Computability of Models for Sequence Assembly

Paul Medvedev[1], Konstantinos Georgiou[1], Gene Myers[2], and Michael Brudno[1]

[1]University of Toronto, Canada [2]Janelia Farms, Howard Hughes Medical Institute, USA
{pashadag,cgeorg,brudno}@cs.toronto.edu, myersg@janelia.hhmi.org

Abstract. Graph-theoretic models have come to the forefront as some of the most powerful and practical methods for sequence assembly. Simultaneously, the computational hardness of the underlying graph algorithms has remained open. Here we present two theoretical results about the complexity of these models for sequence assembly. In the first part, we show sequence assembly to be NP-hard under two different models: string graphs and de Bruijn graphs. Together with an earlier result on the NP-hardness of overlap graphs, this demonstrates that all of the popular graph-theoretic sequence assembly paradigms are NP-hard. In our second result, we give the first, to our knowledge, optimal polynomial time algorithm for genome assembly that explicitly models the double-strandedness of DNA. We solve the Chinese Postman Problem on bidirected graphs using bidirected flow techniques and show to how to use it to find the shortest double-stranded DNA sequence which contains a given set of k-long words. This algorithm has applications to sequencing by hybridization and short read assembly.

1 Introduction

Most current technologies for sequencing genomes rely on the shotgun method – the genome (or its portion) is broken into many small segments (reads) whose sequence is then determined. The problem of combining these reads to reconstruct the source genome is known as sequence (or genome) assembly, and is one of the fundamental algorithmic problems within bioinformatics. One basic assumption made by assembly algorithms is that every read in the input must be present in the original genome. This follows from the fact that it was read from the genome. Motivated by parsimony, some methods made another, less justifiable assumption: that the original genome should be the shortest sequence that contains every read as a substring. This assumption lead to the casting of the genome assembly problem as the Shortest Common Superstring (SCS) problem, which is known to be NP-hard [4].

The problem of modeling genome assembly as the SCS problem is that most genomes have repeats – *multiple* identical, or nearly identical, stretches of DNA, while the SCS solution would represent each of these repeats only once in the assembled genome. This problem is known as over-collapsing the repeats. One way of solving this problem is to build representative strings or structures for each repeat, and allow the assembly algorithm to use these multiple times. Pevzner et al. [12] had the insight that by dividing the reads into shorter k-long stretches (called k-mers), all of the instances of a repeat collapse into a single set of vertices. They represent each read as a walk on a de Bruijn graph (defined below), and the assembly could then be represented as a superwalk – a

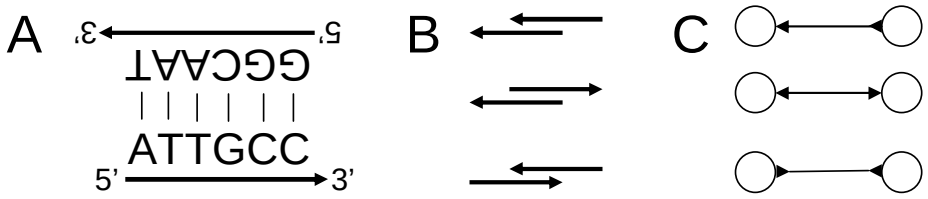


Fig. 1. A. An example of double stranded DNA. The sequence read from this DNA can be either ATTGCC or GGCAAT. **B**. Three possible types of overlaps between two reads: each read can be in either of two orientations, but two of the cases (both to the left and both to the right) are symmetric. **C**. The three corresponding types of bidirected edges. The left node corresponds to the lower read. Note that the arrow points into a node if and only if the overlap covers the start (5') of the read.

walk that includes all of the input walks. In this formulation every edge of the de Bruijn graph has to be present in any solution and can be used multiple times. The solution to the assembly problem is formulated as a variation on finding an Eulerian tour, and because the Eulerian tour problem is solvable in polynomial time this lead to the hope of a polynomial algorithm for sequence assembly. This approach was later expanded to A-Bruijn graphs [13], where the initial subdivision into k-mers is not necessary, but the basic algorithmic problem of searching for a superwalk remains.

