



# The Complexity of Approximate Pattern Matching on de Bruijn Graphs

Daniel Gibney<sup>1</sup>(✉), Sharma V. Thankachan<sup>2</sup>, and Srinivas Aluru<sup>1</sup>

<sup>1</sup> School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, USA

[daniel.j.gibney@gmail.com](mailto:daniel.j.gibney@gmail.com), [aluru@cc.gatech.edu](mailto:aluru@cc.gatech.edu)

<sup>2</sup> Department of Computer Science, University of Central Florida, Orlando, USA  
[sharma.thankachan@gmail.com](mailto:sharma.thankachan@gmail.com)

**Abstract.** Aligning a sequence to a *walk* in a labeled graph is a problem of fundamental importance to Computational Biology. For finding a walk in an arbitrary graph with  $|E|$  edges that exactly matches a pattern of length  $m$ , a lower bound based on the Strong Exponential Time Hypothesis (SETH) implies an algorithm significantly faster than  $\mathcal{O}(|E|m)$  time is unlikely [Equi *et al.*, ICALP 2019]. However, for many special graphs, such as de Bruijn graphs, the problem can be solved in linear time [Bowe *et al.*, WABI 2012]. For approximate matching, the picture is more complex. When edits (substitutions, insertions, and deletions) are only allowed to the pattern, or when the graph is acyclic, the problem is again solvable in  $\mathcal{O}(|E|m)$  time. When edits are allowed to arbitrary cyclic graphs, the problem becomes NP-complete, even on binary alphabets [Jain *et al.*, RECOMB 2019]. These results hold even when edits are restricted to only substitutions. Despite the popularity of de Bruijn graphs in Computational Biology, the complexity of approximate pattern matching on de Bruijn graphs remained open. We investigate this problem and show that the properties that make de Bruijn graphs amenable to efficient exact pattern matching do not extend to approximate matching, even when restricted to the substitutions only case with alphabet size four. Specifically, we prove that determining the existence of a matching walk in a de Bruijn graph is NP-complete when substitutions are allowed to the graph. In addition, we demonstrate that an algorithm significantly faster than  $\mathcal{O}(|E|m)$  is unlikely for de Bruijn graphs in the case where only substitutions are allowed to the pattern. This stands in contrast to pattern-to-text matching where exact matching is solvable in linear time, like on de Bruijn graphs, but approximate matching under substitutions is solvable in subquadratic  $\tilde{\mathcal{O}}(n\sqrt{m})$  time, where  $n$  is the text's length [Abrahamson, SIAM J. Computing 1987].

**Keywords:** Approximate pattern matching · Sequence alignment · de Bruijn graphs · Computational complexity

## 1 Introduction

De Bruijn graphs are an essential tool in Computational Biology. Their role in *de novo* assembly spans back to the 1980s [40], and their application in assembly has been researched extensively since then [9, 10, 17, 33, 38, 39, 43, 46]. More recently, de Bruijn graphs have been applied in metagenomics and in the representation of large collections of genomes [14, 27, 30, 37, 45] and for solving other problems such as read-error correction [32, 35] and compression [8, 24]. Due to the popularity of de Bruijn graphs in the modeling of sequencing data, an algorithm to efficiently find *walks* in a de Bruijn graph matching (or approximately matching) a given query pattern would be a significant advancement. For example, in metagenomics, such an algorithm could quickly detect the presence of a particular species within genetic material obtained from an environmental sample. Or, in the case of read-error correction, such an algorithm could be used to efficiently find the best mapping of reads onto a ‘cleaned’ reference de Bruijn graph with low-frequency k-mers removed [32]. To facilitate such tasks, several algorithms (often seed-and-extend type heuristics) and software tools have been developed that perform pattern matching on de Bruijn (and general) graphs [5, 22, 23, 29, 31, 34, 36, 42].

The importance of pattern matching on labeled graphs in Computational Biology and other fields has caused a recent surge of interest in the theoretical aspects of this problem. In turn, this has led to many new fascinating algorithmic and computational complexity results. However, even with this improved understanding of the theory of pattern matching on labeled graphs, our knowledge is still lacking in many respects concerning specific, yet extremely relevant, graph classes. An overview of the current state of knowledge is provided in Table 1.

**Table 1.** The computational complexity of pattern matching on labeled graphs

	Exact matching	Approximate matching
Easy	<u>Solvable in linear time</u> <ul style="list-style-type: none"> <li>• Wheeler graphs [16] (e.g. de Bruijn graphs, NFAs for multiple strings)</li> </ul>	<u>Solvable in <math>\mathcal{O}( E m)</math> time</u> <ul style="list-style-type: none"> <li>• DAGs: Substitutions/Edits to graph [29]</li> <li>• General graphs: Substitutions/Edits to pattern [6]</li> <li>• de Bruijn graphs: Substitutions to pattern -No strongly Sub-<math>\mathcal{O}( E m)</math> alg. (this paper)</li> </ul>
Hard	<u>NO strongly sub-<math>\mathcal{O}( E m)</math> Alg.</u> <ul style="list-style-type: none"> <li>• General graphs [13, 19] (including DAGs with total degree <math>\leq 3</math>)</li> </ul>	<u>NP-Complete</u> <ul style="list-style-type: none"> <li>• General graphs: Substitutions/Edits to vertex labels [6, 26]</li> <li>• de Bruijn graphs: Substitutions to vertex labels (this paper)</li> </ul>

For general graphs, we can consider exact and approximate matching. For exact matching, conditional lower-bounds based on the Strong Exponential Time

Hypothesis (SETH), and other conjectures in circuit complexity, indicate that an  $\mathcal{O}(|E|m^{1-\varepsilon} + |E|^{1-\varepsilon}m)$  time algorithm with any constant  $\varepsilon > 0$ , for a graph with  $|E|$  edges and a pattern of length  $m$ , is highly unlikely (as is the ability to shave more than a constant number of logarithmic factors from the  $\mathcal{O}(|E|m)$  time complexity) [13, 19]. These results hold for even very restricted types of graphs, for example, DAGs with maximum total degree three and binary alphabets. For approximate matching, when edits are only allowed in the pattern, the problem is solvable in  $\mathcal{O}(|E|m)$  time [6]. If edits are also permitted in the graph, but the graph is a DAG, matching can be done in the same time complexity [29]. However, the problem becomes NP-complete when edits are allowed in arbitrary cyclic graphs. This was originally proven in [6] for large alphabets and more recently proven for binary alphabets in [26]. These results hold even when edits are restricted to only substitutions. The distinction between modifications to the graph and modifications to the pattern is important as these two problems are fundamentally different. When changes are made to cyclic graphs the same modification can be encountered multiple times while matching a pattern with no additional cost (see Section 3.1 in [26] for a detailed discussion). Furthermore, algorithmic solutions appearing in [29, 36, 42] are for the case where modifications are performed only to the pattern.

