The Complexity of Approximate Pattern Matching on De Bruijn Graphs

Daniel Gibney[1], Sharma V. Thankachan[2], and Srinivas Aluru[1]

[1] School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, USA
[2] Department of Computer Science, University of Central Florida, Orlando, USA
Email: daniel.j.gibney@gmail.com, sharma.thankachan@ucf.edu, aluru@cc.gatech.edu

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. 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 $\tilde{O}(n\sqrt{m})$ time, where n is the text's length [Abrahamson, SIAM J. Computing 1987].

1 Introduction

De Bruijn graphs are an essential tool in Computational Biology. Their role in de novo assembly spans back to the 1980s [39], and their application in assembly has been researched extensively since then [10,11,17,32,37,38,42,45]. More recently, de Bruijn graphs have been applied in metagenomics and in the representation of large collections of genomes [14,26,29,36,44] and for solving other problems such as read-error correction [31,34] and compression [9,23]. 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 [31]. 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 [5,21,22,28,30,33,35,41].

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.

	Exact Matching	Approximate Matching
Easy	Solvable in Linear Time • Wheeler Graphs [16] (e.g. de Bruijn graphs, NFAs for multiple strings)	Solvable in $\mathcal{O}(\|E\|m)$ time • DAGs: Substitutions/Edits to graph [28] • General graphs: Substitutions/Edits to pattern [6] • de Bruijn Graphs: Substitutions to pattern -No strongly Sub-$\mathcal{O}(\|E\|m)$ algorithm (this paper)
Hard	NO Strongly Sub-$\mathcal{O}(\|E\|m)$ Algorithm • General graphs [13,19] (including DAGs with total degree ≤ 3)	NP-Complete • General graphs: Substitutions/Edits to vertex labels [6,25] • de Bruijn Graphs: Substitutions to vertex labels (this paper)

Table 1. Overview of the computational complexity of pattern matching on labeled graphs

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 [28]. 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 [25]. 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. Furthermore, algorithmic solutions appearing in [28,35,41] are for the case where modifications are performed only to the pattern.

De Bruijn graphs are an interesting case 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 the exact pattern matching problem is solvable in linear time. However, the complexity of finding an approximate match in a de Bruijn graph when permitting modifications to the graph or modifications to the pattern remained open [25].

Our work makes 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. Such a proof confirms the optimality of the known quadratic time algorithms when considering polynomial factors. 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 length of the text [2,7]. Our results demonstrate that *the properties that make de Bruijn graphs amenable to efficient exact pattern matching do not extend to approximate matching.* To the best of our knowledge, these are the first such results for any type of Wheeler graph.

1.1 Technical Background and Our Results

Some necessary terminology is presented next.

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, ..., 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: A full de Bruijn graph is a directed graph whose vertices consists of all k-mers (strings of length k) from an alphabet $\Sigma = \{0, 1, ..., \sigma - 1\}$ in which there is a directed edge from each vertex $s_1 s_2 ... s_k \in \Sigma^k$ to the σ vertices $s_2 s_3 ... s_k \alpha$, $\alpha \in \Sigma$. We will work with induced subgraphs of full de Bruijn graphs in this paper. We consider a vertex v as being labeled with a single symbol $L(v) \in \Sigma$. The k-mer corresponding to the vertex is referred to as its *implicit label*. Formally, the implicit label of a vertex v is $L(u_1)L(u_2)...L(u_{k-1})L(v)$ where $u_1, u_2, ..., u_{k-1}, v$ is any length $k - 1$ walk ending at v. When no such walk exists, we assume v has an implicit label

compatible with the other implicit labels in the graph. 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^{th} symbol in S. We use $S[i, j]$ to denote the substring $S[i]S[i+1]...S[j]$. 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 $L(v) \in \Sigma = \{0, 1, ..., \sigma - 1\}$. We say that a pattern P matches a walk v_1, v_2, ..., 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 Σ, pattern $P[1, m]$, and integer $\delta \geq 0$, determine if there exists a walk in D matching P after at most δ substitutions to the vertex labels.

Theorem 1. *Problem 1 is NP-complete on de Bruijn graphs having an alphabet of size $\sigma \geq 4$.*

Theorem 1 is proven in Section 2. Intuitively, our reduction transforms a general directed graph into a de Bruijn that maintains key topological properties related to the existence of walks. In our proof, the k-mer size is dependent on the alphabet size σ. For example, when $\sigma = 4$, we require $k = \Theta(\log^2 |V|)$, but when $\sigma = \Theta(\log |V|)$, we can reduce the value of k to $\Theta(\log |V|)$. An extension of the hardness result for large σ on connected graphs is given at the end of Section 2.1.

The distinct problem of approximately matching a pattern to a *path* in a de Bruijn graph was shown to be NP-complete in [30]. 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 [27] 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 Σ, pattern $P[1, m]$, and integer $\delta \geq 0$, determine if there exists a walk in D matching P after at most δ substitutions to the symbols in P.

For Problem 2, in Section 3 we provide a SETH-based hardness result. The Strong Exponential Time Hypothesis, or SETH, is frequently utilized in establishing the conditional optimality of polynomial time algorithms, often for problems on strings or graphs [1,8,13,18,19,24]. We refer the reader to [43] 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 algorithm for Problem 2 on de Bruijn graphs running in time $\mathcal{O}(|E|m^{1-\varepsilon} + |E|^{1-\varepsilon}m)$. This holds for alphabets of size four (or greater) and k-mer size that is $\Theta(\log |E|)$.*

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. 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.

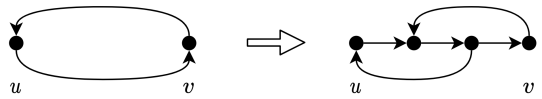

Fig. 1. Gadget to remove cycles of length 2 from the initial input graph.

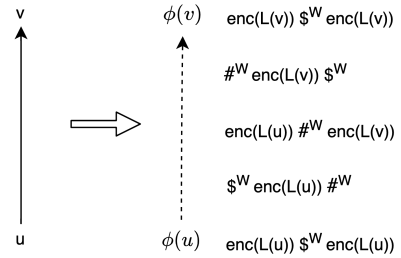

Fig. 2. The transformation from edges to paths used in our reduction.

