On the Sound Covering Cycle Problem in Paired de Bruijn Graphs

Christian Komusiewicz[1⋆] and Andreea Radulescu[2]

[1] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
christian.komusiewicz@tu-berlin.de
[2] LINA - UMR CNRS 6241, Université de Nantes, France
andreea.radulescu@etu.univ-nantes.fr

Abstract. Paired de Bruijn graphs are a variant of classic de Bruijn graphs used in genome assembly. In these graphs, each vertex v is associated with two labels $\mathcal{L}(v)$ and $\mathcal{R}(v)$. We study the NP-hard Sound Covering Cycle problem which has as input a paired de Bruijn graph G and two integers d and ℓ, and the task is to find a length-ℓ cycle C containing all arcs of G such that for every vertex v in C and the vertex u which occurs exactly d positions after v in C, we have $\mathcal{R}(v) = \mathcal{L}(u)$. We present the first exact algorithms for this problem and several variants.

1 Introduction

DNA sequencing is the task of deciphering the sequence of a given DNA fragment. Most technologies approach this task by obtaining a collection of possibly overlapping small subfragments, called *reads*, of the given fragment. Genome assembly aims at recovering the original DNA fragment from the set of reads. When no reference genome is used, this is called *de novo* assembly.

Recent sequencing technologies, known as next-generation sequencing (NGS), create billions of very short erroneous reads. For this new type of data, *de novo* assembly is a challenging task [3, 5, 11, 13]. The use of short reads makes it particularly difficult to correctly assemble repeated regions since the repeat may be longer than the reads. Therefore, many NGS methods generate pairs of reads separated by a known distance, called insert length, that is much longer than the read length. This new type of reads, called *paired-end reads*, is used to span regions that contain long repeats.

One classic approach in de novo genome assembly is the *de Bruijn* method [6, 9] which computes a *de Bruijn graph* from the read set. This is done as follows. First, generate the set of k-mers of the reads, that is, the set of all length-k strings that occur as substrings of at least one read. Each k-mer in this set corresponds to exactly one vertex of the de Bruijn graph. Now draw an arc from a vertex u to a vertex v if and only if there is a $k+1$-mer s in one of the input reads such that the k-mer corresponding to u is a prefix of s and the k-mer corresponding to v is a suffix of s. A walk in this graph then corresponds to a DNA sequence. Classic de

⋆ Partially supported by the DAAD Procope program (project 55934856).

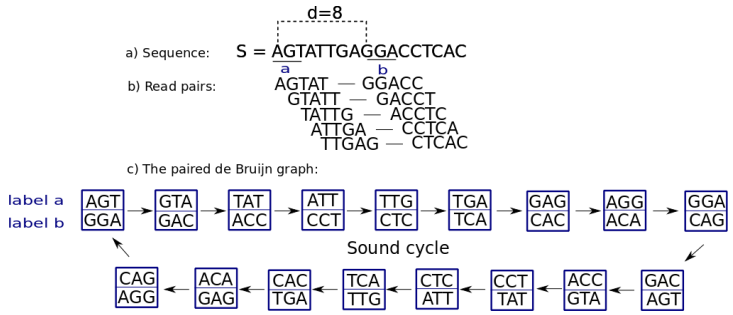

Fig. 1. A paired de Bruijn graph with a sound cycle: a) The DNA fragment to be sequenced. b) The set of paired reads with insert distance d=8. c) The paired de Bruijn graph constructed from the paired reads in b) for a k=3. The upper part of each vertex v is the label $\mathcal{L}(v)$, the lower part is the label $\mathcal{R}(v)$.

Bruijn assembling software uses paired-end information only in a post-processing step.

The *paired de Bruijn graph* model incorporates the paired-end information directly into the graph [8]. It is based on the classic de Bruijn graph, though now the overlaps are computed between pairs of k-mers separated by the insert length of the read pairs. This improves assembly quality and facilitates repeat detection but the computational problems involved in computing an assembly in these graphs become more challenging. In particular, the Sound Covering Cycle which we define below is NP-hard [7]. In this work, we present the first exact algorithms for Sound Covering Cycle and several variants.

Preliminaries. For a string s, let $s[i]$ denote the letter at position i of s and $s[i,j]$ the substring of s that starts at position i and ends at position j. We consider directed graphs $G = (V, A)$ with vertex set V and arc set A. A *walk* $(v_1, \ldots, v_p)$ is a tuple of vertices such that $(v_i, v_{i+1}) \in A$, $1 \leq i < p$. The *length* $|W|$ of a walk $W := (v_1, \ldots, v_p)$ is the number p of tuple elements. A walk is *simple* if $i \neq j$ implies $v_i \neq v_j$. Given two walks $W_1 = (v_1, \ldots, v_p)$ and $W_2 = (u_1, \ldots, u_q)$ such that $(v_p, u_1) \in A$, let $W_1 \cdot W_2 := (v_1, \ldots, v_p, u_1, \ldots, u_q)$ denote the *concatenation* of W_1 and W_2. For a walk $W = (v_1, \ldots, v_p)$, let $W[i] := v_i$ denote the i-th vertex of W. Finally, let $A(W)$ denote the set of arcs contained in a walk W.

