'.

From these facts, we can conclude that $v^1 \dots v^n v^1$ is a Hamiltonian cycle in G. □

With these helper results in hand, we can prove the main theorem of this subsection.

Theorem 7.3. *There is an infinite family of graphs where in each graph one edge has a fixed constant-sized interval such that deciding if there is an Eulerian trail respecting these intervals is* NP*-hard.*

Corollary 7.4. *Let $\gamma \in [0, 1]$. Then, there is an infinite family of graphs where in each graph $\Theta(m^\gamma)$ of the total m edges have a fixed constant-sized interval such that deciding if there is an Eulerian trail respecting these intervals is* NP*-hard.*

Proof. The above construction gives an infinite family of graphs where exactly $1 = m^0$ of the edges are interval-bound. The previous two lemmas show that deciding if there is a c-respecting Eulerian trail on them is NP-hard.

To extend this to families for an arbitrary polynomial density of interval bound edges m^γ, simply add a path of length m^γ between s and v^1_{in} (here m is the size of the graph G'). Additionally, add the length of this path to the label of the edge $x \to y$. □

7.2 Adapting the FPT Algorithms

Consider the algorithm from Section 4. Augment every state by also storing a set $S_F \subseteq E_F$ of used free edges. The rest of the algorithm adapts naturally: treat a free edge like an edge that is always available and update S_F appropriately. Concretely, for any state transition corresponding to an interval-bound edge, the set S_F does not change. For any free edge e, create a transition between two states (t, α, S_F) and $(t+1, \alpha, S'_F)$ if (1) $e \notin S_F$, (2) S'_F is exactly $S_F \cup e$, and (3), as before, α' is obtained from α by the string operation corresponding to e.

The correctness argument follows the same as before. Also note that the algorithms of Ben-Dor et al. and Bumpus and Meeks can be adapted to support free edges in exactly the same manner.

Consider the graph $G - E_F$ and construct the previous state graph on it. Compare this to the new state graph just described. The new state graph has at most $2^{|S_F|}$ states for every state in the previous state graph, yielding the following result.

Theorem 7.5. *Let G be a dBG of order k over alphabet Σ, $|\Sigma| = O(1)$, together with a mixed interval function. There is an $O(m \cdot \lambda^{\frac{w}{k-1}+1} \cdot 2^{|E_F|})$-time algorithm solving the* $\text{diET}^F[w]$ *problem, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$ and $|E_F|$ is the number of free edges.*

References

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

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

[3] Ben Bals, Sebastiaan van Krieken, Solon P. Pissis, Leen Stougie, and Hilde Verbeek. 2025. When is string reconstruction using de Bruijn graphs hard? In *Proceedings of the 33rd Annual European Symposium on Algorithms (ESA '25)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Article 53, 1–16. DOI: https://doi.org/10.4230/LIPICS.ESA.2025.53

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

[5] Giulia Bernardini, Huiping Chen, Gabriele Fici, Grigorios Loukides, and Solon P. Pissis. 2020. Reverse-safe data structures for text indexing. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX '20)*. SIAM, 199–213. DOI: https://doi.org/10.1137/1.9781611976007.16

[6] Giulia Bernardini, Huiping Chen, Gabriele Fici, Grigorios Loukides, and Solon P. Pissis. 2021. Reverse-safe text indexing. *ACM J. Exp. Algorithmics* 26 (2021), Article 1.10, 1–26. DOI: https://doi.org/10.1145/3461698

[7] Giulia Bernardini, Huiping Chen, Inge Li Gørtz, Christoffer Krogh, Grigorios Loukides, Solon P. Pissis, Leen Stougie, and Michelle Sweering. 2024. Connecting de Bruijn graphs. In *Proceedings of the 35th Annual Symposium on Combinatorial Pattern Matching (CPM '24)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Article 6, 1–16. DOI: https://doi.org/10.4230/LIPICS.CPM.2024.6

[8] Giulia Bernardini, Huiping Chen, Grigorios Loukides, Solon P. Pissis, Leen Stougie, and Michelle Sweering. 2022. Making de Bruijn graphs Eulerian. In *Proceedings of the 33rd Annual Symposium on Combinatorial Pattern Matching (CPM '22)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Article 12, 1–18. DOI: https://doi.org/10.4230/LIPICS.CPM.2022.12