De Bruijn graphs are interesting from a theoretical perspective. Many graphs allow for extending Burrows-Wheeler Transformation (BWT) based techniques for efficient pattern matching. Sufficient conditions for doing this are captured by the definition of Wheeler graphs, introduced in [16], and further studied in [3, 4, 12, 15, 20]. De Bruijn graphs are themselves Wheeler graphs, hence on a de Bruijn graph exact pattern matching is solvable in linear time. However, the complexity of approximate matching in de Bruijn graphs when permitting modifications to the graph or modifications to the pattern remained open [26].

We make two important contributions (see Table 1). First, we prove that for de Bruijn graphs, despite exact matching being solvable in linear time, the approximate matching problem with vertex label substitutions is NP-complete. Second, we prove that a strongly subquadratic time algorithm for the approximate pattern matching problem on de Bruijn graphs, where substitutions are only allowed in the pattern, is not possible under SETH. This confirms the optimality of the known quadratic time algorithms when considering polynomial factors. To the best of our knowledge, these are the first such results for any type of Wheeler graph. Note that pattern-to-text matching (under substitutions) can be solved in sub-quadratic  $\tilde{\mathcal{O}}(n\sqrt{m})$  time, where  $n$  is the text's length [2].

## 1.1 Technical Background and Our Results

**Notation for Edges:** For a directed edge from a vertex  $u$  to a vertex  $v$  we will use the notation  $(u, v)$ . Additionally, we will refer to  $u$  as the *tail* of  $(u, v)$ , and  $v$  as the *head* of  $(u, v)$ .

**Walks Versus Paths:** A distinction must be made between the concept of a *walk* and a *path* in a graph. A walk is a sequence of vertices  $v_1, v_2, \dots, v_t$  such

that for each  $i \in [1, t - 1]$ ,  $(v_i, v_{i+1}) \in E$ . Vertices can be repeated in a walk. A path is a walk where vertices are not repeated. The length of a walk is defined as the number of edges in the walk,  $t - 1$ , or equivalently one less than the number of vertices in the sequence (counted with multiplicity). This work will be concerning the existence of walks.

**Induced Subgraphs:** An induced subgraph of a graph  $G = (V, E)$  consists of a subset of vertices  $V' \subseteq V$ , and all edges  $(u, v) \in E$  such that  $u, v \in V'$ . This is in contrast to an arbitrary subgraph of  $G$ , where an edge can be omitted from the subgraph, even if both of its incident vertices are included.

**de Bruijn Graphs:** An *order- $k$*  full de Bruijn graph is a compact representation of all  $k$ -mers (strings of length  $k$ ) from an alphabet  $\Sigma$  of size  $\sigma$ . It consists of  $\sigma^k$  vertices, each corresponding to a unique  $k$ -mer (which we call as its *implicit label*) in  $\Sigma^k$ . There is a directed edge from each vertex with implicit label  $s_1s_2\dots s_k \in \Sigma^k$  to the  $\sigma$  vertices with implicit labels  $s_2s_3\dots s_k\alpha$ ,  $\alpha \in \Sigma$ . We will work with induced subgraphs of full de Bruijn graphs in this paper. We assign to every vertex  $v$  a label  $L(v) \in \Sigma$ , such that the implicit label of  $v$  is  $L(u_1)L(u_2)\dots L(u_{k-1})L(v)$  where  $u_1, u_2, \dots, u_{k-1}, v$  is any length  $k - 1$  walk ending at  $v$ . This is equivalent to the notion of a de Bruijn graph constructed from  $k$ -mers commonly used in Computational Biology.

**Strings and Matching:** For a string  $S$  of length  $n$  indexed from 1 to  $n$ , we use  $S[i]$  to denote the  $i^{\text{th}}$  symbol in  $S$ . We use  $S[i, j]$  to denote the substring  $S[i]S[i+1]\dots S[j]$ . If  $j < i$ , then we take  $S[i, j]$  as the empty string. As mentioned above, we will consider every vertex  $v$  as labeled with a single symbol  $L(v) \in \Sigma$ . A pattern  $P[1, m]$  matches a walk  $v_1, v_2, \dots, v_m$  iff  $P[i] = L(v_i)$  for every  $i \in [1, m]$ .

With these definitions in hand, we can formally define our first problem.

*Problem 1 (Approximate matching with vertex label substitutions).* Given a vertex labeled graph  $D = (V, E)$  with alphabet  $\Sigma$  of size  $\sigma$ , pattern  $P[1, m]$ , and integer  $\delta \geq 0$ , determine if there exists a walk in  $D$  matching  $P$  after at most  $\delta$  substitutions to the vertex labels.

**Theorem 1.** *Problem 1 is NP-complete on de Bruijn graphs with  $\sigma = 4$ .*

Theorem 1 is proven in Sect. 2. Intuitively, our reduction transforms a general directed graph into a de Bruijn that maintains key topological properties related to the existence of walks. The distinct problem of approximately matching a pattern to a *path* in a de Bruijn graph was shown to be NP-complete in [31]. As mentioned by the authors of that work, the techniques used there do not appear to be easily adaptable to the problem for walks. Our approach uses edge transformations more closely inspired by those used in [28] for proving hardness on the paired de Bruijn sound cycle problem.

*Problem 2 (Approximate matching with substitutions within the pattern).* Given a vertex labeled graph  $D = (V, E)$  with alphabet  $\Sigma$  of size  $\sigma$ , pattern  $P[1, m]$ , and integer  $\delta \geq 0$ , determine if there exists a walk in  $D$  matching  $P$  after at most  $\delta$  substitutions to the symbols in  $P$ .

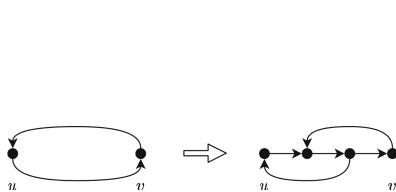
For Problem 2 we provide a hardness result based on SETH, which is frequently used for establishing conditional optimality of polynomial time algorithms [1, 7, 13, 18, 19, 25]. We refer the reader to [44] for the definition of SETH and for the reduction to the Orthogonal Vectors problem (OV), which is utilized to prove Theorem 2.

**Theorem 2.** *Conditioned on SETH, for all constants  $\varepsilon > 0$ , there does not exist an  $\mathcal{O}(|E|m^{1-\varepsilon} + |E|^{1-\varepsilon}m)$  time algorithm for Problem 2 on de Bruijn graphs with  $\sigma = 4$ .*

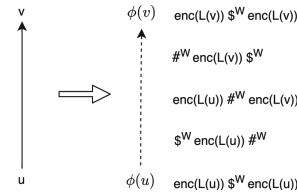
Note that the order of the de Bruijn graphs used in ours proofs are  $\Theta(\log^2 |V|)$  for Theorem 1 and  $\Theta(\log |V|)$  for Theorem 2.