Paired de Bruijn graphs. Before defining the problem, we describe how the paired de Bruijn graph is constructed from the input data; see Fig. 1 for an example. The input is a set $R := \{(r_1^{\mathcal{L}}, r_1^{\mathcal{R}}), \ldots, (r_m^{\mathcal{L}}, r_m^{\mathcal{R}})\}$ of paired-end reads and two integers d and k. Each $(r_i^{\mathcal{L}}, r_i^{\mathcal{R}})$ is a pair of strings of the same length over an alphabet Σ. The integer d is the *insert size* or *shift*. It specifies that the paired-end read corresponds to two substrings of the complete genome whose first letters have distance exactly d in the genome. The integer k is a user-defined parameter.

In a paired de Bruijn graph $G(R)$ constructed from a read set R, each vertex v is associated with a pair of k-mers $(\mathcal{L}(v), \mathcal{R}(v))$. This pair is called *bilabel*. Each

node has a unique bilabel. For a read set R, the vertex set of $G(R)$ is defined as

$$V(R) := \{(s,t) \in \Sigma^k \times \Sigma^k \mid \exists (r_i^{\mathcal{L}}, r_i^{\mathcal{R}}) \in R, p \in \mathbb{N} : s = r_i^{\mathcal{L}}[p, p+k-1] \wedge t = r_i^{\mathcal{R}}[p, p+k-1]\}. \quad (1)$$

An arc is drawn from a vertex u to a vertex v if some read in R contains the bilabels of u and v in consecutive positions. More precisely, the arc set of $G(R)$ is

$$\begin{aligned} A(R) := \{(u,v) \mid & \exists (r_i^{\mathcal{L}}, r_i^{\mathcal{R}}) \in R, p \in \mathbb{N} : \\ & \mathcal{L}(u) = r_i^{\mathcal{L}}[p, p+k-1] \wedge \mathcal{L}(v) = r_i^{\mathcal{L}}[p+1, p+k] \wedge \\ & \mathcal{R}(u) = r_i^{\mathcal{R}}[p, p+k-1] \wedge \mathcal{R}(v) = r_i^{\mathcal{R}}[p+1, p+k]\}. \end{aligned}$$

A walk in a paired de Bruijn graph directly corresponds to a string over Σ and thus to a DNA sequence. More precisely, a walk $W := (v_1, \ldots, v_p)$ in a paired de Bruijn graph spells the two strings $\mathcal{L}(v_1) \cdot \mathcal{L}(v_2)[k] \cdot \ldots \cdot \mathcal{L}(v_p)[k]$ and $\mathcal{R}(v_1) \cdot \mathcal{R}(v_2)[k] \cdot \ldots \cdot \mathcal{R}(v_p)[k]$. There is one walk in $G(R)$ that corresponds to the original DNA sequence. This walk fulfills several properties that we describe below. The computational task that we consider here is deciding whether a walk fulfilling these properties exists in $G(R)$. In the remainder of this work, the read set R is irrelevant in the computational problems, thus, we denote the paired de Bruijn graph $G = (V, A)$ instead of $G(R)$.

The Sound Covering Cycle problem. In the case of organisms with a single circular chromosome, the walk that corresponds to the genome is a cycle. Slightly abusing notation, we define a *cycle* in a graph $G = (V, A)$ as a walk $(v_1, \ldots, v_p)$, $v_i \in V$, such that $(v_p, v_1) \in A$. This cycle should have a length ℓ which is the estimated genome length. As described above, the cycle spells two cyclic strings. The cycle that spells the genome should also spell every pair of observed $k+1$-mers. By construction of G, a pair of $k+1$-mers in the read set corresponds to an arc between two vertices. Thus, we demand the cycle to contain every arc of the graph. Accordingly, a cycle C is called *covering* if for each arc $(u,v) \in A$ there is a v_i such that $(u,v) = (v_i, v_{i+1})$ or $(u,v) = (v_p, v_1)$.

The above properties are also relevant in classic de Bruijn graphs. In a paired de Bruijn graph the walks should also fulfill the insert size constraint. Recall that the distance between the $\mathcal{L}$- and $\mathcal{R}$-label of a vertex is d. The soundness constraint will ensure that the strings spelled by these two labels are consistent with the insert size. More precisely, in a paired de Bruijn graph, a cycle is called *sound* if the pair of strings it spells matches with shift d. Let s and t be the two strings spelled by the bilabels of the cycle C. If $d \leq \ell$, then we call C *sound* if

- $s[i+d] = t[i]$ for $1 \leq i \leq \ell - d$, and
- $s[i] = t[i+\ell-d]$ for $1 \leq i \leq d$.

Accordingly, we call a vertex v_i in a walk $(v_1, \ldots, v_q)$ *sound* if $\mathcal{R}(v_i) = \mathcal{L}(v_{i+d})$. The soundness definition corresponds to the one of Kapun and Tsarev [7]. This leads to our main problem definition.

Sound Covering Cycle
Input: A paired de Bruijn graph $G = (V, A)$ and nonnegative integers d and ℓ.
Question: Does G contain a sound covering cycle of length ℓ?