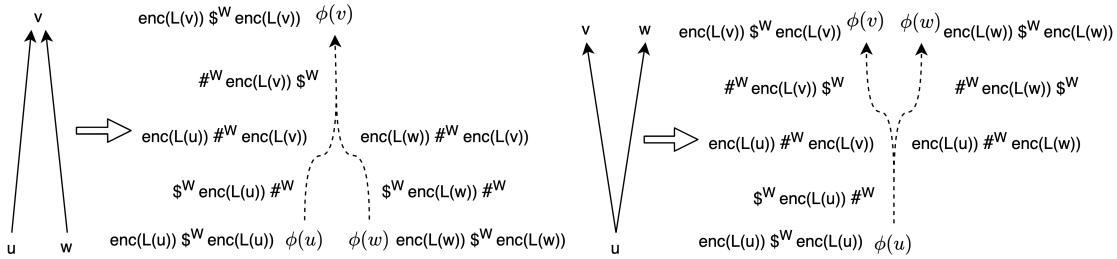

Fig. 3. Vertices with the same implicit label are merged while transforming D to D', causing edges with shared vertices to become paths with shared vertices.

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. The Hamiltonian cycle problem remains NP-complete even when restricted to such graphs [40]. 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 Figure 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 ℓ bits, $\sigma \in [4, 2\ell]$, and $\Sigma = \{\$, \#, 0, 1, ..., \sigma - 3\}$. Define $\text{enc}(i) = (0^{2\ell}1)^\alpha 23...(\sigma-3)\,\text{bin}(i)$, where if $\sigma = 4$ then the substring $23...(\sigma-3)$ is the empty string, and $\alpha = 2\ell - \sigma + 4$. Let $W = |\,\text{enc}(i)| = (2\ell+1)\alpha + \sigma - 4 + \ell$ and $k = 3W$. Note that if $\sigma = 2\ell$, $k = \mathcal{O}(\log n)$.

We construct a new (de Bruijn) graph $D' = (V', E')$ as follows: Initially D' is the empty graph. For $i = 0, 1, \ldots, n-1$, for each edge $(u, v) \in E$ where $L(v) = i$, create a new path whose concatenation of vertex labels is $\#^W\text{enc}(i)\$^W\text{enc}(i)$. 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 Figure 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 (Figure 3). See Figure 4 for a complete example. Finally, let $\delta = 2\ell(n-1)$ and

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

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

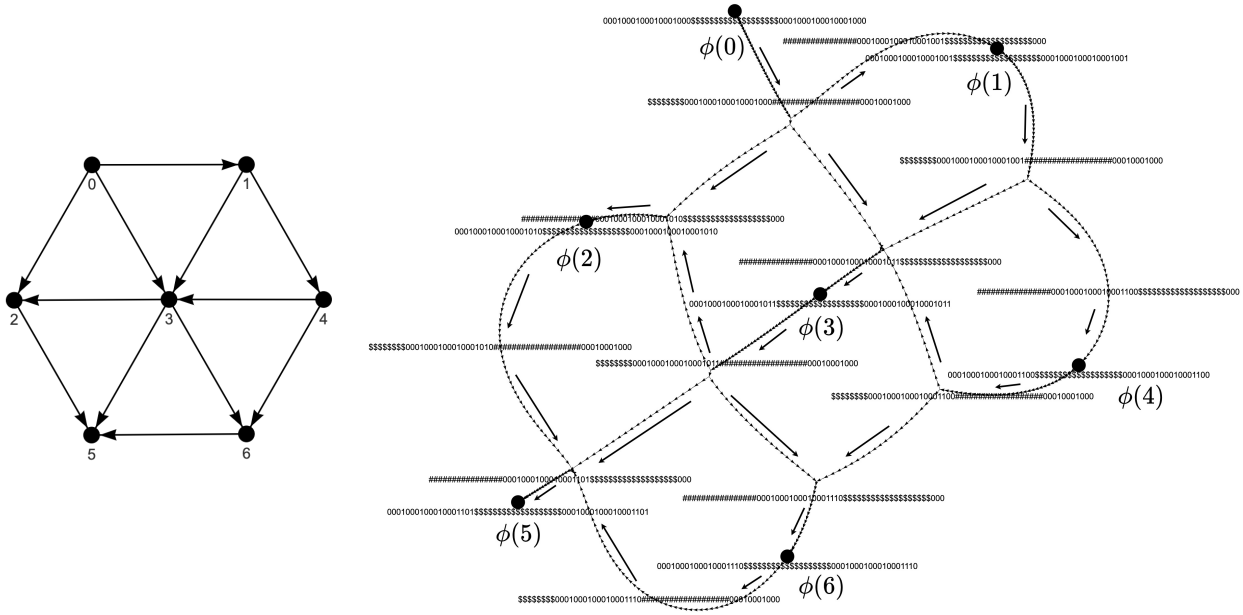

Fig. 4. (Left) A graph before the reduction is applied to it. (Right) The transformed graph. A subset of the implicit labels are shown, and the path directions are annotated by arrows beside each path. Note that enc$(\cdot)$ has been modified to have the prefix $(0^\ell 1)^{\ell+1}$ so that it fits in the figure. Also, unlike in the figure, we assume in our reduction that there are no vertices with in-degree or out-degree zero.

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 full proof is given in Appendix 1. □

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.

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. See Appendix 1 for the full proof. □

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 $\{u_1, v_1\} \cap \{u_2, v_2\} = \emptyset$ (Figure 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 $u_1 \neq v_2$ (Figure 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 Figure 1, D cannot contain the edges (u_1, v_1) and (v_1, u_1). □

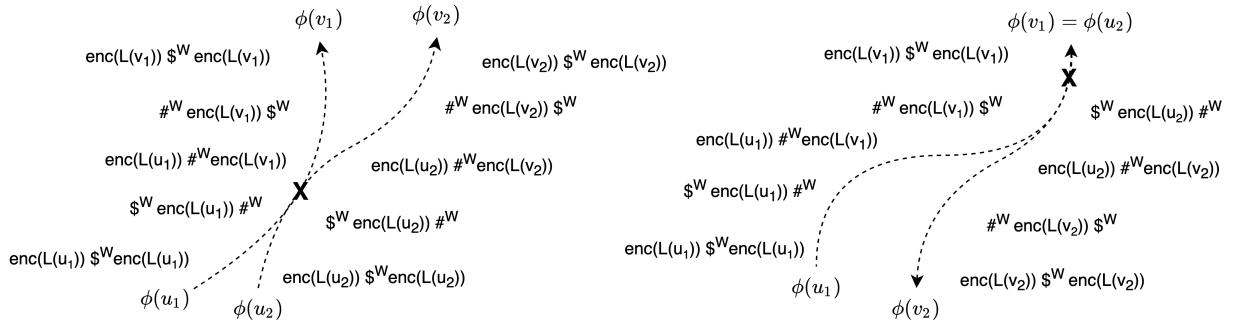

Fig. 5. Examples where paths between marked vertex cannot share any vertex: (Left) The case where $\{u_1, v_1\} \cap \{u_2, v_2\} = \emptyset$. (Right) The case where $v_1 = u_2$ and $u_1 \neq v_2$.