## 2 NP-Completeness of Problem 1 on de Bruijn Graphs

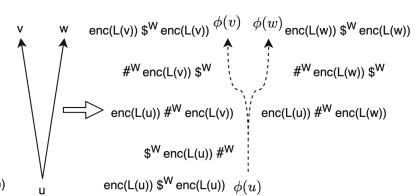
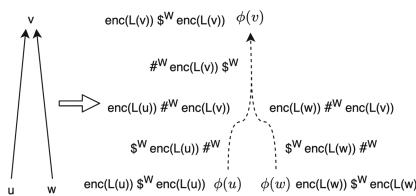
Our proof of NP-completeness uses a reduction from the Hamiltonian Cycle Problem on directed graphs, which is the problem of deciding if there exists a cycle through a directed graph that visits every vertex exactly once. It was proven NP-complete even when restricted to directed graphs where the number of edges is linear in the number of vertices [41]. To present the reduction, we introduce the concept of *merging* two vertices. To merge vertices  $u$  and  $v$ , we create a new vertex  $w$ . We then take all edges with either  $u$  or  $v$  as their head and make  $w$  their new head. Next, we take all edges with either  $u$  or  $v$  as their tail and make  $w$  their new tail. This makes the edges  $(u, v)$  and  $(v, u)$  (if they existed) into self-loops for  $w$ . If two self-loops are formed, we delete one of them. Finally, we delete the original vertices  $u$  and  $v$ .



**Fig. 1.** Gadget to remove cycles of length 2 from the initial input graph.



**Fig. 2.** The transformation from edges to paths used in our reduction.



**Fig. 3.** Vertices with the same implicit label are merged while transforming  $D$  to  $D'$ , causing edges with shared vertices to become paths with shared vertices.

## 2.1 Reduction

We start with an instance of the Hamiltonian cycle problem on a directed graph where the number of edges is linear in the number of vertices. We can assume there are no self-loops or vertices with in-degree or out-degree zero. To simplify the proof, we first eliminate any cycles of length 2 using the gadget in Fig. 1. We denote the resulting graph as  $D = (V, E)$  and let  $n = |V|$ . We assign each vertex  $v \in V$  a unique integer  $L(v) \in [0, n - 1]$ . Let  $\ell = \lceil \log n \rceil$ ,  $\text{bin}(i)$  be the standard binary encoding of  $i$  using  $\ell$  bits and  $\Sigma = \{\$\#, 0, 1\}$ . Define  $\text{enc}(i) = (0^{\ell-1}1)^{2\ell}\text{bin}(i)$ ,  $W = |\text{enc}(i)|$ , and  $k = 3W$ .

We construct a new (de Bruijn) graph  $D' = (V', E')$  as follows: Initially  $D'$  is the empty graph. For  $i = 0, 1, \dots, n - 1$ , for each edge  $(u, v) \in E$  where  $L(v) = i$ , create a new path whose concatenation of vertex labels is  $\#\text{enc}(i)\$^W\text{enc}(i)$ . The vertex  $u$  will correspond with a new vertex  $\phi(u)$  at the start of this path, and the vertex  $v$  will correspond with a new vertex  $\phi(v)$  at the end of this path. The vertex  $\phi(v)$  has the implicit label  $\text{enc}(L(v))\$^W\text{enc}(L(v))$ . The vertex  $\phi(u)$  is temporarily assigned the implicit label  $\text{enc}(L(u))\$^W\text{enc}(L(u))$ . See Fig. 2. We call vertices with implicit labels of the form  $\text{enc}(L(\cdot))\$^W\text{enc}(L(\cdot))$  *marked vertices*. We use the notation  $\phi((u, v))$  to denote the path created when applying this transformation to  $(u, v) \in E$ . After the path  $\phi((u, v))$  is created, vertices in  $V'$  having the same implicit label are merged, and parallel edges are deleted (Fig. 3). See Fig. 4 for a complete example. Finally, let  $\delta = 2\ell(n - 1)$  and

$$P = \#\text{enc}(0)\$^W\text{enc}(0)\#\text{enc}(1)\$^W\text{enc}(1)\#\text{enc}(2)\$^W\text{enc}(2)\dots$$

$$\#\text{enc}(n - 1)\$^W\text{enc}(n - 1)\#\text{enc}(0)\$^W\text{enc}(0).$$

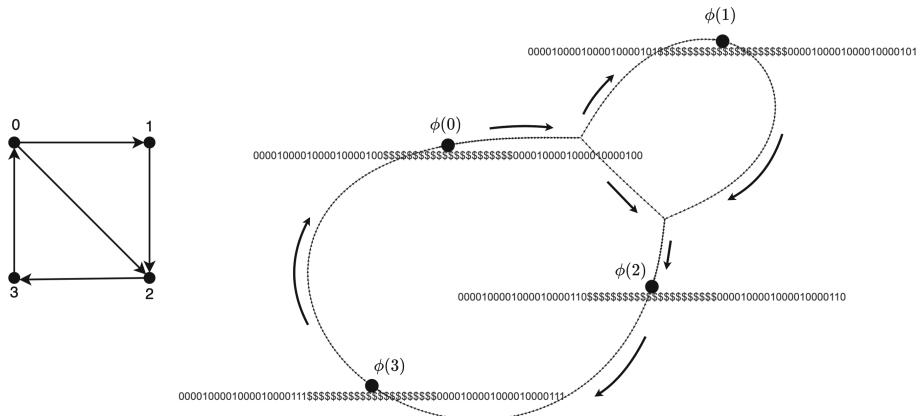
We will show that there exists a walk in  $D'$  matching  $P$  with at most  $\delta$  vertex label substitutions iff  $D$  contains a Hamiltonian cycle.

## Proof of Correctness

**Lemma 1.** *The graph  $D'$  constructed as above is a de Bruijn graph.*

*Proof (Overview).* Three properties must be proven: (i) Implicit labels are unique, meaning for every implicit label at most one vertex is assigned that label; (ii) No edges are missing, i.e., if the implicit label of  $y \in V'$  is  $S\alpha$  for some string  $S[1, k - 1]$  and symbol  $\alpha \in \Sigma$ , and there exists a vertex  $x \in V'$  with implicit label  $\beta S[1, k - 1]$  for some symbol  $\beta \in \Sigma$ , then  $(x, y) \in E'$ ; (iii) Implicit labels are well-defined, in that every walk of length  $k - 1$  ending at a vertex  $x \in V'$  matches the same string (the implicit label of  $x$ ); The most involved of these is proving property (ii), which requires analyzing several cases. The complete proof is given in the full version [21].  $\square$

The correctness of the reduction remains to be shown. Lemmas 2–4 establish useful structural properties of  $D'$ , Lemma 5 proves that the existence of a Hamiltonian Cycle in  $D$  implies an approximate matching in  $D'$ , and Lemmas 6–9 demonstrate the converse.