[9] Graham R. Brightwell and Peter Winkler. 2005. Counting Eulerian circuits is #P-complete. In *Proceedings of the 7th Workshop on Algorithm Engineering and Experiments and the 2nd Workshop on Analytic Algorithmics and Combinatorics (ALENEX /ANALCO '05)*. SIAM, 259–262. Retrieved from http://www.siam.org/meetings/analco05/papers/09gr-brightwell.pdf

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

[11] Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, and Alexandru I. Tomescu. 2017. Optimal omnitig listing for safe and complete contig assembly. In *Proceedings of the 28th Annual Symposium on Combinatorial Pattern Matching (CPM '17)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Article 29, 1–12. DOI: https://doi.org/10.4230/LIPICS.CPM.2017.29

[12] Massimo Cairo, Romeo Rizzi, Alexandru I. Tomescu, and Elia C. Zirondelli. 2024. Genome assembly, from practice to theory: Safe, complete and linear-time. *ACM Trans. Algorithms* 20, 1 (2024), Article 4, 1–26. DOI: https://doi.org/10.1145/3632176

[13] Alessio Conte, Roberto Grossi, Grigorios Loukides, Nadia Pisanti, Solon P. Pissis, and Giulia Punzi. 2021. Beyond the BEST theorem: Fast assessment of Eulerian trails. In *Proceedings of the 23rd International Symposium on Fundamentals of Computation Theory (FCT '21)*. Evripidis Bampis and Aris Pagourtzis (Eds.), Lecture Notes in Computer Science, Vol. 12867, Springer, 162–175. DOI: https://doi.org/10.1007/978-3-030-86593-1

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

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

[16] Thomas Erlebach, Michael Hoffmann, and Frank Kammer. 2021. On temporal graph exploration. *J. Comput. Syst. Sci.* 119 (2021), 1–18. DOI: https://doi.org/10.1016/J.JCSS.2021.01.005

[17] Thomas Erlebach and Jakob T. Spooner. 2018. Faster exploration of degree-bounded temporal graphs. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS '18)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Article 36, 1–13. DOI: https://doi.org/10.4230/LIPICS.MFCS.2018.36

[18] Thomas Erlebach and Jakob T. Spooner. 2020. Non-strict temporal exploration. In *Proceedings of the 27th International Colloquium on Structural Information and Communication Complexity (SIROCCO '20)*. Andréa Werneck Richa *and* Christian Scheideler (Eds.), *Lecture Notes in Computer Science*, Vol. 12156, Springer, 129–145. DOI: https://doi.org/10.1007/978-3-030-54921-3

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

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

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

[22] Richard M. Karp. 1972. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations (The IBM Research Symposia Series)*. Plenum Press, New York, 85–103. DOI: https://doi.org/10.1007/978-1-4684-2001-2

[23] Peter Korteweg and Ton Volgenant. 2006. On the hierarchical Chinese postman problem with linear ordered classes. *Eur. J. Oper. Res.* 169, 1 (2006), 41–52. DOI: https://doi.org/10.1016/J.EJOR.2004.06.003

[24] Orna Kupferman and Gal Vardi. 2016. Eulerian paths with regular constraints. In *Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Article 62, 1–15. DOI: https://doi.org/10.4230/LIPIcs.MFCS.2016.62

[25] Antoine Limasset, Jean-François Flot, and Pierre Peterlongo. 2020. Toward perfect reads: Self-correction of short reads via mapping on de Bruijn graphs. *Bioinformatics* 36, 5 (2020), 1374–1381. DOI: https://doi.org/10.1093/bioinformatics/btz102

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

[27] Paul Medvedev. 2019. Modeling biological problems in computer science: A case study in genome assembly. *Briefings Bioinform.* 20, 4 (2019), 1376–1383. DOI: https://doi.org/10.1093/BIB/BBY003

[28] Othon Michail and Paul G. Spirakis. 2016. Traveling salesman problems in temporal graphs. *Theor. Comput. Sci.* 634 (2016), 1–23. DOI: https://doi.org/10.1016/J.TCS.2016.04.006

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

[30] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. 2001. An Eulerian path approach to DNA fragment assembly. *Proc. Natl Acad. Sci.* 98, 17 (2001), 9748–9753. Retrieved from https://www.pnas.org/doi/pdf/10.1073/pnas.171285098

[31] Pierangela Samarati and Latanya Sweeney. 1998. Generalizing data to provide anonymity when disclosing information (abstract). In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM Press, 188. DOI: https://doi.org/10.1145/275487.275508