Myers [10] provides for an alternative model of sequence assembly, using a string graph. Instead of dividing the reads into k-mers, he builds an overlap graph – a graph where nodes correspond to reads and edges correspond to overlaps (the prefix of one read is the suffix of the other). Through the process of removing redundant edges he is able to classify all edges as either required or optional, and the goal of the assembly is to find the shortest walk which includes all of the required edges. The main algorithmic difference between the de Bruijn / A-Bruijn and the string graph models for sequence assembly is that while in the latter some edges are required and others are optional, in the former all edges are required, but walks have been pre-specified and must be included in the solution. In our first result, we show that sequence assembly with both string graphs and de Bruijn graphs is NP-hard by reduction from Hamiltonian Cycle and Shortest Common Superstring, respectively. Together, these two proofs demonstrate that both of the popular graph-theoretic sequence assembly paradigms are unsolvable by optimal polynomial-time algorithms unless $\mathcal{P} = \mathcal{NP}$.

Another algorithmic problem faced by assembly algorithms is the treatment of double-stranded DNA (see Figure 1A). A DNA molecule consists of two strands which are reverse compliments of each other. The start (called 5') of one strand is complementing the end (called 3') of the other. Whenever DNA is sequenced, the molecule is always read in the same direction, from 5' to 3', but it is impossible to know from which of the two strands the sequence is read. Many sequence assembly algorithms use heuristics to determine the strand for each read. The EULER method [12] uses both the reads and their reverse-complements to build the de Bruijn graph and searches heuristically for two "complementary" paths. In the work of Kececioglu and Myers [6] strand selection for a read is formulated as the NP-hard maximum weight cut problem.

In 1992, Kececioglu [8] introduced an elegant method for dealing with double-strandedness by modeling overlaps between DNA molecules using a bidirected graph. Each read is represented by a single node, and each overlap (edge) has an orientation at both endpoints. The three types of bidirected edges correspond to the three possible ways in which the overlap can occur (see Figure 1B & C). Bidirected graphs were further used for sequence assembly in [9, 10] and to model breakpoint graphs in [7]. Remarkably, however, bidirected graphs have been studied within the context of graph theory already in the 1960s when Edmonds formulated the problem of bidirected flow (a generalization of network flow to bidirected graphs) and showed it equivalent to perfect b-matchings [1]. Edmonds' work was later extended by Gabow [3], who gave the fastest to-date algorithm for bidirected flow. In our second result, we extend Gabow's and Edmonds' work to give a polynomial time algorithm for solving the Chinese Postman Problem in bidirected graphs. By combining this algorithm with Pevzner's work on de Bruijn graphs [11, 12] and Kececioglu's work on modeling strandedness with bidirected graphs [8], we show how it can be used to find the shortest (double-stranded) DNA sequence with a given set of k-long DNA fragments. To the best of our knowledge, this is the first optimal polynomial time assembly algorithm which explicitly deals with the double-stranded nature of DNA.

2 Preliminaries

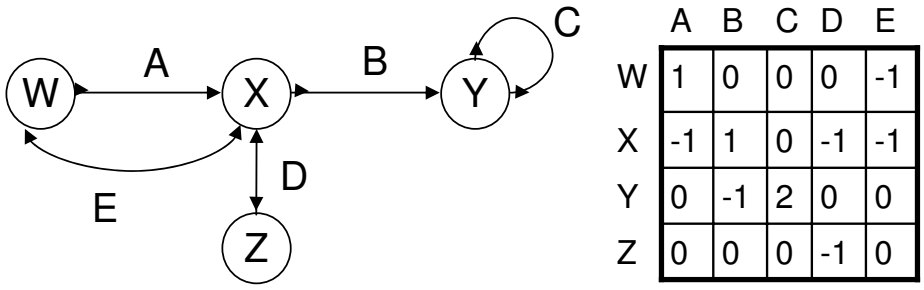
In this section, we give the background and definitions needed for the rest of this paper.