Related work. Kapun and Tsarev [7] show that Sound Covering Cycle is NP-hard even if the values of d or k are small constants. They also claim that if $|\Sigma|$ and k are constants, which implies that the graph has constant size, then Sound Covering Cycle cannot be NP-hard as the language defined by it is sparse since d is encoded in unary. We do not make this assumption, that is, in our case d and ℓ are encoded in binary. Thus, the complexity of Sound Covering Cycle for fixed graph size is open in our encoding. A related graph-based approach of modeling the information of paired-end reads are rectangle graphs [1, 10, 12]. Computing a covering cycle of length *at most* ℓ in a directed graph is known as Directed Chinese Postman and can be solved in polynomial time [4].

Contribution and organization of the paper. In Section 2, we describe a decomposition of cycles in directed graphs that we use throughout this work. Moreover, we describe an algorithm for computing a fixed-length covering cycle. This algorithm is used as a subroutine in Section 5 and may also be of independent interest. In Section 3, we present an algorithm for Sound Covering Cycle that runs in $f(n, d) \cdot \text{poly}(\log \ell)$ time. In Section 4, we present similar algorithms for variants of Sound Covering Cycle such as searching for a shortest covering sound cycle and dealing with relaxed models of soundness that model noisy input data. In Section 5, we present a special case of Sound Covering Cycle that is solvable in $f(n) \cdot \text{poly}(\log \ell + \log d)$ time. Since paired de Bruijn graphs are very sparse, we use the maximum outdegree Δ in our running time bounds.

Due to space constraints, most proofs are deferred to an appendix. We use the following observations in our algorithms.

Lemma 1. *Let $G = (V, A)$ be a directed graph with maximum outdegree Δ and let ℓ be an integer. There are at most $n \cdot \Delta^{\ell-1}$ walks and cycles of length ℓ in G and they can be enumerated in $O(n \cdot \Delta^{\ell-1} \cdot (\ell + \Delta))$ time.*

This statement implies the following bound on the number of simple walks.

Lemma 2. *A directed graph $G = (V, A)$ with n vertices and maximum outdegree Δ has at most $2n \cdot \Delta^{n-1}$ different simple walks.*

2 Cycle-Walk Decompositions

Before presenting our algorithms, we describe a structured representation of cycles and walks.

First, we show that we can decompose any walk or cycle into maximal simple walks (denoted by Ω_i) and possibly empty simple walks between them (denoted by W_i). Herein, the term maximal refers to the property that in C each $\Omega_i = (u_1, \ldots, u_t)$ is followed by its first vertex u_1. This implies in particular that Ω_i is a cycle.

Lemma 3. *Let C be a walk in a graph G. Then C can be written as a concatenation of simple walks $\Omega_1 \cdot W_1 \cdot \ldots \cdot \Omega_q \cdot W_q$ such that*

1. *$|\Omega_i| > 0$ for each $i \in \{1, \ldots, q\}$, and*
2. *for each $\Omega_i := (u_1, u_2, \ldots, u_s)$, $1 \leq i \leq q$ it holds that $u_1 = v_1$ where v_1 is the first vertex of $W_i \cdot \Omega_{i+1}$.*

A representation adhering to Lemma 3 is called *cycle-walk decomposition* of C. Our next aim is to show the existence of cycle-walk decompositions with a compact description. The proof exploits the fact that if there are too many different cycles in the decomposition, then some of them can be replaced by repetitions of other cycles.

Before proving Lemma 5, we show the correctness of the following exchange operation.

Lemma 4. *Let C be a covering cycle of a graph G with cycle-walk decomposition $\Omega_1 \cdot W_1 \cdot \ldots \cdot \Omega_q \cdot W_q$. If C contains a cycle Ω_j such that*

- *there is a walk Ω_i, $i < j$, that has the same length as Ω_j, and*
- *for each arc $a \in A(\Omega_j)$, there is a walk Ω_p, $p \neq j$, such that $a \in A(\Omega_p)$,*

then $C' := \Omega_1 \cdot \ldots \cdot W_{i-1} \cdot \Omega_i^2 \cdot W_i \cdot \ldots \cdot W_{j-1} \cdot W_j \cdot \ldots \cdot W_q$ is a covering cycle of the same length in G.

Proof. Let $\Omega_i := (u_1, u_2, \ldots, u_s)$ and $W_i \cdot \Omega_{i+1} := (w_1, w_2, \ldots, w_t)$. Since Ω_i is a cycle, $\Omega_i \cdot \Omega_i \cdot W_i$ is a walk. Now consider $\Omega_{j-1} \cdot W_{j-1} := (x_1, \ldots, x_s)$, $\Omega_j := (y_1, \ldots, y_t)$, and $W_j \cdot \Omega_{j+1} := (z_1, \ldots, z_r)$. Since C is a walk we have $(x_s, y_1) \in A$ and by the properties of cycle-walk decompositions also $y_1 = z_1$. Therefore, $(x_s, z_1) \in A$ and thus $\Omega_{j-1} \cdot W_{j-1} \cdot W_j \cdot \Omega_{j+1}$ is a walk. Consequently, C' is a walk. Since Ω_i and Ω_j have the same length, C' has the same length as C. Moreover, C' is covering as every arc of Ω_j is contained in some Ω_p, $p \neq j$. □

Using the exchange operation described by Lemma 4, we now show that there are compact cycle-walk decompositions.