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 there is an edge $(u, v) \in E$.*

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 full proof is provided in Appendix 1. □

Lemma 5. *If D has a Hamiltonian cycle, then P can be matched in D' with at most δ 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. □

Lemma 6. *If P can be matched in D' with at most δ 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 δ substitutions. The full proof is provided in Appendix 1. □

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 δ 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 $S_1\$^W S_2$ 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 $S_1\$^W S_2$ post-substitution. □

Lemma 8. *If P can be matched in D' with at most δ 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 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 enc(0) is visited at least three times, a contradiction. □

Lemma 9. *If P can be matched in D' with at most δ 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 enc(0)), the walk matched by P in D' corresponds to a Hamiltonian cycle through D beginning and ending at the vertex labeled 0. □

Finally, we note that the proof of Lemma 6 establishes that there is a unique set of at least $\Theta(k)$ vertices for every marked vertex. Additionally, at most $4W|E| = \mathcal{O}(k|V|)$ vertices are created in the reduction. Hence, $|V'| = \Theta(k|V|)$. For constant alphabet sizes, $k = \Theta(\log^2 |V|)$, implying $k = \Theta(\log^2 |V'|)$, and for $\sigma = 2\lceil \log n \rceil$, $k = \Theta(\log |V|)$, implying $k = \Theta(\log |V'|)$. This completes the proof of Theorem 1 and subsequent statements regarding k and σ.

Hardness for large alphabets: One can increase the alphabet size to $\sigma' > \sigma$ for $\sigma' \leq \frac{|V''|}{2}$ ($|V''|$ being the final number of vertices in the instance of Problem 1). This can be done with only slight modifications to the existing reduction. Additionally, with the modifications, the implicit label length k remains unaltered. For every symbol $\alpha \in \{\sigma+1, \sigma+2, ..., \sigma'\}$, add a new vertex y labeled with α to V'. Pick an arbitrary original vertex $x \in V'$ and add the edge (x, y). Then y can only be used when matching P if it is matched with the end symbols in P. However, in an optimal solution, we can assume this is not done, as α matches no symbol in P.

3 SETH-Based 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 $\varepsilon > 0$ [43].

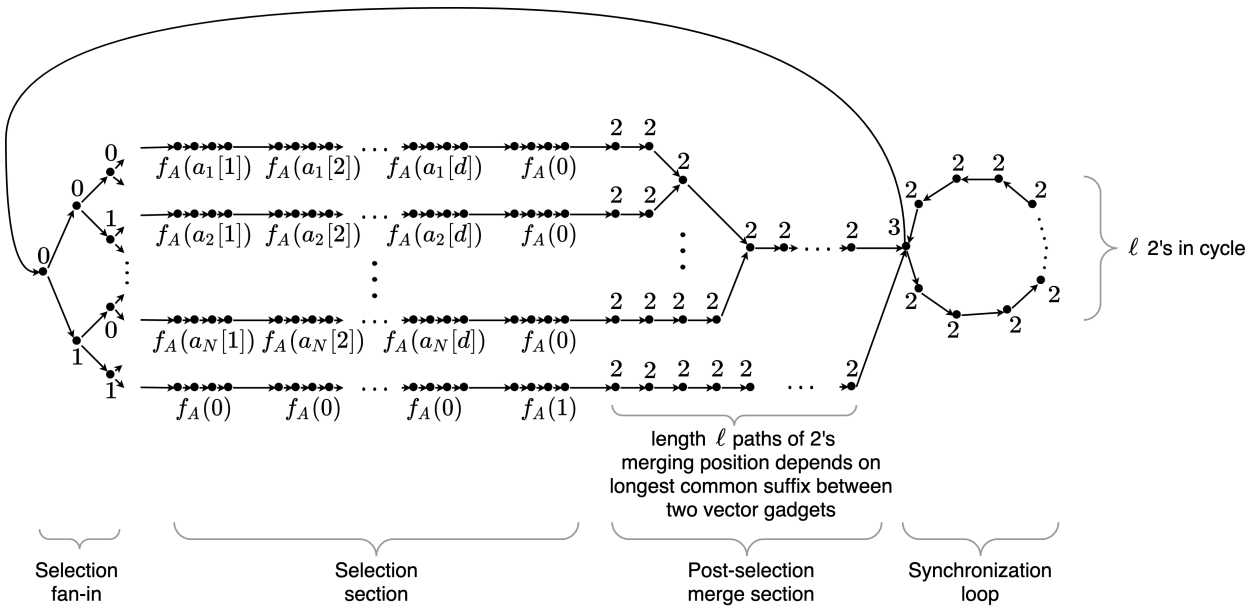

Fig. 6. An illustration of the reduction from OV to Problem 2.

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 $A = \{a_1, a_2, ..., a_N\}$ and the pattern P it creates from the set $B = \{b_1, b_2, ..., b_N\}$. The reader may find Figure 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 $d_H(f_A(1), f_B(1)) = 4$. We make the i^{th} path for $1 \leq i \leq N$ a path of $4(d+1)$ vertices with labels matching the string $f_A(a_i[1])f_A(a_i[2])...f_A(a_i[d])f_A(0)$. 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 denoted the start vertex of path i. We arbitrarily choose $N+1$ leaves, l_1, l_2,..., 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^ℓ. 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^ℓ. This is done so that every path matching 2^ℓ 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 $f_A(a_i[1]) f_A(a_i[2]) ... f_A(a_i[d]) f_A(0)$ vector gadgets. Note that $|E| = \mathcal{O}(dN)$ and $m = |P| = \mathcal{O}(d^2 N)$. 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 given in Appendix 2.

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. □

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 δ 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 δ. The synchronization loop can be used for matching all additional symbols in P without any further substitutions. □

Lemma 14. *If P can be matched in D with at most δ 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, which ever 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. □

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 (in general, not just on de Bruijn graphs) are hard for small δ values. In applications, the allowed error thresholds are quite small. Clearly, the problems are slice-wise-polynomial with respect to δ, i.e., for a constant δ it is solvable in polynomial time via brute force, but are they fixed-parameter-tractable in δ? The reduction presented here (as well as the reductions in [6,25]) is based on the Hamiltonian cycle problem, where a large δ value is used. This makes the existence of such a fixed-parameter-tractable algorithm a distinct possibility.