2.1 Strings, Overlap Graphs, de Bruijn Graphs, and Molecules

Let v and w be two strings over the alphabet Σ. The concatenation of these strings is denoted as $v \cdot w$. The length of v is denoted by $|v|$. The i^{th} character of v is denoted by $v[i]$. If $1 \leq i \leq j \leq |v|$, then $v[i, j]$ is the substring beginning at the i^{th} position and ending at the j^{th} position, inclusive. If there exists i, j such that $v = w[i, j]$, then we say v is a **substring** of w. For $x \in \Sigma$, x^k is x concatenated with itself k times if $k \geq 1$, and ϵ otherwise. A string of length k is called a **k-mer**. The **k-spectrum** of v is the set of all k-mers that are substrings of v. A **k-molecule** is a pair of k-mers which are reverse compliments of each other. We say a k-molecule **corresponds** to each of its two constitutive k-mers. The **k-molecule-spectrum** of a DNA molecule is the set of all k-molecules corresponding to the k-mers of the k-spectrum of either of the DNA strands.

We say w **overlaps** v if there exists a maximal length non-empty string z which is a prefix of w and a suffix of v (notice this definition is not symmetric). The length of the overlap is $ov(v, w) = |z|$. If w does not overlap v then $ov(v, w) = 0$. Let $S = \{s_1, \ldots, s_n\}$ be a set of non-empty strings over an alphabet Σ. An **overlap graph** of S is a complete weighted directed graph where each string in S is a vertex and the length of the edge $x \to y$ is $|y| - ov(x, y)$.

We say w is a **superstring** of S if for all i, s_i is a substring of w. The **Shortest Common Superstring (SCS)** problem is to find the shortest superstring of S. It was proven to be NP-hard for $|\Sigma| \geq 2$ [4, 5]. We define the **de Bruijn graph** $\mathbf{B^k(S)}$ as a



	A	B	C	D	E
W	1	0	0	0	-1
X	-1	1	0	-1	-1
Y	0	-1	2	0	0
Z	0	0	0	-1	0

Fig. 2. This is an example of a bidirected graph and its incidence matrix. We draw an edge that is positive incident to a vertex using an arrow that is pointing out of the vertex, but this choice of graphical representation is arbitrary.

directed graph, using a positive integer parameter k. The vertices of $B^k(S)$ are $\{d \in \Sigma^k \mid \exists i$ such that d is a substring of $s_i\}$. We identify a vertex by the k-mer associated with it. We abuse notation here by referring to a vertex in $B^k(S)$ by the k-mer associated with it. The edges are $\{d[1..k] \rightarrow d[2..k+1] \mid d \in \Sigma^{k+1}, \exists i$ such that d is a substring of $s_i\}$.

2.2 Bidirected Graphs and Flow

Consider an undirected (multi) graph G with a set of vertices V and a set of edges E. The **multiplicity** of an edge e is the number of edges in G whose endpoints are the same as e's If the endpoints are distinct, the edge is called a **link**, otherwise it is a **loop**. Additionally, we assign orientations to the edges. Every link has two orientations, one with respect to each of its endpoints, while every loop has one orientation. There are two kinds of orientations – positive and negative – and thus we can say an edge is **positive-incident** or **negative-incident** to an endpoint. When taken together with the orientations of its edges, G is called a **bidirected graph**. If there is additionally a weight function w_e associated with the edges, we say the graph is **weighted**. The weight of a graph is the sum of the weights of its edges. A bidirected graph is **connected** if its underlying undirected graph is connected.

The orientations of the edges can be represented by an **incidence matrix** $I^G : V \times E \longrightarrow \{-2, -1, 0, 1, 2\}$ (we omit G when it is obvious from the context). If an edge e is not incident to a vertex x then $I(x, e) = 0$. For a link e and a vertex x, $I(x, e) = +1$ if e is positive-incident to x, and $I(x, e) = -1$ if e is negative-incident to x. For a loop e and a vertex x, $I(x, e)$ has the value of +2 if e is positive-incident to x, and the value of -2 if e is negative-incident to x. See Figure 2 for an example of a bidirected graph and its incidence matrix. The **in-degree** of a vertex x in graph G is defined as $deg^-_G(x) = -\sum_{\{e \in E \mid I(x,e)<0\}} I(x, e)$. Similarly, the **out-degree** is defined as $deg^+_G(x) = \sum_{\{e \in E \mid I(x,e)>0\}} I(x, e)$. Let $bal^G(x) = deg^+_G(x) - deg^-_G(x) = \sum I(x, e)$ be the **balance** at each vertex. G is **balanced** if the balance of each vertex is 0.