Lemma 5. *If a directed graph G has a covering cycle C of length ℓ, then it has a covering cycle C' of length ℓ such that C' has a cycle-walk decomposition $(\Omega_1)^{r_1} \cdot W_1 \cdot \ldots \cdot (\Omega_q)^{r_q} \cdot W_q$ where $q \leq n + m$.*

Proof. Assume that G has a covering cycle C of length ℓ. According to Lemma 3, C has a cycle-walk decomposition $(\Omega_1)^{r_1} \cdot W_1 \cdot \ldots \cdot (\Omega_q)^{r_q} \cdot W_q$ (Lemma 3 shows the existence of the special case $r_1 = \ldots = r_q = 1$). Now consider of all covering cycles of G one with a decomposition in which $q + \sum_{i=1}^{q} |W_i|$ is minimum.

Now assume towards a contradiction, that in this decomposition there are indices i and j, $i \neq j$, such that $|\Omega_i| = |\Omega_j|$ and each arc of Ω_j is contained in some Ω_p, $p \neq j$. Without loss of generality assume $i < j$. We transform C into a new cycle C' in which $q + \sum_{i=1}^{q} |W_i|$ is smaller. This contradicts our choice of C.

By the assumption on i and j and by Lemma 4, $C' := \Omega_1 \cdot \ldots \cdot W_{i-1} \cdot (\Omega_i)^{r_i + r_j} \cdot W_i \cdot \ldots \cdot \Omega_{j-1} \cdot W_{j-1} \cdot W_j \cdot \Omega_{j+1} \cdot \ldots \cdot W_q$ is also a covering cycle of G. Clearly, $|C| = |C'| = \ell$ and C' is also a covering cycle. Now consider two cases.

Case 1: $W_{j-1} \cdot W_j$ contains a simple cycle Ω^*. Let $W_{j-1} \cdot W_j = W_1^* \cdot \Omega^* \cdot W_2^*$ where W_1^* and W_2^* are not simple cycles. Then, $C' := \Omega_1 \cdot \ldots \cdot W_{i-1} \cdot (\Omega_i)^{r_i+r_j} \cdot W_i \cdot \ldots \cdot \Omega_{j-1} \cdot W_1^* \cdot \Omega^* \cdot W_2^* \cdot \Omega_{j+1} \cdot \ldots \cdot W_q$. In this decomposition, the overall number of Ω's has not changed but, since $|W_1^*| + |W_2^*| < |W_{j-1}| + |W_j|$, the sum of the lengths of the W_i's has decreased. This contradicts our choice of C.

Case 2: Otherwise. In this case, $W_{j-1} \cdot W_j$ is a simple walk. Thus, the number of Ω's has decreased by one while $\sum_{i=1}^{q} |W_i|$ remains the same. This contradicts our choice of C.

Since both cases lead to a contradiction to the choice of C we can assume that for each Ω_j in C there is either one arc a_j that is not contained in any other Ω_p, $p \neq j$, or there is no other Ω_i of the same length as Ω_j. By pigeonhole principle,there can be at most $|A| = m$ cycles Ω_j for which the first condition is true. For all further cycles, the first condition is false. Now since each Ω_j has length at most n there can be, again by pigeonhole principle, at most n further cycles for which the second condition is true. This implies that $q \leq m + n$. □

The following lemma shows that the cycle lengths in a decomposition suffice to determine the possible overall cycle length.

Lemma 6. *A graph G has a covering cycle C of length ℓ if and only if it has a covering cycle C' with cycle-walk decomposition $\Omega_1 \cdot W_1 \cdot \ldots \cdot \Omega_q \cdot W_q$ such that*

1. *C' has length $x \leq 2n(m+n)$, and*
2. *there are nonnegative integers p_i, $1 \leq i \leq q$, such that $x + \sum_{1 \leq i \leq q} p_i \cdot |\Omega_i| = \ell$.*

We now bound the running time for determining the existence of such a cycle.

Theorem 1. *Let $G = (V, A)$ be a directed graph with n vertices and m arcs and let ℓ be an integer. Then, in $O(8^m \cdot 2^n) \cdot \text{poly}(n + \log \ell)$ time we can determine whether G contains a covering cycle of length exactly ℓ.*

3 An Algorithm for the Parameters n and d

We now describe our first algorithm for Sound Covering Cycle. The running time of this algorithm is exponential in n and d. Thus, we avoid a combinatorial explosion in the number ℓ which is at least as large as d and usually much larger.

The algorithm exploits that in a sound walk, parts with distance more than d are "independent" with respect to the soundness property. To make the argument more precise, consider a yes-instance (G, d, ℓ) of Sound Covering Cycle with a solution cycle $C = W_1 \cdot W_2 \cdot \ldots \cdot W_q \cdot W^*$, where $|W_i| = d$ for $1 \leq i \leq q$ and $|W^*| = \ell \mod d$. For each vertex v_j in W_i the vertex that is relevant to determine whether v_j is sound is contained in W_{i+1}. Thus, consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ which contains each length-d walk as a vertex. In particular, $\mathcal{G}$ contains each W_i. Moreover, assume that $\mathcal{G}$ contains an arc (W, W') if $W \cdot W'$ is a walk in G which is sound for all positions in W. Then $(W_i, W_{i+1}) \in \mathcal{A}$ for each $i < q$. Consequently, the walk $W_1 \cdot W_2 \cdot \ldots \cdot W_q$ in G corresponds to a walk $\mathcal{W}$ in $\mathcal{G}$ and $\mathcal{W} \cdot W^*$ is a sound covering cycle of length ℓ in G.