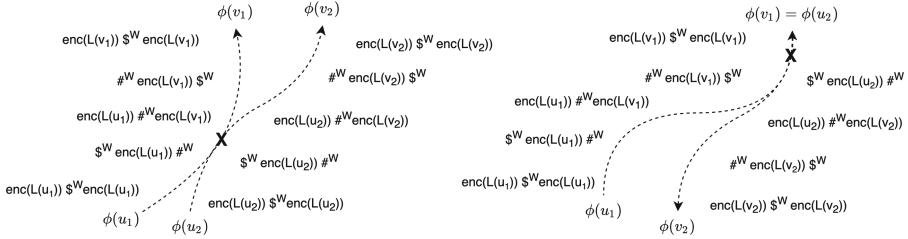
**Fig. 4.** (Left) A graph before the reduction is applied to it. (Right) The transformed graph. Implicit labels for marked vertices are shown and the path directions are annotated by arrows beside each path.

**Lemma 2.** *Any walk between two marked vertices  $\phi(u)$  and  $\phi(v)$  containing no additional marked vertices has length  $4W$ . Hence, we can conclude any such walk is a path.*

*Proof (Overview).* This is proven using induction on the number of edges transformed. It is shown that for every vertex, a key property regarding the distances to its closest marked vertices continues to hold after vertices on any newly created path are merged. We defer the complete proof to the full version [21].  $\square$

**Lemma 3.** *For  $(u_1, v_1), (u_2, v_2) \in E$ , unless  $u_1 = u_2$  or  $v_1 = v_2$ ,  $\phi((u_1, v_1))$  and  $\phi((u_2, v_2))$  share no vertices.*

*Proof.* In the case where  $\{u_1, v_1\} \cap \{u_2, v_2\} = \emptyset$  (Fig. 5 left), every implicit vertex label in  $\phi((u_1, v_1))$  contains  $\text{enc}(L(u_1))$  or  $\text{enc}(L(v_1))$  (or both), and contains neither  $\text{enc}(L(u_2))$  nor  $\text{enc}(L(v_2))$ . Similarly, every implicit vertex label in  $\phi((u_2, v_2))$  contains  $\text{enc}(L(u_2))$  or  $\text{enc}(L(v_2))$  (or both) and contains neither  $\text{enc}(L(u_1))$  nor  $\text{enc}(L(v_1))$ . This implies that none of the implicit labels match between the two paths, thus no vertices are merged. In the case where  $v_1 = u_2$  and  $u_1 \neq v_2$  (Fig. 5, right), the implicit labels of vertices  $\phi((u_1, v_1))$  not containing  $\text{enc}(L(u_1))$  have  $\#$  symbols in different positions than implicit labels of vertices in  $\phi((u_2, v_2))$  not containing  $\text{enc}(L(v_2))$ , and, since  $v_1 \neq v_2$ , cannot match the implicit labels of vertices in  $\phi((u_2, v_2))$  containing  $\text{enc}(L(v_2))$ . Vertices in  $\phi((u_1, v_1))$  with implicit labels containing  $\text{enc}(L(u_1))$  have  $\#$  symbols in different positions than implicit labels of vertices in  $\phi((u_2, v_2))$  not containing  $\text{enc}(L(u_2))$ , and, since  $u_1 \neq u_2$ , cannot match the implicit labels of vertices in  $\phi((u_2, v_2))$  containing  $\text{enc}(L(u_2))$ . The case  $u_1 = v_2$  and  $u_2 \neq v_1$  is symmetric. The case  $u_1 = v_2$  and  $v_1 = u_2$  cannot happen since, by the use of our gadget in Fig. 1,  $D$  cannot contain the edges  $(u_1, v_1)$  and  $(v_1, u_1)$ .  $\square$



**Fig. 5.** Examples where paths between marked vertex cannot share any vertex: (Left) The case where  $\{u_1, v_1\} \cap \{u_2, v_2\} = \emptyset$ . (Right) The case where  $v_1 = u_2$  and  $u_1 \neq v_2$ .

**Lemma 4.** *There exists a path from a marked vertex  $\phi(u) \in V'$  to a marked vertex  $\phi(v) \in V'$  containing no other marked vertices iff  $(u, v) \in E$ .*

*Proof (Overview).* It is clear from construction that if  $(u, v) \in D$ , then such a path exists in  $D'$ . In the other direction, we utilize Lemmas 2 and 3 to show that such a path existing without a corresponding edge would create a contradiction. The complete proof is provided in the full version [21].  $\square$

**Lemma 5.** *If  $D$  has a Hamiltonian cycle, then  $P$  can be matched in  $D'$  with at most  $\delta$  substitutions to vertex labels of  $D'$ .*

*Proof.* To obtain a matching walk, follow the cycle corresponding to a solution in  $D$  starting with the marked vertex in  $V'$  corresponding to the vertex in  $V$  with label 0. By Lemma 4, each edge traversed in  $D$  corresponds to a path in  $D'$ . While traversing these paths, modify the vertex labels in  $D'$  corresponding to the substrings  $\text{bin}(i)$  to match  $P$ . Assuming no conflicting substitutions are needed, this requires at most  $2\ell(n-1)$  substitutions.

It remains to be shown that no conflicting label substitutions will be necessary. Consider the edges  $(u_1, v_1), (u_2, v_2) \in E$  used in the Hamiltonian cycle in  $D$ . We will never have  $u_1 = u_2$  or  $v_1 = v_2$ . Hence, by Lemma 3, the sets of vertices on the paths  $\phi((u_1, v_1))$  and  $\phi((u_2, v_2))$  are disjoint.  $\square$

**Lemma 6.** *If  $P$  can be matched in  $D'$  with at most  $\delta$  substitutions to vertex labels of  $D'$ , then all  $\$$ 's in  $P$  are matched with non-substituted  $\$$ 's in  $D'$  and all  $\#$ 's in  $P$  are matched with non-substituted  $\#$ 's in  $D'$ . Consequently, we can assume the only substitutions are to the vertex labels corresponding to  $\text{bin}(i)$ 's within  $\text{enc}(i)$ 's.*

*Proof (Overview).* We establish the existence of a long, non-branching path for every marked vertex that can be traversed at most once when matching  $P$ . This, combined with maximal paths of,  $\$$ ,  $\#$ , and 0/1-symbols, all being of length  $W$ , makes it so that ‘shifting’  $P$  to match a portion of  $D$  forces the shift to occur throughout the walk traversed while matching  $P$ . Utilizing the large Hamming distance between shifted instances of two encodings, we can then show that not matching all non-0/1 symbols requires more than  $\delta$  substitutions. The complete proof is provided in the full version [21].  $\square$

Post-substitution to vertex labels, we will refer to a vertex as marked if there exists a walk ending at it that matches a string of the form  $\text{enc}(L(u))\$^W \text{enc}(L(u))$ ,  $u \in V$ . Note that this definition does not require all length  $k-1$  walks ending at such a vertex to match the same string.