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, March 26-29, 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. Amir, A., Lewenstein, M., Porat, E.: Faster algorithms for string matching with k mismatches. J. Algorithms 50(2), 257–275 (2004), https://doi.org/10.1016/S0196-6774(03)00097-X
8. 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, Hyatt Regency, New Brunswick, New Jersey, USA. pp. 457–466 (2016), https://doi.org/10.1109/FOCS.2016.56
9. Benoit, G., Lemaitre, C., Lavenier, D., Drezen, E., Dayris, T., Uricaru, R., Rizk, G.: 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
10. 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
11. 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
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, July 9-12, 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, November 16-21, 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: String Processing and Information Retrieval - 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13-15, 2020, Proceedings. pp. 76–88 (2020), https://doi.org/10.1007/978-3-030-59212-7_6
19. Gibney, D., Hoppenworth, G., Thankachan, S.V.: Simple reductions from formula-sat to pattern matching on labeled graphs and subtree isomorphism. In: 4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 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, September 9-11, 2019, Munich/Garching, Germany. pp. 51:1–51:16 (2019), https://doi.org/10.4230/LIPIcs.ESA.2019.51
21. 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
22. Holley, G., Peterlongo, P.: Blastgraph: intensive approximate pattern matching in string graphs and de-bruijn graphs. In: PSC 2012 (2012)

23. 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
24. 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, September 7-9, 2020, Pisa, Italy (Virtual Conference). pp. 61:1–61:19 (2020), https://doi.org/10.4230/LIPIcs.ESA.2020.61
25. Jain, C., Zhang, H., Gao, Y., Aluru, S.: On the complexity of sequence to graph alignment. In: Research in Computational Molecular Biology - 23rd Annual International Conference, RECOMB 2019, Washington, DC, USA, May 5-8, 2019, Proceedings. pp. 85–100 (2019), https://doi.org/10.1007/978-3-030-17083-7_6
26. Kamal, M.S., Parvin, S., Ashour, A.S., Shi, F., Dey, N.: De-bruijn graph with mapreduce framework towards metagenomic data classification. International Journal of Information Technology 9(1), 59–75 (2017)
27. Kapun, E., Tsarev, F.: On np-hardness of the paired de bruijn sound cycle problem. In: Algorithms in Bioinformatics - 13th International Workshop, WABI 2013, Sophia Antipolis, France, September 2-4, 2013. Proceedings. pp. 59–69 (2013), https://doi.org/10.1007/978-3-642-40453-5_6
28. 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
29. 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. Bioinform. 31(10), 1674–1676 (2015), https://doi.org/10.1093/bioinformatics/btv033
30. 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
31. Limasset, A., Flot, J., Peterlongo, P.: Toward perfect reads: self-correction of short reads via mapping on de bruijn graphs. Bioinform. 36(2), 651 (2020), https://doi.org/10.1093/bioinformatics/btz548
32. Lin, Y., Shen, M.W., Yuan, J., Chaisson, M., Pevzner, P.A.: Assembly of long error-prone reads using de bruijn graphs. In: Research in Computational Molecular Biology - 20th Annual Conference, RECOMB 2016, Santa Monica, CA, USA, April 17-21, 2016, Proceedings. p. 265 (2016), https://link.springer.com/content/pdf/bbm%3A978-3-319-31957-5%2F1.pdf
33. Liu, B., Guo, H., Brudno, M., Wang, Y.: debga: read alignment with de bruijn graph-based seed and extension. Bioinform. 32(21), 3224–3232 (2016), https://doi.org/10.1093/bioinformatics/btw371
34. Morisse, P., Lecroq, T., Lefebvre, A.: Hybrid correction of highly noisy long reads using a variable-order de bruijn graph. Bioinform. 34(24), 4213–4222 (2018), https://doi.org/10.1093/bioinformatics/bty521
35. Navarro, G.: Improved approximate pattern matching on hypertext. Theor. Comput. Sci. 237(1-2), 455–463 (2000), https://doi.org/10.1016/S0304-3975(99)00333-3
36. 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
37. Peng, Y., Leung, H.C.M., Yiu, S., Chin, F.Y.L.: IDBA - A practical iterative de bruijn graph de novo assembler. In: Research in Computational Molecular Biology, 14th Annual International Conference, RECOMB 2010, Lisbon, Portugal, April 25-28, 2010. Proceedings. pp. 426–440 (2010), https://doi.org/10.1007/978-3-642-12683-3_28
38. 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. Bioinform. 29(13), 326–334 (2013), https://doi.org/10.1093/bioinformatics/btt219
39. Pevzner, P.A.: 1-tuple dna sequencing: computer analysis. Journal of Biomolecular structure and dynamics 7(1), 63–73 (1989)
40. 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), https://doi.org/10.1016/0020-0190(79)90023-1
41. Rautiainen, M., Marschall, T.: Aligning sequences to general graphs in o (v+ me) time. bioRxiv p. 216127 (2017)
42. Ren, X., Liu, T., Dong, J., Sun, L., Yang, J., Zhu, Y., Jin, Q.: Evaluating de bruijn graph assemblers on 454 transcriptomic data. PloS one 7(12), e51188 (2012)
43. 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, September 16-18, 2015, Patras, Greece. pp. 17–29 (2015), https://doi.org/10.4230/LIPIcs.IPEC.2015.17
44. Ye, Y., Tang, H.: Utilizing de bruijn graph of metagenome assembly for metatranscriptome analysis. Bioinform. 32(7), 1001–1008 (2016), https://doi.org/10.1093/bioinformatics/btv510
45. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short read assembly using de bruijn graphs. Genome research 18(5), 821–829 (2008)

Appendix

1 Missing Proofs in Section 2.1

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

Proof. There are three properties that must be proven: (i) Implicit labels are unique, meaning for every implicit label at most one vertex is assigned that label; (ii) There are no edges 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). Property (i) holds since after every edge transformation, vertices with the same implicit label are merged, making every implicit label occur at most once. For property (ii), consider the completed D' and an arbitrary vertex y on an arbitrary path $\phi((u, v))$. Regarding a possible edge $(x, y) \in E'$, we have the following cases:

- Case: the implicit label of y is $S\alpha = \text{enc}(L(u))\$^W \text{enc}(L(u))$. Then, any potential $x \in V'$ must have an implicit label $\beta S = \beta \,\text{enc}(L(u))\$^W \text{enc}(L(u))[1, W-1]$. However, the only implicit labels created that have a suffix of the form $\text{enc}(L(u))\$^W \text{enc}(L(u))[1, W-i]$ have a prefix $\#^{W-i}$. This implies that $\beta = \#$, and the edge (x, y) already exists in E' (under the assumption that there are no vertices with in-degree zero in V).
- Case: the implicit label of y is $S\alpha = \text{enc}(L(u))[i, W]\$^W \text{enc}(L(u))\#^{i-1}$, $1 < i \leq W + 1$. Then, any potential x must have an implicit label $\beta S = \beta \,\text{enc}(L(u))[i, W]\$^W \text{enc}(L(u))\#^{i-2}$. Because the only implicit labels with the substring $\$^W \text{enc}(L(u))$ have a prefix consisting of some suffix of $\text{enc}(L(u))$, this implies $\beta = \text{enc}(L(u))[i-1]$, and (x, y) already exists in E'.
- Case: the implicit label of y is $S\alpha = \$^{W-i} \text{enc}(L(u))\#^W \text{enc}(L(v))[1, i]$, $1 \leq i \leq W$. Then, any potential x must have an implicit label $\beta S = \beta\$^{W-i} \text{enc}(L(u))\#^W \text{enc}(L(v))[1, i-1]$. In the case $i < W$, since there are no runs of \$ in any implicit label shorter than W, this implies that $\beta = \$$, and the edge (x, y) already exists in E'. In the case where $i = W$, the only implicit label with a suffix of the form $\text{enc}(L(u))\#^W \text{enc}(L(v))[1, W-1]$, has a prefix \$, and the edge (x, y) already exists in E'.
- Case: the implicit label of y is $S\alpha = \text{enc}(L(u))[i, W]\#^W \text{enc}(L(v))\$^{i-1}$, $1 < i \leq W + 1$. Then, any potential x must have an implicit label $\beta S = \beta \,\text{enc}(L(u))[i, W]\#^W \text{enc}(L(v))\$^{i-2}$. Because the only implicit labels with the substring $\#^W \text{enc}(L(v))$ have a prefix consisting of some suffix of $\text{enc}(L(u'))$ where the edge (u', v) is in D, the edge (x, y) already exists in E'. This is an interesting case, as merges can happen, i.e., $\beta \,\text{enc}(L(u))[i, W] = \text{enc}(L(u'))[i-1, W]$, $u' \neq u$.
- Case: the implicit label of y is $S\alpha = \#^{W-i} \text{enc}(L(v))\$^W \text{enc}(L(v))[1, i]$, $1 \leq i \leq W$. Then, any potential x must have an implicit label $\beta S = \beta\#^{W-i} \text{enc}(L(v))\$^W \text{enc}(L(v))[1, i-1]$. For $i < W$, since there are no runs of \# in any implicit label shorter than W, $\beta = \#$, and the edge (x, y) already exists in E'. For $i = W$, this is equivalent to the first case.

We prove (iii) using induction on the number of edges transformed into paths. Our inductive hypothesis (IH) is that prior to an edge being replaced by a path, property (iii) holds for every vertex added to V' thus far. Let i denote the number of edges transformed. For $i = 1$, all vertices

where there exists such a walk ending at them are on the newly created path, and implicit labels are well-defined.

For $i > 1$, we assume the IH holds for all vertices created in the previous $i-1$ steps of transforming edges and merging. First consider a new vertex x that is created by transforming the i^{th} edge (u_i, v_i). Starting with $x = \phi(u_i)$. If x is merged with another transformed vertex x' having the same implicit label, then all length $k-1$ walks ending at x' match this implicit label, and thus the IH holds for x after merging. Using a secondary induction step, we assume the IH holds post-merging for all vertices between $\phi(u_i)$ and x (not including x) on $\phi((u_i, v_i))$. Let x_{prev} be the vertex on $\phi((u_i, v_i))$ before x. Since all length $k-1$ walks ending at x_{prev} match x_{prev}'s implicit label, the length $k-1$ walks obtained by disregarding the vertex at the start of these walks, and adding the vertex x at the end, all match the implicit label of x. At the same time, any vertices merged with x, by the IH also have the same implicit label and hence the walks ending at them match the implicit label of x. Hence, the IH holds for x after merging it with all vertices having the same implicit label. After processing all vertices on $\phi((u_i, v_i))$, we next consider a previously created vertex $x'' \in V'$ not in $\phi((u_i, v_i))$. Consider a newly created walk W ending at x'' that is due to a vertex merging with vertices in $\phi((u_i, v_i))$. Since all length $k-1$ walks ending at a vertex in $\phi((u_i, v_i))$ match the same implicit label, when disregarding some number of vertices at the start of a walk that ends in $\phi((u_i, v_i))$ and adding new vertices at the end to obtain W, the resulting walk W matches the implicit label for x'', and the IH continues to hold for x'' as well. □

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. We first define forward distance and backward distance. Let $x, y \in V'$. The forward distance from x to y is defined as the minimum number of edges on any path from x to y (the usual distance in a directed graph). The backward distance from x to y is defined as the minimum number of edges on any path from y to x. We say a marked vertex $\phi(u)$ is backward adjacent to x if there exists a walk from $\phi(u)$ to x not containing any other marked vertices, and x is forward adjacent $\phi(u)$ if there exists a walk from x to $\phi(u)$ not containing any other marked vertices.

We use induction on the number of edges transformed. Our inductive hypothesis (IH) will be that the length of all walks that end at and contain only two marked vertices is $4W$. We add to our IH that a vertex x created from an edge transformation having an implicit label of the form:

1. $\text{enc}(L(u))[j, W]\$^W \text{enc}(L(u))\#^{j-1}$, $1 \le j \le W$, has backward distance $j-1$ from $\phi(u)$, which is its only backward adjacent marked vertex, and forward distance $4W - j + 1$ from all of its forward adjacent marked vertices;
2. $\$^{W-j} \text{enc}(L(u))\#^W \text{enc}(L(v)))[1, j]$, $0 \le j \le W$, has backward distance $W + j$ from $\phi(u)$, which is its only backward adjacent marked vertex, and forward distance $3W - j$ from all of its forward adjacent marked vertices;
3. $\text{enc}(L(u))[j, W]\#^W \text{enc}(L(v)))\$^{j-1}$, $1 \le j \le W$, has backward distance $2W + j - 1$ from all of its backward adjacent marked vertices, and forward distance $2W - j + 1$ from $\phi(v)$, which is its only forward adjacent marked vertex;
4. $\#^{W-j} \text{enc}(L(v)))\$^W \text{enc}(L(v))[1, j]$, $0 \le j \le W$, has backward distance $3W + j$ from all of its backward adjacent marked vertices, and forward distance $W - j$ from $\phi(v)$, which is its only forward adjacent marked vertex.