A (x_1, x_k)-**walk** is a sequence $x_1, e_1, \ldots, x_{k-1}, e_{k-1}, x_k$ where e_i is an edge incident to x_i and x_{i+1}, and for all $2 \leq i \leq k-1$, e_{i-1} and e_i have opposite orientations at x_i. Since the specification of vertices is redundant, we may omit them sometimes and specify a walk as just a sequence of edges. A walk is said to be **cyclical** if its endpoints

are the same and e_1 and e_{k-1} have opposite orientations at x_1. A bidirected graph is **strongly connected** if it is connected and for every edge there is a cyclical walk containing it.

Note that we can view a loopless directed graph as a special kind of bidirected graph, where every edge is positive-incident to one of its endpoints and negative-incident to the other one. In this case, the definition of a walk reduces to its usual meaning in directed graphs. However, there are some caveats. For example, it is possible for the shortest walk between two vertices to repeat a vertex in a bidirected graph. In Figure 2, observe that there does not exist a walk between W and Z which does not repeat a vertex, something that is not possible in a directed graph.

A **Chinese walk** is a cyclical walk that traverses every edge at least once. Given a weighted bidirected graph, the **Chinese Postman Problem(CPP)** is to find a minimum weight Chinese walk (called a **Chinese Postman Tour**), or report that one doesn't exist. An **Eulerian tour** of a graph is a cyclical walk that contains every edge of the graph exactly once, and a graph which contains an Eulerian tour is called **Eulerian**. The following is a generalization of a well-known fact for directed graphs whose proof is almost identical to the directed case and is therefore ommited.

Lemma 1. *A bidirected graph G contains an Eulerian tour if and only if it is connected and balanced.*

Given a bidirected graph G, and vectors $a, b \in \mathbb{Z}^{V(G)}$ and $d, c, w \in \mathbb{Z}^{E(G)}$, a **minimum cost bidirected flow** problem [14] is an integer linear program where the goal is to find $x \in \mathbb{Z}^{E(G)}$ that minimizes $w \cdot x$ subject to the constraints that $d \leq x \leq c$ and $a \leq I^G \cdot x \leq b$. Here, $\cdot$ refers to the inner product between two vectors, and $\leq$ is a component-wise comparison operator.

3 The String Graph Framework

In [10], Myers introduces a string graph framework for sequence assembly. A string graph is built from an overlap graph through the process of transitively inferable edge reduction – whenever y and z overlap x, and z overlaps y, the overlap of z to x is said to be inferable from the other two overlaps, and is removed from the graph. Myers demonstrates a fast algorithm for removing transitively inferable edges from the graph, which, in combination with statistical methods, associates a "selection" constraint with each edge. The selection constraint states that the edge must appear in the target genome either at least once (it is *required*), exactly once (it is *exact*), or any number of times (it is *optional*). The key property of string graphs is that any cyclical walk that respects the selection constraints represents a valid assembly of the genome, and the weight of the walk is the length of the assembled genome. After building the string graph, the algorithmic problem is to find a cyclical walk that visits each edge in accordance with its selection constraint. Appealing to parsimony, the goal is to find a walk with minimum weight. In this section, we show that this problem is NP-hard.

Formally, a *selection* function s is a function that classifies each edge into one of three categories: *optional, required, exact*. We call a walk which contains all the required edges at least once, all the exact edges exactly once, and all the optional

edges any number of times an **s-walk**. The **Minimum s-Walk Problem(MSWP)** for a weighted directed graph G and a selection function s is the problem of finding a minimum weight cyclical s-walk of G, or report that one doesn't exist.

Theorem 1. *The Minimum s-Walk Problem is NP-hard.*

The proof works by reducing the Hamiltonian Cycle problem in directed graphs to MSWP. A cycle is Hamiltonian if it visits every vertex exactly once. The reduction works by splitting each vertex into 'in' and 'out' counterparts and adding a required edge between them, while making all other edges optional. Having optional edges is essential for the reduction; if they are not present, the problem can be efficiently solved using a variant of the algorithm of Section 5.1. Also note that in [10] the edges of the string graph are bidirected in order to reflect the double strandedness of DNA. Since directed graphs are a special type of bidirected graphs, Theorem 1 holds for bidirected graphs as well.