**Lemma 7.** *If  $P$  can be matched in  $D'$  with at most  $\delta$  substitutions to vertex labels of  $D'$ , then no additional marked vertices are created due to vertex substitutions.*

*Proof.* Pre-substitution, only marked vertices have implicit labels of the form  $S_1\$^WS_2$  where  $S_1$  and  $S_2$  contain no  $\$$  symbols. Hence, the only way that a vertex could have a walk ending at it that matches a pattern of that form post-substitution is if either it was originally a marked vertex, or some non-0/1-symbols were substituted in  $D'$ . However, by Lemma 6 the latter case cannot happen, and only originally marked vertices have walks ending at them matching strings of the form  $S_1\$^WS_2$  post-substitution.  $\square$

**Lemma 8.** *If  $P$  can be matched in  $D'$  with at most  $\delta$  substitutions to vertex labels of  $D'$ , then each originally marked vertex in  $D'$  is visited exactly once, except for an originally marked vertex at the end of a path matching  $\text{enc}(0)\$^W \text{enc}(0)$  that is visited twice.*

*Proof.* First, we show that all marked vertices, except the one with implicit label  $\text{enc}(0)\$^W \text{enc}(0)$ , are visited at most once. Pre-substitution, a marked vertex with implicit label  $\text{enc}(i)\$^W \text{enc}(i)$  is at the end of a unique, branchless path of length  $W$  matching  $\text{enc}(i)$ . By Lemma 6, the only substitutions to this path made while matching  $P$  are substitutions making it match  $\text{enc}(i')$ ,  $i' \neq i$ . If this path were modified to match  $\text{enc}(i')$ ,  $i' > 0$ , then the only way the marked vertex could be visited twice while matching  $P$  is if after traversing the path, another path matching  $\$^W$  is taken back to the start of this  $\text{enc}(i')$  path. However, any edges leaving this marked vertex are labeled with  $\#$ , making this impossible. By similar reasoning, the path matching  $\text{enc}(0)$  ending at a marked vertex is visited at most twice. We now show that each marked vertex is visited at least once. Suppose some marked vertex is not visited. By Lemma 7, no additional marked vertices are created. Hence, a marked vertex ending a path matching  $\text{enc}(i)$ ,  $i > 0$  is visited at least twice, or a marked vertex ending a path matching  $\text{enc}(0)$  is visited at least three times, a contradiction.  $\square$

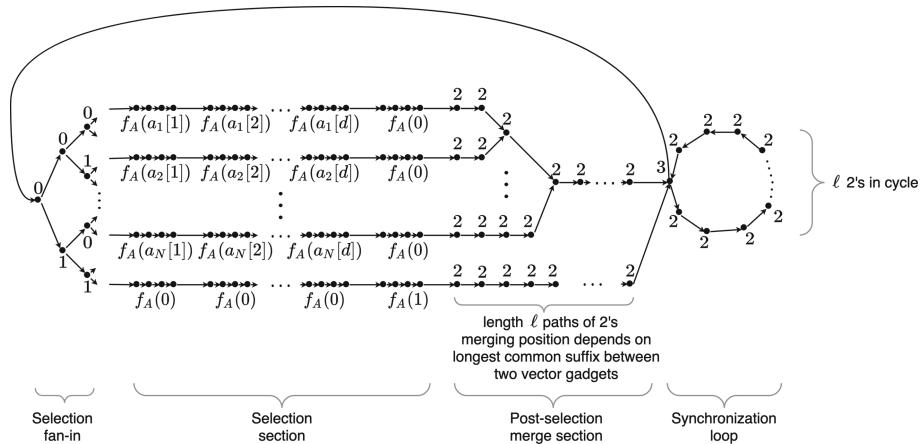
**Lemma 9.** *If  $P$  can be matched in  $D'$  with at most  $\delta$  substitutions to vertex labels of  $D'$ , then  $D$  has a Hamiltonian cycle.*

*Proof.* By Lemma 4, the paths between marked vertices traversed while matching with  $P$  correspond to edges between vertices in  $D$ . Combined with marked vertices being visited exactly once from Lemma 8 (except the marked vertex ending a path matching  $\text{enc}(0)$ ), the walk matched by  $P$  in  $D'$  corresponds to a Hamiltonian cycle through  $D$  beginning and ending at the vertex labeled 0.  $\square$

This completes the proof of Theorem 1. To see that  $k = \Theta(\log^2 |V'|)$ , first recall that  $|V|$  is the number of vertices in the original graph, where we assumed  $|E| = \mathcal{O}(|V|)$ . At most  $4W|E| = \mathcal{O}(k|V|)$  vertices are created in the reduction. Also, the proof of Lemma 6 establishes that there is a unique set of at least  $\Theta(k)$  vertices for every marked vertex, each one corresponding to a vertex in the original graph. Combining, we have that  $|V'| = \Theta(k|V|)$ . By construction,  $k = \Theta(\log^2 |V|)$ , and since  $|V'| = \Theta(k|V|)$ ,  $k = \Theta(\log^2 |V'|)$  as well.

### 3 Hardness for Problem 2 on de Bruijn Graphs

**Reduction.** The Orthogonal Vectors Problem is defined as follows: given two sets of binary vectors  $A, B \subseteq \{0, 1\}^d$  where  $|A| = |B| = N$ , determine whether there exists vectors  $a \in A$  and  $b \in B$  such that their inner product is zero. Conditioned on SETH, a standard reduction shows that this cannot be solved in time  $d^{\Theta(1)}N^{2-\varepsilon}$  for any constant  $\varepsilon > 0$  [44].



**Fig. 6.** An illustration of the reduction from OV to Problem 2.

Let the given instance of OV consist of  $A, B \subseteq \{0, 1\}^d$  where  $|A| = |B| = N = 2^m$  for some natural number  $m$ . Hence, we have  $\lceil \log(N+1) \rceil = \log N + 1$ . This will ease computation later. We also assume that  $d > \log N$ . This is reasonable, as if  $d \leq \log N$ , then  $|A|$  and  $|B|$  would contain either all vectors of length  $d$  or repetitions.

We will next provide a formal description of the graph  $D$  our reduction creates from the set  $A = \{a_1, a_2, \dots, a_N\}$  and the pattern  $P$  it creates from the set  $B = \{b_1, b_2, \dots, b_N\}$ . The reader may find Fig. 6 helpful. The graph will consist of four sections. We name these according to their function in the reduction: the Selection fan-in, the Selection section, the Post-Selection merge section, and the Synchronization loop.

We start with the Selection fan-in. Let  $2^c$  be the smallest power of 2 such that  $2^c \geq N + 1$ . The Selection fan-in consists of a complete binary tree with  $2^c$  leaves where all paths are directed away from the root. The root is labeled 0 and the children of every node are labeled 0 and 1, respectively.