[32] Jinghao Sun, Guozhen Tan, and Xianchao Meng. 2011. Graph transformation algorithm for the time dependent Chinese postman problem with time windows. In *Proceedings of the 2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, 955–960. DOI: https://doi.org/10.1109/MEC.2011.6025623

[33] Latanya Sweeney. 2002. k-Anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* 10, 5 (2002), 557–570. DOI: https://doi.org/10.1142/S0218488502001648
[34] Alexandru I. Tomescu and Paul Medvedev. 2017. Safe and complete contig assembly through omnitigs. *J. Comput. Biol.* 24, 6 (2017), 590–602. DOI: https://doi.org/10.1089/CMB.2016.0141
[35] Hsiao-Fan Wang and Yu-Pin Wen. 2002. Time-constrained Chinese postman problems. *Comput. Math. Appl.* 44, 3 (2002), 375–387. DOI: https://doi.org/10.1016/S0898-1221(02)00156-6
[36] Wiktor Zuba, Oded Lachish, and Solon P. Pissis. 2025. Shortest undirected paths in de Bruijn graphs. In *Proceedings of the 36th Annual Symposium on Combinatorial Pattern Matching (CPM '25)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Article 12, 1–13. DOI: https://doi.org/10.4230/LIPICS.CPM.2025.12

Appendix

A Generalizations of the Algorithms by Ben-Dor et al. and Bumpus and Meeks

First, we discuss our notation for the algorithm. This also shows how to shave off a factor of w in exchange for computing an Eulerian trail instead of an Eulerian cycle compared to the Bumpus and Meeks version. For completeness, we give a correctness proof, but the argument is entirely analogous to the one presented in [4] and [10].

Lemma A.1. *Let $t \in [m]$ be a timestep. The total number of sets $S \subseteq E(t)$, for which there exists some Eulerian trail $P = e_1 \dots e_m$, such that $\{e_i \in P \mid i \in [0,t], e_i \in E(t)\} = S$, is in $O(4^w/\sqrt{w})$. Furthermore, all such sets S have size $t-a$, where a is the number of edges which will not be available at or after t.*

Proof. Call an edge e expired at t if $\max c(e) \leq t$. Let $\mathcal{S}(t)$ be the set of sets S such that there exists some Eulerian trail $P = e_1 \dots e_m$, with $\{e_i \in P \mid i \in [0,t], e_i \in E(t)\} = S$. We will show that any such $S \in \mathcal{S}(t)$ has size $t-a$, where a is the number of edges expired before t. Let S be an arbitrary set in $\mathcal{S}(t)$ and let P be an Eulerian trail $P = e_1 \dots e_m$ such that $S = \{e_i \in P \mid i \in [0,t], e_i \in E(t)\}$. Then, at time t, P must have traversed all edges that expire before time t. Since, at timestep t, t steps have passed, $|S| = |\{i \in [0,t] \mid e_i \in E(t)\}| = t - a$. S was taken arbitrarily, so all such sets have size $t-a$. This means that $\mathcal{S}(t) \subseteq \{T \subseteq E(t) \mid |T| = t-a\}$ and thus $|\mathcal{S}(t)| \leq \binom{|E(t)|}{t-a}$. We also know by Lemma 4.2, that $|E(t)| \leq 2w-1$, so $|\mathcal{S}(t)| \leq \binom{2w-1}{(t-a)}$. Therefore, $|\mathcal{S}(t)| \leq \binom{2w-1}{w} \sim \frac{4^w}{\sqrt{w}} \in O(\frac{4^w}{\sqrt{w\pi}})$. □

We now define the construction of the nodes in V'. We create a node in V' for every triple (t, v, S) such that

- $t \in [0, m]$ is a timestep,
- $v \in V$ is a node from G which has at least one incoming edge available at t,
- $S \subseteq E(t)$, if $|S| = t - a(t)$, where $a(t)$ is the number of edges that have expired at time t (for simplicity, we say $E(0) = \emptyset$), and
- if $t > 0$, there is some edge in S that ends in v

We also create auxiliary source and target nodes s and z, respectively.