Proof. Let G be a directed graph, with vertices $v_1, \ldots, v_n$, for which we wish to find a Hamiltonian cycle. Let G' be a directed graph with vertex set $\{v_i^-, v_i^+ \mid 1 \leq i \leq n\}$ and edge set $O \cup R$, where $O = \{v_i^+ \to v_j^- \mid (v_i \to v_j) \in E(G)\}$ and $R = \{v_i^- \to v_i^+ \mid 1 \leq i \leq n\}$. The weight of each edge is 1. Let s be a selection function on G' that labels all the O edges as optional and all the R edges as required. We show that G has a Hamiltonian cycle if and only if G' has a cyclical s-walk of weight at most $2n$.

First, suppose $C = v_{i_1} \to \ldots \to v_{i_n} \to v_{i_1}$ is a Hamiltonian cycle of G. Then $C' = v_{i_1}^- \to v_{i_1}^+ \to v_{i_2}^- \to v_{i_2}^+ \to \ldots \to v_{i_{n-1}}^- \to v_{i_{n-1}}^+ \to v_{i_1}^-$ is a cyclical s-walk in G' of weight $2n$. For the other direction, let C' be a cyclical s-walk in G' of length at most $2n$. Because the R edges form a matching and all n of them must be in C', the edges of C' must alternate between R and O edges, and thus have a total of n edges of each kind. If we remove all the R edges from C' and map all the vertices of C' to their counterparts in G, we get a Hamiltonian cycle of G. □

4 The de Bruijn Graph Framework.

One of the original graph-theoretic frameworks for sequence assembly was proposed by Pevzner, Tang, and Waterman in [12]. They note that by tiling every read by $(k+1)$-mers they can view the read as a walk in a de Bruijn graph, where the vertices are k-mers and edges are $(k+1)$-mers. Thus, any walk that contains all the reads as subwalks represents a valid assembly. Consequently, they formulate the assembly problem as finding the shortest superwalk, a problem closely related to the polynomial time Eulerian tour problem (which was previously used to solve the problem of sequencing by hybridization [11]). What we show in this section is that the de Bruijn graph framework does not make the problem of read assembly more tractable.

Let $S = \{s_1, \ldots, s_n\}$ be a set of strings over an alphabet Σ and let $G = B^k(S)$ be the de Bruijn graph of S for some k. The strings s_i correspond to walks in $B^k(S)$ via the function $w(s) = s[1..k] \to s[2..k+1] \to \ldots \to s[|s|-k+1, |s|]$. A walk is called a **superwalk** of G if, for all i, it contains $w(s_i)$ as a subwalk. Thus, a superwalk represents a valid assembly of the reads into a genome. Within this framework, the goal

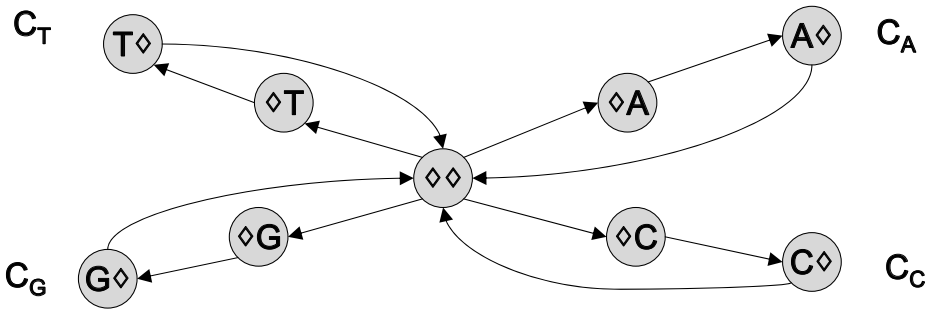


Fig. 3. An example of the reduction from Shortest Common Superstring to De Bruijn Superwalk. The set of strings S is over the alphabet {A,C,G,T}, and the graph drawn is $B^2(f(S))$. The cycles in the edge decomposition are C_A, C_C, C_G, C_T and have three edges each. As an example, the walk $w(f(ATT))$ starts at the central node and is C_A followed by C_T followed by C_T again.

of finding a parsimonious genome assembly is to find a minimal length superwalk. The assembly algorithm of [12] looks for such a superwalk, however, it uses heuristics and may not produce the correct answer.