The algorithm outline hence is as follows: First construct the graph $\mathcal{G}$, called *walk graph* from now on. Second, compute "candidate" walks in $\mathcal{G}$. Finally, check for each candidate walk, whether there is some short walk W^* such that concatenating W^* at its end gives a sound covering cycle of the correct length.

Theorem 2. Sound Covering Cycle *can be solved in* $O(8^{n\cdot\Delta}\cdot 2^{n\cdot\Delta^d})\cdot\text{poly}(n\cdot\Delta^d+\log\ell)$ *time where* Δ *is the maximum outdegree of* G.

Proof. We describe each of the three main steps of the algorithm in detail and then bound its running time.

Constructing the walk graph $\mathcal{G}$. First, enumerate all walks of length d in G. Let $\mathcal{V}$ denote the set of these walks and make $\mathcal{V}$ the vertex set of $\mathcal{G}$. Now construct the arc set $\mathcal{A}$ of $\mathcal{G}$ as follows. For each pair of vertices W and W' in $\mathcal{V}$, check whether $W\cdot W'=(v_1,\ldots,v_{2d})$ is a walk in G and whether it is sound for each v_i, $1\leq i\leq d$. That is, check whether $(v_d,v_{d+1})\in A$ and whether $\mathcal{R}(v_i)=\mathcal{L}(v_{i+d})$ for each v_i, $1\leq i\leq d$. If this is the case, then add the arc (W,W') to $\mathcal{G}$; otherwise, do not add this arc. This completes the construction of $\mathcal{G}$. Now "almost" sound walks in G correspond to walks in $\mathcal{G}$.

> *Observation 1:* A walk $W_1\cdot\ldots\cdot W_i$ of length $d\cdot i$ in G with $|W_j|=d$, $1\leq j\leq i$, is sound for all of its first $d\cdot(i-1)$ positions $\Leftrightarrow$ $(W_1,\ldots,W_i)$ is a walk in $\mathcal{G}$.

Dynamic programming. Now, for a walk $(W_1,\ldots,W_i)$ in $\mathcal{G}$, let $A(W_1,\ldots,W_i)$ denote the arcs of $W_1\cdot\ldots\cdot W_i$ in G. Moreover, for an arc (W,W') in $\mathcal{G}$ with $W=(v_1,\ldots,v_d)$ and $W'=(v_{d+1},\ldots,v_{2d})$ let $\text{arc}(W,W')$ denote the arc (v_d,v_{d+1}) in G (by the construction of the walk graph, this arc is present in G).

Following the discussion above, we now solve Sound Covering Cycle by determining whether there is a walk $(W_1,\ldots,W_q)$ of length $q:=\lfloor\ell/d\rfloor$ in $\mathcal{G}$ and a walk W^* of length $\ell \mod d$ in G such that 1) $W_q\cdot W^*\cdot W_1$ is a walk of length $2d+(\ell \mod d)$ in G which is sound for its first $d+(\ell \mod d)$ positions, and 2) every arc of G is contained in $A(W_1,\ldots,W_q)$ or in $W_q\cdot W^*\cdot W_1$. This is done by a dynamic programming algorithm that fills a table $\mathcal{T}$ with entries of the type $\mathcal{T}[W,W',A',\Lambda,y]$ where

- W and W' are vertices of $\mathcal{G}$,
- A' is a subset of A (note that A is the arc set of G not of $\mathcal{G}$),
- Λ is a subset of $\{1,\ldots,|\mathcal{V}|\}$, and
- y is a nonnegative integer of value at most $|\mathcal{V}|\cdot(|\mathcal{V}|+|\mathcal{A}|)$.

Each entry in $\mathcal{T}$ is either true or false. The aim of the algorithm is to fill the table $\mathcal{T}$ such that $\mathcal{T}[W,W',A',\Lambda,y]$ is true if and only if $\mathcal{G}$ contains a walk $(W,\ldots,W')$ with cycle-walk decomposition $\Omega_1\cdot\Psi_1\cdot\ldots\cdot\Omega_i\cdot\Psi_i$ such that

- $A(W,\ldots,W')=A'$, that is, the walk $W\cdot\ldots\cdot W'$ in G contains exactly the arcs of A',
- $\Lambda=\{|\Omega_j|\mid 1\leq j\leq i\}$, and

– $(W, \ldots, W')$ has length y.

The idea behind $\mathcal{T}$ is that, by Lemma 6, it suffices to consider walks of length at most $|\mathcal{V}| \cdot (|\mathcal{V}| + |\mathcal{A}|)$ and then to extend them by using the cycle lengths in Λ. In a preprocessing, we compute a table $\mathcal{D}$. For the correctly filled table $\mathcal{D}$, an entry $\mathcal{D}[W, W', A', y]$ is true if and only if $\mathcal{G}$ contains a walk $(W, \ldots, W')$ of length y such that $A(W, \ldots, W') = A'$. The table is filled for all $A' \subseteq A$ and for increasing $y < |\mathcal{V}|$. Initially, set $\mathcal{D}[W, W, A(W), 1]$ to true for each $W \in \mathcal{V}$. Then the recurrence for $\mathcal{D}$ is

$$\mathcal{D}[W, W', A', y] := \begin{cases} & \text{if } \exists (\tilde{W}, W') \in \mathcal{A}, \tilde{A} \subseteq A' : \\ \text{true} & \quad \tilde{A} \cup \{\text{arc}(\tilde{W}, W')\} \cup A(W') = A' \wedge \\ & \quad \mathcal{D}[W, \tilde{W}, \tilde{A}, y-1], \\ \text{false} & \text{otherwise.} \end{cases}$$