The Selection section consists of  $N + 1$  paths. We first define the mappings  $f_A$  and  $f_B$  from  $\{0, 1\}$  to sequences of length four as  $f_A(0) = 1100$ ,  $f_A(1) = 1111$ ,  $f_B(0) = 0110$ ,  $f_B(1) = 0000$ . These mappings have the property that  $d_H(f_A(0), f_B(0)) = d_H(f_A(0), f_B(1)) = d_H(f_A(1), f_B(0)) = 2$  and  $d_H(f_A(1), f_B(1)) = 4$ , where  $d_H(x, y)$  is the Hamming distance between strings  $x$  and  $y$ . We make the  $i^{th}$  path for  $1 \leq i \leq N$  a path of  $4(d + 1)$  vertices with labels matching the string  $f_A(a_i[1])f_A(a_i[2])\dots f_A(a_i[d])f_A(0)$ . We make the  $(N + 1)^{th}$  path have  $4(d + 1)$  vertices and match the string  $f_A(0)^d f_A(1)$ . Let  $s_i$  denote the start vertex of path  $i$ . We arbitrarily choose  $N + 1$  leaves,  $l_1, l_2, \dots, l_{N+1}$ , from the Selection fan-in and add the edges  $(l_i, s_i)$  for  $1 \leq i \leq N + 1$ .

We define the implicit label size as  $k = \lceil \log(N + 1) \rceil + 4(d + 1)$  and  $\ell = k - 1$ . To construct the Post-selection merge section, we start with  $N + 1$  length  $\ell - 1$  paths, each matching the string  $2^\ell$ . For every path in the Selection section, we add an edge from the last vertex in the path to one of the paths matching  $2^\ell$ . This is done so that every path matching  $2^\ell$  in the Post-selection merge section is connected to exactly one path from the Selection section. Next, we merge two vertices if they have the same implicit label. This is repeated until all vertices in the Post-selection merge section have a unique implicit label.

To construct the Synchronization loop we create a directed cycle with  $\ell + 1 = k$  vertices. One of these is labeled with the symbol 3, and the rest with the symbol 2. Edges from each ending vertex in the Post-selection Merge section to the vertex labeled 3 are then added. A final edge from the vertex labeled 3 to the root of the binary tree in the Selection fan-in completes the graph, which we denote as  $D$ .

Let  $t = 5d + \lceil \log(N + 1) \rceil$ . To complete the reduction, we make the pattern

$$\begin{aligned} P = & (2^\ell 3)^t 2^{\lceil \log(N + 1) \rceil} f_B(b_1[1])f_B(b_1[2])\dots f_B(b_1[d])f_B(1) \\ & (2^\ell 3)^t 2^{\lceil \log(N + 1) \rceil} f_B(b_2[1])f_B(b_2[2])\dots f_B(b_2[d])f_B(1) \\ & \dots \\ & (2^\ell 3)^t 2^{\lceil \log(N + 1) \rceil} f_B(b_N[1])f_B(b_N[2])\dots f_B(b_N[d])f_B(1) \end{aligned}$$

and the maximum number of allowed substitutions  $\delta = N \lceil \log_2(N + 1) \rceil + 2(d + 1) + (2d + 4)(N - 1)$ .

We call substrings in  $P$  of the form  $f_B(b_i[1])f_B(b_i[2])\dots f_B(b_i[d])f_B(1)$  and paths in  $D$  matching strings of the form  $f_A(a_i[1])f_A(a_i[2])\dots f_A(a_i[d])f_A(0)$  vector gadgets. Note that  $|E| = \mathcal{O}(dN)$  and  $m = |P| = \mathcal{O}(d^2N)$ . Hence, an algorithm for approximate matching running in time  $\mathcal{O}(m|E|^{1-\varepsilon} + m^{1-\varepsilon}|E|)$  for some  $\varepsilon > 0$  would imply an algorithm for OV running in time  $d^{\Theta(1)}N^{2-\varepsilon}$ . This implies that once the correctness of the reduction has been established, Theorem 2 follows.

### 3.1 Proof of Correctness

Proofs of Lemma 10 and Lemma 11 are provided in the full version [21].

**Lemma 10.** *The graph  $D$  is a de Bruijn graph.*

**Lemma 11.** *In an optimal solution, 3's in  $P$  are matched with 3's in  $D$ .*

**Lemma 12.** *In an optimal solution, vector gadgets in  $P$  are matched with vector gadgets in  $D$ .*

*Proof.* Suppose otherwise. By Lemma 11, this can only occur if some vector gadget in  $P$  is matched against the Synchronization loop. This requires at least  $4(d+1)$  substitutions. We can instead match the  $\lceil \log(N+1) \rceil$  2's preceding the vector gadget in  $P$  with the Selection fan-in and the vector gadget in  $P$  with the  $(N+1)^{th}$  path in the Selection section. Due to  $d_H(f_A(0), f_B(0)) = d_H(f_A(0), f_B(1)) = 2$  and  $d(f_A(1), f_B(1)) = 4$ , this requires  $\lceil \log(N+1) \rceil + 2d + 4$  substitutions in  $P$ . Since,  $\log N < d < 2d$  we have  $\log N < 2d - 1$ . Using that  $N$  is some power of 2,  $\lceil \log(N+1) \rceil + 2d + 4 = \log N + 1 + 2d + 4 < 4d + 4$ . Hence, the cost decreases by matching the vector gadget in  $P$  to a vector gadget in  $D$  instead.  $\square$

**Lemma 13.** *If there exists a vector  $a \in A$  and  $b \in B$  such that  $a \cdot b = 0$ , then  $P$  can be matched to  $D$  with at most  $\delta$  substitutions.*

*Proof.* Match the vector gadget for  $b$  in  $P$  with the vector gadget for  $a$  in the Selection section of  $D$ . This costs  $2(d+1)$  substitutions. Match the remaining  $N-1$  vector gadgets in  $P$  with the  $(N+1)^{th}$  path in the Selection section, requiring  $(2d+4)(N-1)$  substitutions in total. The total number of substitutions of 2's in  $P$  to match the Selection fan-in is  $N\lceil \log(N+1) \rceil$ . Adding these, the total number of substitutions is exactly  $\delta$ . The synchronization loop can be used for matching all additional symbols in  $P$  without any further substitutions.  $\square$

**Lemma 14.** *If  $P$  can be matched in  $D$  with at most  $\delta$  substitutions, then there exists vectors  $a \in A$  and  $b \in B$  such  $a \cdot b = 0$ .*

*Proof.* By Lemma 12, we can assume vector gadgets in  $P$  are only matched against vector gadgets in  $D$ . Suppose that there does not exist a pair of orthogonal vectors  $a \in A$  and  $b \in B$ . Then, whichever vector gadget in  $D$  we choose to match a vector gadget in  $P$  to, matching the vector gadget requires at least  $2d+4$  substitutions. Hence, the total cost is at least  $(2d+4)N + N\lceil \log(N+1) \rceil > \delta$ , proving the contrapositive of Lemma 14.  $\square$