The base case, $i = 1$, is satisfied since there exists only one such path and all stated properties hold. Now, for $i > 1$, let (u_i, v_i) be the i^{th} edge transformed. We assume the IH holds for all vertices and

walks created in the first $i-1$ edge transformations. First, observe that for any walk ending at, and containing only two previously created marked vertices, for all vertices on this walk the distances from their forward adjacent marked vertices and backward adjacent marked vertices will not be altered unless one of the vertices on this walk is merged with a vertex on $\phi((u_i, v_i))$. Also, all of the stated properties in the IH also hold for $\phi((u_i, v_i))$ prior to merging any vertices. Now, let y be a vertex on $\phi((u_i, v_i))$. Starting with $y = \phi(u_i)$, and continuing from $\phi(u_i)$ to $\phi(v_i)$, we merge y with existing vertices when their implicit labels match. Because the stated distance properties hold for x and y prior to merging, they continue to hold for the vertex created from merging x and y as well. Moreover, for all of the vertices on any walk containing this now merged vertex the distances from its forward adjacent and backward adjacent marked vertices are unaltered. Because for every vertex in the new graph, these distances are unaltered, the IH regarding the length of $4W$ for walks containing only two marked vertices continues to hold as well. □

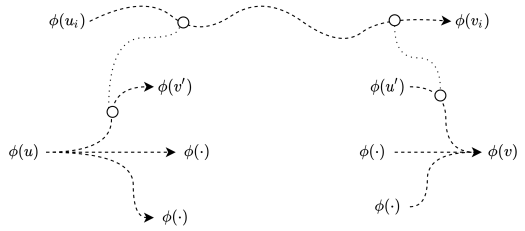

Fig. 7. In the proof of Lemma 4, we consider whether the path $\phi((u_i, v_i))$ being added could potentially cause a path between $\phi(u)$ and $\phi(v)$. The white circles connected by the thin dashed curve represent merged vertices.

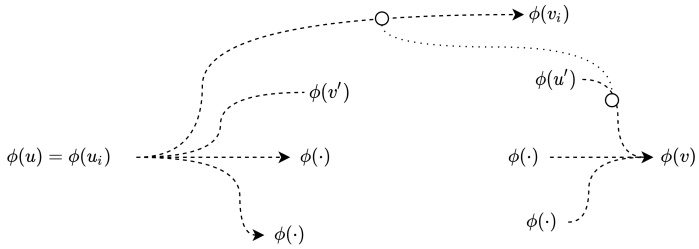

Fig. 8. In the proof of Lemma 4, the case where $u = u_i$ and $v' \neq v_i$.

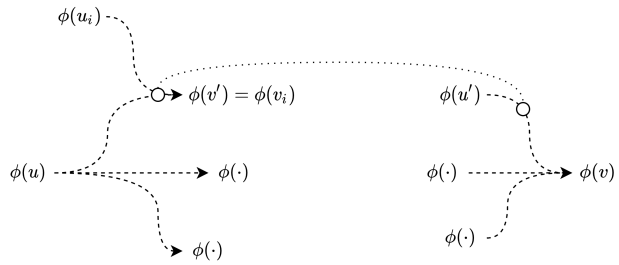

Fig. 9. In the proof of Lemma 4, the case where $u \neq u_i$, $v' = v_i$, and $u_i \neq u'$.

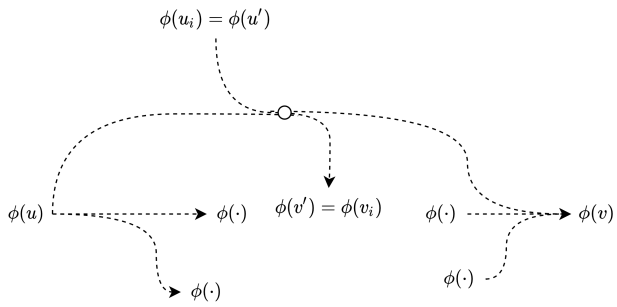

Fig. 10. In the proof of Lemma 4, the case where $u \neq u_i$, $v' = v_i$, and $u_i = u'$.

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 there is an edge $(u, v) \in E$.*

Proof. It is clear from construction that if there is an edge $(u, v) \in D$, then such a walk is in D'.

In the other direction, suppose for the sake of contradiction that there exists such a walk starting at $\phi(u)$ and ending at $\phi(v)$ with no other marked vertices between $\phi(u)$ and $\phi(v)$ on the walk, and

$(u, v) \notin E$. Let the first such walk be created when transforming the i^{th} edge (u_i, v_i). The only way such a walk could exists is if some vertex in $\phi((u_i, v_i))$ is merged with a vertex on a walk $\phi((u, v'))$ for some $v' \neq v$, and some vertex in $\phi((u_i, v_i))$ merged with a vertex in a walk $\phi((u', v))$ for some $u' \neq u$. This is since, prior to creating $\phi((u_i, v_i))$ all walks starting at $\phi(u)$ encountered some other marked vertex, say $\phi(v')$, before $\phi(v)$. Similarly, there existed some set of marked vertices not including $\phi(u)$ such that every walk containing a marked vertex and ending at $\phi(v)$ must include at least one vertex in this set, say $\phi(u')$. See Figure 7. Consider cases:

- $u = u_i$ and $v' = v_i$: This contradicts the assumption that (u_i, v_i) is transformed on the i^{th} step.
- $u = u_i$ and $v' \neq v_i$ (Figure 8): By Lemma 3, since $u_i = u \neq u'$, $\phi((u_i, v_i))$ and $\phi((u', v))$ can only share a vertex if $v_i = v$. However, this implies the edge $(u_i, v_i) = (u, v) \in E$, a contradiction.
- $u \neq u_i$ and $v' \neq v_i$: We can directly use Lemma 3 to say no such merged vertices exists between $\phi((u, v'))$ and $\phi((u_i, v_i))$.
- $u \neq u_i$ and $v' = v_i$ (Figure 9): By Lemma 3, if $u_i \neq u'$, then $\phi((u_i, v_i))$ and $\phi((u', v))$ can only share a vertex $v = v_i$. However, this would imply $v = v'$, a contradiction.
  The more interesting case is if $u_i = u'$ (Figure 10). Any vertex y having an implicit label containing $\text{enc}(L(u'))$ and occuring in $\phi((u_i, v_i))$ and $\phi((u', v))$ occurs before (has smaller backward distance to $\phi(u')$) any vertex with implicit label containing $\text{enc}(L(v'))$. At the same time, any vertex x occuring in $\phi((u, v'))$ and $\phi((u_i, v_i))$ has an implicit label containing $\text{enc}(L(v'))$. Since the vertex x occurs later in $\phi((u_i, v_i))$ than any shared vertex y in $\phi((u_i, v_i))$ and $\phi((u', v))$, the only way any vertices in $\phi((u_i, v_i))$ are in a walk from $\phi(u)$ to $\phi(v)$ not containing any other marked vertices is if there is walk from x to y not containing marked vertices, however, the cycle this creates contradicts Lemma 2. □

Lemma 6. *If P can be matched in D' with at most δ 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. We first make the following observations: pre-substitution of any of the vertex labels in D',

- (1) For all $u \in V$, there is exactly one path in D' that matches $\text{enc}(L(u))\#^W \text{enc}(L(u))[1, W-\ell]$, and all vertices on this path have in-degree and out-degree one. This follows from the only vertices with in-degree greater than one having implicit labels $\text{enc}(L(u))[i, W]\#^W \text{enc}(L(v))\$^{i-1}$ where $W - \ell < i \leq W + 1$ (these vertices have vertex label \$). And the vertices with out-degree greater than one having implicit labels of the form $\$^{W-i} \text{enc}(L(u))\#^W \text{enc}(L(v))[1, i]$ where $W - \ell \leq i \leq W$ (the last ℓ symbols in $\#^W \text{enc}(L(v))$. This path contains the marked vertex $\phi(u)$. Furthermore, all marked vertices are included on exactly one such path.
- (2) Every maximal walk containing only \$ or # symbols is of length W, and the distance from the end of any maximal walk consisting of only \$ symbols (or # symbols) to the start of a maximal walk consisting of only # (or \$ symbols resp.) is W. This follows from the construction of D': every vertex added in the construction has an implicit label where all maximal substrings consisting of non-\$ or non-# are of length W, and maximal substrings consisting of \$ or # are of length W.

To see the 'local' number of substitutions caused by matching a #/\$-symbol in D' to a 0/1 symbol in P, suppose the matching of $\text{enc}(L(u))$ in P is 'shifted left' by $1 \leq s < W$ so that

the first s symbols of some enc$(L(u))$ in P are matched against the last s symbols in some walk of \$/#-symbols in D', requiring s substitutions. Now, assuming $s < 2\ell$ and $\sigma > 4$, due to the prefix $(0^{2\ell}1)^{\alpha}23...(\sigma-3)$ of enc$(L(u))$, matching the shifted enc$(L(u))$ in D' will require at least $\alpha - 1 + (\sigma - 3)$ substitutions that do not involve a # or \$ symbol. When $\sigma = 4$, $\alpha = 2\ell$ the prefix is $(0^{2\ell}1)^{2\ell}$ and at least $2\ell - 1$ substitutions that do not involve a # or \$ symbol are needed.

We now look at the number of substitutions needed on a 'global' level. Using Lemma 2, it can be inferred that every walk of length $4W$ contains an originally marked vertex. Hence, while matching P' at least $\lfloor |P'|/4W \rfloor = 4Wn/4W = n$ times an originally marked vertex is visited. Because every substring of $P' = P[1, |P| - 4W]$ of length $3W - \ell$ is distinct, every path described in Observation 1 is traversed at most once while matching P'. Since each originally marked vertex is on a unique path that can be traversed at most once, and we traverse at least n such paths, we traverse n distinct paths described in Observation 1. We can now use Observation 2 to infer that the substitutions needed to match the shifted patterns in P' must be repeated n times. Hence, to match P' the total number of substitutions involving \$/# symbols is at least sn. For $s < 2\ell$ and $\sigma > 4$, the total number of substitutions involving only non-\$/# symbols is at least $(\alpha + \sigma - 4)n$. Using that $\alpha = 2\ell - \sigma + 4$, we have that the total number of substitutions is at least $(s+\alpha+\sigma-4)n = (s+2\ell)n > 2\ell(n-1) = \delta$. When $\sigma = 4$ and $\alpha = 2\ell$ the inequality above becomes $(s + 2\ell - 1)n > 2\ell(n-1) = \delta$. Independent of σ, when $2\ell \leq s < W$, then α substitutions to match the substring $(0^{2\ell}1)^{\alpha}$ in P may not be needed, but the total number of substitutions required is still greater than δ since $sn \geq 2\ell n > \delta$. A symmetric argument can be used for when the matching of P to D' is 'shifted right' by s so that the last s symbols in enc$(L(u))$ in P are matched against the first s symbols in some walk of \$/#-symbols in D'.

For $W < s < 4W$, it still holds that all paths described in Observation 1 are traversed exactly once. Combined with Observation 2, we can infer that the substitution cost incurred when making one path of length W originally matching $\#^W$'s match a substring of P without #'s is incurred at least n times. This results in the total number of needed substitutions being at least $nW > \delta$. □

2 Missing Proofs in Section 3.1

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

Proof. For each of the four graph sections discussed above, we will prove for each vertex in that section that Conditions (i)-(iii) from the proof of Lemma 1 hold. That is, every vertex v, v's implicit label well-defined, unique, and there are no additional edges that should have v as their head.

- Selection fan-in:
  - (well-defined) For any vertex v in the Selection fan-in, there are two paths of length $k-1$ leading to v (one containing vertices labeled with 2's from the Post-selection merge section and one containing vertices labeled with 2's from the Synchronization loop). Both match the same string $2^{\ell'}3B$ where $\ell' < \ell$ and B is a binary string of length at most $\lceil \log N + 1 \rceil$.
  - (unique) The binary string B could only possibly occur again as a suffix the Selection section. However, all implicit labels occurring in that section contain longer binary strings. Hence the implicit label occurs only once in D.
  - (no missing inbound edges) Let u be any vertex such that (u, v) is in D. A vertex v in the Selection fan-in has an implicit label of the form $S\alpha = 2^{\ell'}3B_{i'}[1, i]$, $\ell' < \ell$, $1 \leq i < \lceil \log N \rceil$, $0 \leq i' \leq N + 1$. This implies that u has the implicit label $\beta S = \beta 2^{\ell'}3B_{i'}[1, i-1]$. Based on

the limited number of implicit labels present in D, it must be that $\beta = 2$, and there exists only one such u. Hence, the edge (u, v) already exists.