After $\mathcal{D}$ is completely filled, compute the table $\mathcal{T}$. The recurrence is

$$\mathcal{T}[W, W', A', \Lambda, y] := \begin{cases} \text{true} & \text{if } \Lambda = \{y\} \wedge \mathcal{D}[W, W', A', y] \wedge (W', W) \in \mathcal{A}, \\ & \text{if } \exists (\tilde{W}, W') \in \mathcal{A}, \tilde{A} \subseteq A' : \\ \text{true} & \quad \tilde{A} \cup \{\text{arc}(\tilde{W}, W')\} \cup A(W') = A' \wedge \\ & \quad \mathcal{T}[W, \tilde{W}, \tilde{A}, \Lambda, y-1], \\ & \text{if } \exists \mathcal{T}[W, \tilde{W}, \tilde{A}, \tilde{\Lambda}, y-z], \mathcal{D}[\tilde{W}, W', A^*, z] : \\ & \quad \mathcal{T}[W, \tilde{W}, \tilde{A}, \tilde{\Lambda}, y-z] \wedge \mathcal{D}[\hat{W}, W', A^*, z] \wedge \\ \text{true} & \quad (\tilde{W}, \hat{W}) \in \mathcal{A} \wedge \tilde{A} \cup \{\text{arc}(\tilde{W}, \hat{W})\} \cup A^* = A' \wedge \\ & \quad \Lambda \setminus \{z\} \subseteq \tilde{\Lambda} \subseteq \Lambda, \\ \text{false} & \text{otherwise.} \end{cases}$$

Determining the possible lengths for candidate entries. The next step is to compute for each entry whether it can be extended, by repeating cycles of the cycle-walk decomposition, to obtain a walk of length $\ell - (\ell \mod d)$.

This is done by reducing to Money Changing [2]. More precisely, for each true entry $\mathcal{T}[W, W', A', \Lambda, y]$, we check whether there is a walk of length $\ell - (\ell \mod d)$ that can be obtained from repeating the simple cycles in a length-y walk $\mathcal{W}$ whose existence is implied by the table entry. The set of different lengths of simple cycles in $\mathcal{W}$ is $\Lambda = \{\lambda_1, \ldots, \lambda_{|\Lambda|}\}$. Thus, determining the existence of a cycle of length $\ell - (\ell \mod d)$ is equivalent to checking whether the equation

$$p_1\lambda_1 + p_2\lambda_2 + \ldots + p_{|\Lambda|}\lambda_{|\Lambda|} = \ell - (\ell \mod d) - y$$

has a solution in which each p_i is a nonnegative integer. Each table entry for which the above equation has such a solution is labeled as *candidate entry*. By Lemma 6, the existence of a walk of length $\ell - (\ell \mod d)$ implies that there is a corresponding candidate entry.

Closing the cycle. The final step of the algorithm is to check whether any of the candidate entries can be completed to a sound cycle of length ℓ by adding

a "short" walk of length $\ell \mod d$. To this end, first enumerate all walks W^* of length $\ell \mod d$ in G. Now, for each candidate entry $\mathcal{T}[W, W', A', \Lambda, y]$ and each W^* and for each enumerated short walk W^* do the following. Check whether $W' \cdot W^* \cdot W$ is a walk in G. If yes, then $W \cdot \ldots \cdot W' \cdot W^*$ corresponds to a cycle C in G. This cycle has length $y + (\ell \mod d)$ but since we consider only candidate entries in $\mathcal{T}$, the walk $W \cdot \ldots \cdot W'$ which has length y can be extended to one of length $\ell - (\ell \mod d)$. Thus, the existence of C implies the existence of a cycle C' of length ℓ in G. It remains to check whether C' is sound and covering. To check whether C' is sound it is sufficient to check whether the walk $W' \cdot W^* \cdot W$ is sound for its first $d + (\ell \mod d)$ positions (as every walk in $\mathcal{G}$ corresponds to a walk in G whose positions are sound except for the last d). Finally, it remains to check whether C' is covering. This is done by checking whether $A = A' \cup \{(w'_d, w_1^*), (w^*_{\ell \mod d}, w_1)\} \cup A(W^*)$ where w'_d is the last vertex of W' and w_1 is the first vertex of W in G. If yes, C' is a solution. If none of the combinations of candidate entry and W^* yields a solution, then the instance is a no-instance.

Running time analysis. By Lemma 1 the number of different walks of length d in G is at most $n \cdot \Delta^{d-1}$ and thus $|\mathcal{V}| \leq n \cdot \Delta^{d-1}$. Consequently, the construction of $\mathcal{G}$ can be performed in $\text{poly}(n \cdot \Delta^{d-1})$ time.