Construct the edges in E' as follows. Every edge either connects two nodes with consecutive time steps, or it connects s to a node with timestep 0 or a node with timestep m to z. For the source and target nodes, we add the edges $(s, (0, v, \emptyset))$ and $((m, v, S), z)$ for every node $v \in V$ and $S \subseteq E(m)$. We define the edges between consecutive timesteps as follows; refer to Figure A1 for an illustration of the construction.

Definition A.2. Let $(t-1, u, S)$ and (t, v, S') be two nodes from V'. We add the edge $((t-1, u, S), (t, v, S'))$ to E' if and only if

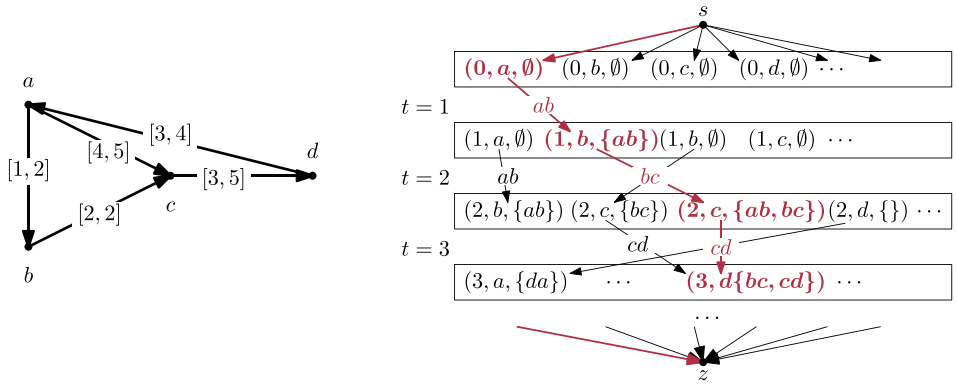


Fig. A1. The construction of graph H (on the right) for the graph G (on the left). The Eulerian trail in G is indicated in red in H. Some of the nodes and edges in H are omitted for legibility.

(1) (u, v) is an edge in G and available at time step t,
(2) $(u, v) \notin S$, and
(3) $S' = (S \cap E(t)) \cup \{(u, v)\}$.

Note that by constructing H using an implicit DFS, we can exclude nodes that do not have an incoming edge. In the next lemma, we prove an upper bound on both the number of nodes and edges of H if it is created this way, as functions of n, m and w.

Lemma A.3. *Graph H contains at most $2 + n + m \cdot O(\sqrt{w} \cdot 4^w)$ nodes and $2n + m \cdot O(w^{1.5} \cdot 4^w)$ edges, where n and m are the number of nodes resp. edges in G.*

Proof. By Lemma 4.2, we have that $|E(t)| \leq 2w - 1$ for every timestep $t > 0$. Every node other than s and z is labeled with a timestep $t \in [0, m]$, a node $v \in V$, and a set $S \subseteq E(t)$. Moreover, v must be the endpoint of one of the edges in $E(t)$, meaning the number of choices for v comes down to selecting one of the at most $2w - 1$ edges in $E(t)$. By Lemma A.1, the number of sets $S \subseteq E(t)$ we have to consider is bounded by $O(4^w/\sqrt{w})$. Thus, for timesteps greater than zero, there are at most $O(\sqrt{w} \cdot 4^w)$ distinct triples. For nodes with timestep zero, we have $E(0) = \emptyset$, and so there are just n such nodes. Adding nodes s and z gives a total node count of $2 + n + m \cdot O(\sqrt{w} \cdot 4^w)$.

For the number of edges, we first consider the outgoing edges of all labeled nodes with a timestep less than m. For a given node (t, u, S), all outgoing edges go to nodes labeled $(t, v, (S \cap E(t+1)) \cup \{(u, v)\})$ for some $(u, v) \in E(t+1)$. Since by Lemma 4.2, $|E(t+1)| \leq 2w - 1$, we have at most $2w - 1$ outgoing edges for each such labeled node. All other edges in H are incident on either s or z, with n such edges for each, giving a total edge count of $2n + m \cdot O(w^{1.5} \cdot 4^w)$. □

Let us remark that given a node (t, v, S) of H, its outgoing edges can easily be determined by iterating over v's outgoing edges (that are not already in S) and checking their availability at timestep $t + 1$.