– Selection section:

- (well-defined) For a vertex v in the Selection section, there are two length $k-1$ paths leading to v (one with 2's from the Post-selection merge section and one with 2's from the Synchronization loop). Both match a string of the form $2^{\ell'}3B_{i'}f_A(a_i[1])f_A(a_i[2])...f_A(a_i[j])[1,h]$ where $0 \leq \ell' < \ell$ and $1 \leq h \leq 4$.
- (unique) If v has a path of length $k-1$ matching $2^{\ell'}3B_{i'}f_A(a_i[1])f_A(a_i[2])...(f_A(a_i[j])[1,i]$, then it must be in the Selection section. The substring $B_{i'}$ following the prefix $2^{\ell'}3$ is distinct, hence this implicit label only occurs once in the Selection section.
- (no missing inbound edges) Taking u and v as above, if the vertex v has an implicit label of the form $S\alpha = 2^{\ell'}3B_{i'}f_A(a_i[1])f_A(a_i[2])...f_A(a_i[j])[1,h]$, $1 \leq h \leq 4$, this implies that the any potential u has an implicit label $\beta S = \beta 2^{\ell'}3B_{i'}[1,h-1]f_A(a_i[1])f_A(a_i[2])...f_A(a_i[j])[1,h-1]$ or $\beta S = \beta 2^{\ell'}3B_{i'}[1,h-1]f_A(a_i[1])f_A(a_i[2])...f_A(a_i[j-1])$. In either case, $\beta = 2$, and the edge (u, v) already exists. If the vertex v has an implicit label of the form $S\alpha = B_{i'}f_A(a_i[1])f_A(a_i[2])...f_A(a_i[d])$, then any potential vertex u has an implicit label $\beta S = \beta B_{i'}f_A(a_i[1])f_A(a_i[2])...f_A(a_i[d])[1,3]$ where β must be 3, and the edge (u, v) already exists.

– Post-selection merge section:

- (well-defined) For a vertex v in this section, all length $k-1$ paths ending at v match a string of the form $B2^{\ell'}$ where B is a binary string. By construction, the paths ending at v match the same string (they were merged based on this condition).
- (unique) Again by construction, if another vertex v' in the Post-selection merging section has a length $k-1$ path ending at it that matches v's implicit label v' will be merged with v. At the same time, vertices in the other sections of D will not have an implicit label of the form $B2^{\ell}$.
- (no missing inbound edges) Taking u and v as above, vertex v has an implicit label of the form $S\alpha = B2^{\ell'}$, $\ell' \geq 1$, this implies that any potential vertex u has an implicit label $\beta S = \beta B2^{\ell'-1}$. Such a vertex u is already in the Post-selection merge section or is a vertex at the end of a path in the Selection section (if $\ell' = 1$). Since appending a 2 and removing β will make the implicit label of u equal to the implicit label of v, the vertex at the head of the edge with tail u must have been merged with v. Hence, the edge (u, v) already exists.

– Synchronization loop:

- (well-defined) There are two length $k-1$ paths to a vertex v in the synchronization loop. Both match the string $2^{\ell'}32^{\ell''}$ where $\ell' + \ell'' = k - 1 = \ell$, and ℓ' depends on v's position within the Synchronization loop.
- (unique) An implicit label for a vertex in any other section contains a symbol that is not a 2 or a 3. Within the synchronization loop, each implicit label clearly occurs exactly once.
- (no missing inbound edges) Taking u and v as above, vertex v has an implicit label of the form $S\alpha = 2^{\ell'}32^{\ell''}$. This implies that any potential vertex u has an implicit label $\beta S = \beta 2^{\ell'}32^{\ell''-1}$. If $\ell' < \ell$ it must be that $\beta = 2$ and the edge (u, v) already exists. If instead $\ell' = \ell$, then for both $\beta S = 02^{\ell}3$ and $\beta S = 12^{\ell}3$ there already exists an edge (u, v) as well. □

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

Proof. Suppose that some 3 in P is not matched with 3 in D or with the final vertex in a path in the Selection section. Since any walk between 3's in D has a length that is a multiple of k and 3 in P are $k-1$ symbols apart, all 3's must then not be matched with 3 in D. This requires at least tN substitutions within P. On the other hand, when 3's in P are matched with 3's in D, there exists a solution requiring at most $4d(N+1)+N\lceil\log(N+1)\rceil$. Specifically, this is obtained by matching each vector gadget in P, $f_B(b_i[1])...f_B(b_i[d])$ to the $N+1^{th}$ path in the Selection section. Since $t = 5d+\lceil\log(N+1)\rceil > 4d+\frac{4d}{N}+\lceil\log(N+1)\rceil$ for $d = o(N)$ and N large enough, we can assume that $tN > 4d(N+1)+N\lceil\log(N+1)\rceil$. Hence, all 3's in P are matched with the 3 in D or with some final vertex in a path in the Selection section

Next, suppose some 3 in P is matched with the last vertex in a path in the Selection section. We consider the first such occurrence. In the case where this occurrence of 3 in P is followed in P by a substring $2^{\lceil\log(N+1)\rceil}f_B(a_i[1])...f_B(a_i[d])f_B(1)$, a cost of at least $8(d+1)$ is incurred, first at least $4(d+1)$ from matching the substring $2^\ell 3$ in P to a path through Selection fan-in and the Selection section, then an additional $4(d+1)$ from matching a vector gadget in P to a path of 2's in the Post-selection merge section. We could have instead matched the Synchronization loop twice with a cost of only $4(d+1)$ substitutions, and started and ended at the same vertex while still matching $2^\ell 32^{\lceil\log(N+1)\rceil}f_B(a_i[1])...f_B(a_i[d])f_B(1)$. Hence, in this case, matching 3 in P with the last vertex in a path in the Selection section is suboptimal. In the case where the occurrence of 3 in P is followed in P by $2^\ell 3$, then the cost incurred is only $4(d+1)$. However, we could have instead matched $2^\ell 32^\ell 3$ with the Synchronization loop twice with a substitution cost of 0, and again started and ended at the same vertex. Hence, matching 3 in P with the last vertex in a path in the Selection section is again suboptimal. □