Formally, given a set of strings S as defined above and a positive integer k, the **De Bruijn Superwalk Problem (BSP)** is to find a minimum length superwalk in $B^k(S)$, or report that one does not exist. Observe that since every edge in $B^k(S)$ is covered by at least one walk $w(s_i)$, a superwalk will traverse every edge at least once. We shall show that BSP is NP-hard by a reduction from the Shortest Common Superstring (SCS) problem. Informally, we will transform a string by inserting $\diamond^k$ in between every character, as well as in the beginning and end, where $\diamond$ is a special character that does not appear in the input strings. For example, we transform the string 'abc' into '$\diamond^k$a$\diamond^k$b$\diamond^k$c$\diamond^k$'. This transformation preserves overlaps and introduces a $\diamond^k$ overlap between otherwise non-overlapping strings. The idea is that while a superstring can be built by appending non-overlapping strings, a superwalk must correspond to a string built by overlaps of at least k characters. See Figure 3 for an illustration of the de Bruijn graph on a set of transformed strings.

Theorem 2. *The De Bruijn Superwalk Problem is NP-hard, for $|\Sigma| \geq 3$ and for any positive integer k.*

Proof. SCS is NP-hard even if the size of the alphabet is 2 [5]. We reduce an instance of SCS to an instance of BSP which has an additional character $\diamond$ in the alphabet. Let $S = \{s_1, \ldots, s_n\}$ be the set of strings of an SCS instance, and Σ be the set of characters appearing in S. We define a function $f(s)[i]$ for $1 \leq i \leq k(|s|+1)+|s|$ as follows: For all i divisible by $k+1$, $f(s)[i] = s[\frac{i}{k+1}]$. For all other i, $f(s)[i] = \diamond$. Let $G = B^k(f(S))$, where $f(S) = \{f(s_i) \mid 1 \leq i \leq n\}$.

We first make some observations about G, which follow directly from the definition of de Bruijn graphs and from f. The vertices of G, which are the k-mers appearing in $f(S)$, are $\{\diamond^k\} \cup \{\diamond^{k-i}x\diamond^{i-1} \mid x \in \Sigma, 1 \leq i \leq k\}$. The edges of G are $\{E_x \mid x \in \Sigma\}$, where $E_x = \{\diamond^k \to \diamond^{k-1}x\} \cup \{x\diamond^{k-1} \to \diamond^k\} \cup \{\diamond^{k-i}x\diamond^{i-1} \to \diamond^{k-i-1}x\diamond^i \mid 1 \leq i \leq k-1\}$. The edge set of G forms a disjoint union of cycles

$\bigcup_{x\in\Sigma} C_x$, where $C_x = \diamond^k \to \diamond^{k-1}x \to \diamond^{k-2}x\diamond \to \ldots \to \diamond x\diamond^{k-2} \to x\diamond^{k-1} \to \diamond^k$. We also note that $w(f(s_i)) = w(\diamond^k s_i[1]\diamond^k \ldots \diamond^k s_i[|s|]\diamond^k) = C_{s_i[1]} \to \ldots \to C_{s_i[|s_i|]}$. For an illustration see Figure 3.

Now we show that the length of the shortest superwalk of G is $k+1$ times the length of the shortest superstring of S. First, suppose s is a superstring of S. Let $w = C_{s[1]} \to \ldots \to C_{s[|s|]}$. We claim that w is a superwalk of G of length $|s|(k+1)$. We have to show that $w(f(s_i))$ is a subwalk of w for all i. Since s_i is a substring of s, there is some j and k such that $s_i = s[j,k]$. Then, $w(f(s_i)) = C_{s[j]} \to \ldots \to C_{s[k]}$, which is indeed a subwalk of w.