In the next lemma, we prove a one-to-one correspondence between length-t paths in H starting from the source node and length-t trails in G:

Lemma A.4. *For any $t \in [m], v \in V, S \subseteq E(t)$, we have that H contains a path from s to the node labeled (t, v, S) 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, the edge e_t is available at timestep t,*

(3) *W ends in v unless it is empty (either v is the endpoint of e_t or $t = 0$), and*
(4) *$W \cap E(t) = S$ (i.e., S contains all available but already used edges).*

Proof. We use induction over the layers of H, that is over t in the statement of the lemma. For the base case, consider the nodes $(0, v, \emptyset)$ which have a direct edge from s. Given that we are looking at timestep 0, the trail W is empty so it trivially exists. Moreover, $W \cap E(0) = \emptyset$ fulfilling Condition 4.

For the induction step, we start by proving that the existence of a path from s to (t, v, S') implies the four listed conditions. By this assumption, (t, v, S') has an incoming edge from some $(t-1, u, S)$ such that $(t-1, u, S)$ is reachable from s. By the induction hypothesis, there exists a valid trail W that ends in u after $t-1$ timesteps, with $W \cap E(t-1) = S$. We will show that the trail W' obtained by appending (u, v) to W fulfills the given conditions.

For Condition 1, we show that (u, v) is not in W. All other edges are unique by the induction hypothesis. Note that because there is an edge from $(t-1, u, S)$ to (t, v, S'), it must be that $(u, v) \notin S$; otherwise, this edge would not fulfill Condition 2 from Condition A.2. Similarly, (u, v) must be available at timestep t by Condition 1 of Definition A.2. Now assume (u, v) appears at some earlier timestep t' on W. Then, because we are considering interval functions, it must be available at $t-1$ as well since $t' \leq t-1 < t$. Thus $(u, v) \in W \cap E(t-1) = S$, which violates Condition 2 of Definition A.2.

Condition 2 holds trivially due to the corresponding edge existing in H. Condition 3 also holds trivially, as the final edge of W' is (u, v).

For Condition 4, combine that we already know by the induction hypothesis that $W \cap E(t-1) = S$ with the fact that, by the construction of H, we have $S' = (S \cap E(t)) \cup \{(u, v)\}$.

To prove the implication in the other direction, suppose there exists some (non-empty) trail $W' = e_1 \dots e_t$ for which the listed conditions hold. Let $t \geq 1$, $e_t = (u, v)$ and $S' = W' \cap E(t)$. We will show that the node (t, v, S') is reachable in H. Let $W = W' \setminus \{(u, v)\}$ and let $S = W \cap E(t-1)$. W and S fulfill the four conditions, so by the induction hypothesis, the node $(t-1, u, S)$ is reachable in H. We will now show that the edge $((t-1, u, S), (t, v, S'))$ exists in H using the conditions from Definition A.2.

Conditions 1 and 2 are trivially true. For Condition 3, we have to show that $S' = (S \cap E(t)) \cup \{(u, v)\}$, which holds since $S' = W' \cap E(t) = (W \cap E(t-1) \cap E(t)) \cup \{(u, v)\} = (S \cap E(t)) \cup \{(u, v)\}$. With all three conditions proven, the edge $((t-1, u, S), (t, v, S'))$ must exist, making the latter node reachable. □

Lemma A.5. *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 timestep i.*

Proof. By construction of H, there are edges from all nodes $(m, v, E(m))$ to z. Lemma A.4 tells us that there is a path from s to $(m, v, E(m))$ if and only if there exists a trail W of m edges in G such that:

(1) every edge occurs at most once in W,
(2) for every $t \in [m]$, the tth edge is available at timestep t,
(3) W ends in node v, and
(4) $E(m) \subseteq W$.

It follows immediately that W is an Eulerian trail, as it has length m. Since each $(m, v, E(m))$ node has an edge going into z, it follows that there is an s-to-z path in H if and only if there exists a valid Eulerian trail in G. □

Proof of Theorem 4.3. We solve the problem by constructing the graph H and checking if there exists an s-to-z path. By Lemma A.3, H takes $O(m \cdot w^{1.5}4^w)$ space, and it can be constructed in the same time by first creating all the nodes and then creating the outgoing edges for each node by iterating over its available edges. Identifying an s-to-z path is possible using, for example, depth-first search, which takes time linear in the size of H. By Lemma A.5, there exists a valid Eulerian trail in G if and only if we find an s-to-z path in H; after deciding if it exists, we can retrieve the trail by observing the labels of the nodes on the s-to-z path. □

Received 2 April 2026; revised 2 April 2026; accepted 13 August 2026