The running time in the dynamic programming part is dominated by the time for filling the table $\mathcal{T}$. This is dominated by the time needed to check whether the third case of the recursion applies. To do this, one needs to consider, for each table entry, $O(|\mathcal{V}|^2 \cdot 4^{n \cdot \Delta})$ possibilities (recall that $n \cdot \Delta \geq |A|$). For each possibility, the check can be performed in $\text{poly}(|\mathcal{V}|)$ time and there are $O(|\mathcal{V}|^2 \cdot 2^{n \cdot \Delta} \cdot 2^{|\mathcal{V}|} \cdot |\mathcal{V}| \cdot (|\mathcal{V}| + |\mathcal{E}|)) = 2^{n \cdot \Delta} \cdot 2^{|\mathcal{V}|} \cdot \text{poly}(|\mathcal{V}|)$ table entries to compute. Thus, the overall running time in the dynamic programming part is $8^{n \cdot \Delta} \cdot 2^{|\mathcal{V}|} \cdot \text{poly}(|\mathcal{V}|)$.

Next, we solve $O(2^{n \cdot \Delta} \cdot 2^{|\mathcal{V}|})$ Money Changing instances, each in $\text{poly}(|\mathcal{V}| + \log \ell)$ time [2]. The checks in the final stage require $\text{poly}(|\mathcal{V}|)$ time for each candidate entry since less than $|\mathcal{V}|$ different short walks are considered and each check needs $\text{poly}(|\mathcal{V}|)$ time. The overall running time follows. □

4 Shortest Sound Cycles and Approximately Sound Cycles

The idea described above can be used in several problem variants. One possibility is to find a shortest sound covering cycle instead of one with fixed length.

Shortest Sound Covering Cycle
Input: A paired de Bruijn graph $G = (V, A)$ and a nonnegative integer d.
Task: Find a sound covering cycle of minimum length in G.

We first bound the length of a shortest sound covering cycle G.

Lemma 7. *Let $G = (V, A)$ be a directed graph with maximum outdegree Δ. If G has a sound covering cycle, then it has a sound covering cycle of length at most $2d(n \cdot \Delta + 1) \cdot (n \cdot \Delta^{d-1}) + d$.*

Proof. Consider the walk graph $\mathcal{G}$ of G. This graph has $n \cdot \Delta^{d-1}$ vertices. Now consider a walk $\mathcal{W}$ in $\mathcal{G}$ such that $\mathcal{W} \cdot W^*$ is a sound covering cycle in G, where $|W^*| < d$ and assume that $\mathcal{W}$ has minimum length with this property.

Let $\Omega_1 \cdot W_1 \cdot \ldots \cdot \Omega_q \cdot W_q$ be the cycle-walk decomposition of $\mathcal{W}$ which exists due to Lemma 3. Then, for each Ω_i, $i < q$, there is some $a \in A(\Omega_i)$ which is not contained in any Ω_j, $j \neq i$, otherwise removing Ω_i from $\mathcal{W}$ yields a shorter walk in $\mathcal{G}$ whose corresponding walk in G also covers all arcs. By pigeonhole principle this implies $q \leq |A|+1$. Thus, the length of $\mathcal{W}$ in $\mathcal{G}$ is at most $2(|A|+1)\cdot(n\cdot\Delta^{d-1})$ as each Ω_i and W_i are simple walks in $\mathcal{G}$. The corresponding walk in G has length at most $2d(|A|+1)\cdot(n\cdot\Delta^{d-1})$ since each vertex of $\mathcal{G}$ corresponds to a length-d walk in G. The overall bound now follows from the fact that $|A| \leq n \cdot \Delta$ and $|W^*| < d$. □

Using this bound, we now derive a dynamic programming algorithm that computes walks of increasing length in G. In this computation, we store the last d vertices of the walk, since these have influence on the soundness condition. Moreover, we store which arcs of G are already covered by the walk.

Theorem 3. Shortest Sound Covering Cycle *can be solved in* $O(n^4 \cdot d \cdot \Delta^{3d})$ *time.*

The second variant relaxes the soundness constraint. This is motivated by the fact that the insert size between the paired end reads is not always exactly d. This relaxed notion of soundness is defined as follows. Recall that a cycle in a paired de Bruijn graph spells two strings s and t. Now, a length-ℓ cycle C is called *x-approximately sound* if

$$\forall i \in \{1 \leq i \leq\} : t[i \mod \ell] \in \{s[i+d-x \mod \ell], \ldots, s[i+d \mod \ell]\}.$$

Informally, this means that the right label has distance at least $d-x$ and distance at most d to the left label. This definition leads to the following problem.

Approximately Sound Covering Cycle
Input: A paired de Bruijn graph $G = (V, A)$ and nonnegative integers d, x, and ℓ.
Question: Does G contain an x-approximately sound covering cycle of length ℓ?

We slightly modify the algorithm for Sound Covering Cycle to obtain the following.

Theorem 4. Approximately Sound Covering Cycle *can be solved in* $O(8^{n\cdot\Delta}\cdot 2^{n\cdot\Delta^d})\cdot\mathrm{poly}(n\cdot\Delta^d+\log\ell)$ *time.*

Finally, we consider the combination of finding a short *and* approximately sound cycle. In this variant, we can even allow a number y of mismatches, that is, there can be y positions that are not approximately sound. More formally, a length-ℓ walk W is called *x-approximately sound with cost at most y* if there is a set $M \subseteq \{1, \ldots, \ell\}$ of size at most y such that

$$\forall i \in \{1, \ldots, \ell\} \setminus M : t[i \mod \ell] \in \{s[i+d-x \mod \ell], \ldots, s[i+d \mod \ell]\}.$$

The Cost-bounded Shortest Approximately Sound Covering Cycle problem now is to find a shortest covering cycle that is x-approximately sound with cost at most y (if such a cycle exists).