## 4 Discussion

We leave open several interesting problems. An NP-completeness proof for Problem 1 on de Bruijn graphs when  $k = \mathcal{O}(\log n)$  and the alphabet size is constant is

still needed. Additionally, we need to extend these hardness results to when substitutions are allowed in both the graph and the pattern, and when insertions and deletions in some form are allowed in the graph and (or) the pattern. It seems unlikely that adding more types of edit operations would make the problems computationally easier, and we conjecture these variants are NP-complete on de Bruijn graphs as well. It also needs to be determined whether Problem 1 is NP-complete on de Bruijn graphs with binary alphabets, or whether the SETH-based hardness results hold for Problem 2 on binary alphabets. A practical question is whether these problems are hard for small  $\delta$  values on de Bruijn graphs (the problem for general graphs was proven to  $W[2]$  hard in terms of  $\delta$  in [11]). In applications, the allowed error thresholds are quite small. Clearly, the problems are slice-wise-polynomial with respect to  $\delta$ , i.e., for a constant  $\delta$  it is solvable in polynomial time via brute force, but are they fixed-parameter-tractable in  $\delta$ ? The reduction presented here (as well as the reductions in [6, 26]) is based on the Hamiltonian cycle problem, where a large  $\delta$  value is used. This makes the existence of such a fixed-parameter-tractable algorithm a distinct possibility.

**Acknowledgement.** This research is supported in part by the U.S. National Science Foundation (NSF) grants CCF-1704552, CCF-1816027, CCF-2112643, and CCF-2146003.

## References

1. Abboud, A., Backurs, A., Hansen, T.D., Williams, V.V., Zamir, O.: Subtree isomorphism revisited. *ACM Trans. Algorithms* **14**(3), 27:1–27:23 (2018). <https://doi.org/10.1145/3093239>
2. Abrahamson, K.R.: Generalized string matching. *SIAM J. Comput.* **16**(6), 1039–1051 (1987). <https://doi.org/10.1137/0216067>
3. Alanko, J., D'Agostino, G., Policriti, A., Prezza, N.: Wheeler languages. *CoRR* abs/2002.10303 (2020). <https://arxiv.org/abs/2002.10303>
4. Alanko, J.N., Gagie, T., Navarro, G., Benkner, L.S.: Tunneling on wheeler graphs. In: Data Compression Conference, DCC 2019, Snowbird, UT, USA, -26–29 March 2019. pp. 122–131 (2019). <https://doi.org/10.1109/DCC.2019.00020>
5. Almodaresi, F., Sarkar, H., Srivastava, A., Patro, R.: A space and time-efficient index for the compacted colored de Bruijn graph. *Bioinform* **34**(13), i169–i177 (2018). <https://doi.org/10.1093/bioinformatics/bty292>
6. Amir, A., Lewenstein, M., Lewenstein, N.: Pattern matching in hypertext. *J. Algorithms* **35**(1), 82–99 (2000). <https://doi.org/10.1006/jagm.1999.1063>
7. Backurs, A., Indyk, P.: Which regular expression patterns are hard to match? In: IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9–11 October 2016, pp. 457–466. Hyatt Regency, New Brunswick (2016). <https://doi.org/10.1109/FOCS.2016.56>
8. Benoit, G., et al.: Reference-free compression of high throughput sequencing data with a probabilistic de Bruijn graph. *BMC Bioinform* **16**, 288:1–288:14 (2015). <https://doi.org/10.1186/s12859-015-0709-7>

9. Chikhi, R., Limasset, A., Jackman, S., Simpson, J.T., Medvedev, P.: On the representation of de Bruijn graphs. *J. Comput. Biol.* **22**(5), 336–352 (2015). <https://doi.org/10.1089/cmb.2014.0160>
10. Chikhi, R., Rizk, G.: Space-efficient and exact de Bruijn graph representation based on a bloom filter. *Algorithms Mol. Biol.* **8**, 22 (2013). <https://doi.org/10.1186/1748-7188-8-22>
11. Dondi, R., Mauri, G., Zoppis, I.: Complexity issues of string to graph approximate matching. In: Leporati, A., Martín-Vide, C., Shapira, D., Zandron, C. (eds.) LATA 2020. LNCS, vol. 12038, pp. 248–259. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-40608-0\\_17](https://doi.org/10.1007/978-3-030-40608-0_17)
12. Egidi, L., Louza, F.A., Manzini, G.: Space efficient merging of de Bruijn graphs and wheeler graphs. *CoRR* abs/2009.03675 (2020). <https://arxiv.org/abs/2009.03675>
13. Equi, M., Grossi, R., Mäkinen, V., Tomescu, A.I.: On the complexity of string matching for graphs. In: 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, 9–12 July 2019, Patras, Greece. pp. 55:1–55:15 (2019). <https://doi.org/10.4230/LIPIcs.ICALP.2019.55>
14. Flick, P., Jain, C., Pan, T., Aluru, S.: Reprint of “a parallel connectivity algorithm for de Bruijn graphs in metagenomic applications”. *Parallel Comput.* **70**, 54–65 (2017). <https://doi.org/10.1016/j.parco.2017.09.002>
15. Gagie, T.:  $r$ -indexing wheeler graphs. *CoRR* abs/2101.12341 (2021). <https://arxiv.org/abs/2101.12341>
16. Gagie, T., Manzini, G., Sirén, J.: Wheeler graphs: a framework for BWT-based data structures. *Theor. Comput. Sci.* **698**, 67–78 (2017). <https://doi.org/10.1016/j.tcs.2017.06.016>
17. Georganas, E., Buluç, A., Chapman, J., Oliker, L., Rokhsar, D., Yelick, K.A.: Parallel de Bruijn graph construction and traversal for de novo genome assembly. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, New Orleans, LA, USA, -16–21 November 2014. pp. 437–448 (2014). <https://doi.org/10.1109/SC.2014.41>
18. Gibney, D.: An efficient elastic-degenerate text index? not likely. In: Boucher, C., Thankachan, S.V. (eds.) SPIRE 2020. LNCS, vol. 12303, pp. 76–88. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-59212-7\\_6](https://doi.org/10.1007/978-3-030-59212-7_6)
19. Gibney, D., Hoppenworth, G., Thankachan, S.V.: Simple reductions from formulas to pattern matching on labeled graphs and subtree isomorphism. In: 4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, 11–12 January 2021. pp. 232–242 (2021). <https://doi.org/10.1137/1.9781611976496.26>
20. Gibney, D., Thankachan, S.V.: On the hardness and inapproximability of recognizing wheeler graphs. In: 27th Annual European Symposium on Algorithms, ESA 2019, 9–11 September 2019, Munich/Garching, Germany. pp. 51:1–51:16 (2019). <https://doi.org/10.4230/LIPIcs.ESA.2019.51>
21. Gibney, D., Thankachan, S.V., Aluru, S.: The complexity of approximate pattern matching on de Bruijn graphs (2022)
22. Heydari, M., Miclotte, G., de Peer, Y.V., Fostier, J.: Browniealigner: accurate alignment of illumina sequencing data to de Bruijn graphs. *BMC Bioinform.* **19**(1), 311:1–311:10 (2018). <https://doi.org/10.1186/s12859-018-2319-7>
23. Holley, G., Peterlongo, P.: Blastgraph: intensive approximate pattern matching in string graphs and de-Bruijn graphs. In: PSC 2012 (2012)
24. Holley, G., Wittler, R., Stoye, J., Hach, F.: Dynamic alignment-free and reference-free read compression. *J. Comput. Biol.* **25**(7), 825–836 (2018), <https://doi.org/10.1089/cmb.2018.0068>