To obtain an algorithm for this variant, first note that the bound of Lemma 7 also holds for shortest approximately sound cycles of bounded cost: the replacement argument in the proof only removes cycles in the walk graph which does not increase the cost of the solution.

Theorem 5. Cost-bounded Shortest Approximately Sound Covering Cycle *can be solved in* $O(n^4 \cdot d^2 \cdot \Delta^{3d})$ *time.*

5 A Tractable Special Case for the Parameter n

Finally, we present an $f(n) \cdot \text{poly}(\log \ell)$-time algorithm for a special case of Sound Covering Cycle. To describe the structure of this special case, we introduce the following notion: the *compatibility graph* of a paired de Bruijn graph $G = (V, A)$ is a graph $H = (V, B)$ such that $(a, b) \in B$ if $\mathcal{L}(a) = \mathcal{R}(b)$.

We now exploit this structure by presenting an algorithm for the case that H is a union of loops. Then, each pair of vertices with distance d within a sound cycle is identical. Due to this periodic behavior, we obtain the following relationship between sound cycles and shorter covering cycles.

Lemma 8. *Let G be a paired de Bruijn graph such that its compatibility graph H is a union of loops. Then, G has a sound covering cycle of length ℓ with shift d if and only if G has a covering cycle of length* $\gcd(d, \ell)$.

Here, $\gcd(d, \ell)$ denotes the greatest common divisor of d and ℓ.

Theorem 6. Sound Covering Cycle *can be solved in* $O(8^{n \cdot \Delta} \cdot 2^n) \cdot \text{poly}(n + \log \ell)$ *time if the compatibility graph H of G is a union of loops.*

Proof. By Lemma 8 the problem reduces to one of finding a covering cycle of the correct length $\ell' \leq \ell$. Since G has n vertices and at most $n \cdot \Delta$ arcs, this can be done in $O(8^{n \cdot \Delta} \cdot 2^n) \cdot \text{poly}(n + \log \ell)$ time by Theorem 1. □

6 Outlook

It would be clearly desirable to improve the presented algorithms. Any substantial improvement would need to avoid the enumeration of all length-d walks in G. Also it would be interesting to extend the algorithm for the case that H is a disjoint union of loops, for example to the case that every vertex in H has outdegree one. Finally, in a subroutine we consider the problem of computing a covering cycle of fixed length. It is open whether this problem is NP-hard or solvable in polynomial time.

References

[1] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012.

[2] S. Böcker and Z. Lipták. A fast and simple algorithm for the money changing problem. *Algorithmica*, 48(4):413–432, 2007.

[3] D. Earl, K. Bradnam, J. S. John, A. Darling, D. Lin, J. Fass, H. O. K. Yu, V. Buffalo, D. R. Zerbino, M. Diekhans, et al. Assemblathon 1: A competitive assessment of de novo short read assembly methods. *Genome Research*, 21(12):2224–2241, 2011.

[4] J. Edmonds and E. L. Johnson. Matching, Euler tours and the chinese postman. *Mathematical Programming*, 5(1):88–124, 1973.

[5] N. Haiminen, D. N. Kuhn, L. Parida, and I. Rigoutsos. Evaluation of methods for de novo genome assembly from high-throughput sequencing reads reveals dependencies that affect the quality of the results. *PLOS ONE*, 6 (9):e24182, 2011.

[6] R. M. Idury and M. S. Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2(2):291–306, 1995.

[7] E. Kapun and F. Tsarev. On NP-hardness of the paired de bruijn sound cycle problem. In *Proc. 13th WABI*, volume 8126 of *LNCS*, pages 59–69. Springer, 2013.

[8] P. Medvedev, S. Pham, M. Chaisson, G. Tesler, and P. Pevzner. Paired de bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *Journal of Computational Biology*, 18(11):1625–1634, 2011.

[9] P. A. Pevzner, H. Tang, and M. S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proccedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

[10] A. D. Prjibelski, I. Vasilinetc, A. Bankevich, A. Gurevich, T. Krivosheeva, S. Nurk, S. Pham, A. Korobeynikov, A. Lapidus, and P. A. Pevzner. ExSPAnder: a universal repeat resolver for DNA fragment assembly. *Bioinformatics*, 30(12):i293–i301, 2014.

[11] S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts, et al. GAGE: A critical evaluation of genome assemblies and assembly algorithms. *Genome Research*, 22(3):557–567, 2012.

[12] N. Vyahhi, A. Pyshkin, S. Pham, and P. A. Pevzner. From de Bruijn graphs to rectangle graphs for genome assembly. In *Proc. 12th WABI*, pages 249–261. Springer, 2012.

[13] W. Zhang, J. Chen, Y. Yang, Y. Tang, J. Shang, and B. Shen. A practical comparison of de novo genome assembly software tools for next-generation sequencing technologies. *PLOS ONE*, 6(3):e17915, 2011.