25. Hoppenworth, G., Bentley, J.W., Gibney, D., Thankachan, S.V.: The fine-grained complexity of median and center string problems under edit distance. In: 28th Annual European Symposium on Algorithms, ESA 2020, 7–9 September 2020, Pisa, Italy (Virtual Conference). pp. 61:1–61:19 (2020). <https://doi.org/10.4230/LIPIcs.ESA.2020.61>
26. Jain, C., Zhang, H., Gao, Yu., Aluru, S.: On the complexity of sequence to graph alignment. In: Cowen, L.J. (ed.) RECOMB 2019. LNCS, vol. 11467, pp. 85–100. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17083-7\\_6](https://doi.org/10.1007/978-3-030-17083-7_6)
27. Kamal, M.S., Parvin, S., Ashour, A.S., Shi, F., Dey, N.: De-Bruijn graph with MapReduce framework towards metagenomic data classification. *Int. J. Inf. Technol.* **9**(1), 59–75 (2017)
28. Kapun, E., Tsarev, F.: On NP-hardness of the paired de Bruijn sound cycle problem. In: Darling, A., Stoye, J. (eds.) WABI 2013. LNCS, vol. 8126, pp. 59–69. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40453-5\\_6](https://doi.org/10.1007/978-3-642-40453-5_6)
29. Kavya, V.N.S., Tayal, K., Srinivasan, R., Sivadasan, N.: Sequence alignment on directed graphs. *J. Comput. Biol.* **26**(1), 53–67 (2019). <https://doi.org/10.1089/cmb.2017.0264>
30. Li, D., Liu, C., Luo, R., Sadakane, K., Lam, T.W.: MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinformatics* **31**(10), 1674–1676 (2015). <https://doi.org/10.1093/bioinformatics/btv033>
31. Limasset, A., Cazaux, B., Rivals, E., Peterlongo, P.: Read mapping on de Bruijn graphs. *BMC Bioinform.* **17**, 237 (2016). <https://doi.org/10.1186/s12859-016-1103-9>
32. Limasset, A., Flot, J., Peterlongo, P.: Toward perfect reads: self-correction of short reads via mapping on de Bruijn graphs. *Bioinformatics* **36**(2), 651 (2020). <https://doi.org/10.1093/bioinformatics/btz548>
33. Lin, Y., Shen, M.W., Yuan, J., Chaisson, M., Pevzner, P.A.: Assembly of long error-prone reads using de Bruijn graphs. In: Proceedings of the Research in Computational Molecular Biology - 20th Annual Conference, RECOMB 2016, Santa Monica, CA, USA, 17–21 April 2016, p. 265 (2016). <https://link.springer.com/content/pdf/bbm%3A978-3-319-31957-5%2F1.pdf>
34. Liu, B., Guo, H., Brudno, M., Wang, Y.: deBGA: read alignment with de Bruijn graph-based seed and extension. *Bioinformatics* **32**(21), 3224–3232 (2016). <https://doi.org/10.1093/bioinformatics/btw371>
35. Morisse, P., Lecroq, T., Lefebvre, A.: Hybrid correction of highly noisy long reads using a variable-order de Bruijn graph. *Bioinformatics* **34**(24), 4213–4222 (2018). <https://doi.org/10.1093/bioinformatics/bty521>
36. Navarro, G.: Improved approximate pattern matching on hypertext. *Theor. Comput. Sci.* **237**(1–2), 455–463 (2000). [doi.org/10.1016/S0304-3975\(99\)00333-3](https://doi.org/10.1016/S0304-3975(99)00333-3)
37. Pell, J., Hintze, A., Canino-Koning, R., Howe, A., Tiedje, J.M., Brown, C.T.: Scaling metagenome sequence assembly with probabilistic de Bruijn graphs. *Proc. Natl. Acad. Sci. USA* **109**(33), 13272–13277 (2012). <https://doi.org/10.1073/pnas.1121464109>
38. Peng, Y., Leung, H.C.M., Yiu, S., Chin, F.Y.L.: IDBA - a practical iterative de Bruijn graph de novo assembler. In: Proceedings of the 14th Annual International Conference on Research in Computational Molecular Biology, RECOMB 2010, Lisbon, Portugal, 25–28 April 2010. pp. 426–440 (2010). [https://doi.org/10.1007/978-3-642-12683-3\\_28](https://doi.org/10.1007/978-3-642-12683-3_28)

39. Peng, Y., Leung, H.C.M., Yiu, S., Lv, M., Zhu, X., Chin, F.Y.L.: IDBA-tran: a more robust de novo de Bruijn graph assembler for transcriptomes with uneven expression levels. *Bioinformatics* **29**(13), 326–334 (2013). <https://doi.org/10.1093/bioinformatics/btt219>
40. Pevzner, P.A.: 1-tuple DNA sequencing: computer analysis. *J. Biomol. Struc. Dyn.* **7**(1), 63–73 (1989)
41. Plesník, J.: The np-completeness of the hamiltonian cycle problem in planar digraphs with degree bound two. *Inf. Process. Lett.* **8**(4), 199–201 (1979). [doi.org/10.1016/0020-0190\(79\)90023-1](https://doi.org/10.1016/0020-0190(79)90023-1)
42. Rautiainen, M., Marschall, T.: Aligning sequences to general graphs in  $O(v + me)$  time. *bioRxiv* p. 216127 (2017)
43. Ren, X., et al.: Evaluating de Bruijn graph assemblers on 454 transcriptomic data. *PLoS ONE* **7**(12), e51188 (2012)
44. Williams, V.V.: Hardness of easy problems: basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In: 10th International Symposium on Parameterized and Exact Computation, IPEC 2015, 16–18 September 2015, Patras, Greece. pp. 17–29 (2015). <https://doi.org/10.4230/LIPIcs.IPEC.2015.17>
45. Ye, Y., Tang, H.: Utilizing de Bruijn graph of metagenome assembly for meta-transcriptome analysis. *Bioinformatics* **32**(7), 1001–1008 (2016). <https://doi.org/10.1093/bioinformatics/btv510>
46. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.* **18**(5), 821–829 (2008)