Now, suppose w is a superwalk of G. Every edge that appears before the first $\diamond^k$ and after the last $\diamond^k$ in w can be removed from w while preserving it as a superwalk. Therefore, we can assume that the first and last vertex of w is $\diamond^k$, and w can be uniquely expressed as a sequence of cycles $C_{j_1} \to \ldots \to C_{j_{\frac{|w|}{k+1}}}$. Let $s = j_1 \cdot j_2 \cdots j_{\frac{|w|}{k+1}}$. For all i, since $w(f(s_i))$ is a subwalk of w, we can write it as $w(f(s_i)) = C_{j_m} \to \ldots \to C_{j_{m+\frac{|w_i|}{k+1}-1}}$ for some m. By definition, $w(f(s_i)) = C_{s_i[1]} \to \ldots \to C_{s_i[|s_i|]}$. Since the decomposition of a walk into cycles C_x is unique, we conclude that $s_i[k] = j_{m+k-1}$ for $1 \le k \le |s_i|$. Therefore, s_i is a substring of s, and s is a superstring of length $\frac{|w|}{k+1}$. □

5 Assembly of Double-Stranded DNA with Bidirected Flow

In this section, we demonstrate the first, to our knowledge, polynomial algorithm for assembly of a double-stranded genome. First, we give a polynomial time algorithm for solving the Chinese Postman Problem (CPP) on bidirected graphs. Subsequently, we will show how to construct a bidirected de Bruijn graph from the set of k-long molecules that are present in it (the k-molecule-spectrum). By solving the CPP on the resulting graph we are able to reconstruct the shortest DNA molecule with the given k-molecule-spectrum.

5.1 The Bidirected Chinese Postman Problem

Given a weighted bidirected graph G, recall that the Chinese Postman Problem (CPP) is to find a minimum weight Chinese walk of G, or report that one does not exist. CPP is polynomially time solvable on both undirected and directed graphs [2]. It becomes NP-Hard on mixed graphs, which are graphs with both directed and undirected edges [5]. For undirected graphs, CPP is reducible to minimum cost perfect matchings. For directed graphs, it is reducible to minimum cost network flow. In this section, we give an efficient algorithm for solving CPP on bidirected graphs via a reduction to minimum cost bidirected flow.

We will show in Lemma 2 that for G to have a Chinese walk it is necessary and sufficient for it to be strongly connected. To find a min-weight Chinese walk, first consider the case G is Eulerian. An Eulerian tour of G is also a Chinese walk, since it visits every edge exactly once. Furthermore, since any Chinese walk has to visit every edge at least once, the Eulerian tour is also a Chinese postman tour. In the general case, however, when G is not Eulerian, our approach is to make the graph Eulerian by duplicating some

1: **if** G is not connected **then return** "no Chinese walk exists"
2: Use algorithm of [3] to solve the corresponding minimum cost bidirected flow (see text).
3: **if** there is no solution **then return** "no Chinese walk exists"
4: Let G' be the graph G with f_e copies of every edge e, in addition to e itself.
5: Use a standard algorithm to find an Eulerian circuit C of G'.
6: Relabel C according to Theorem 3.
7: **return** C

Fig. 4. Algorithm for the Chinese Postman Problem on bidirected graphs.

of the edges, and then using a standard algorithm to find an Eulerian tour. We shall prove that if we minimize the total weight of the duplicated edges, the Eulerian tour we find in the modified graph will correspond to a Chinese postman tour in the original graph.

Formally, we say a graph G' is an **extension** of G if it can be obtained from G by duplicating some of its edges. The **Eulerization Problem (EP)** is to find a min-weight Eulerian extension of G, or report that one does not exist. The following theorem shows that CPP and EP are polynomially equivalent.

Theorem 3. *There exists a Chinese walk of weight i if and only if there exists an Eulerian extension of weight i. Moreover, they can be derived from each other in polynomial time.*

Proof. For the only if direction, let W be a Chinese walk in G. Let $\widehat{W}$ be the graph induced by W, where the multiplicity of each edge is the number of time it is traversed by W. Then $\widehat{W}$ is an extension of G because W visits every edge at least once. Also W is an Eulerian circuit of $\widehat{W}$ whose weight is that of $\widehat{W}$. Thus $\widehat{W}$ is an Eulerian extension of G with weight of W.

For the if direction, let G' be an Eulerian extension of G. Let W' be an Eulerian circuit in G'. Construct W from W' by replacing every edge $e' \notin G$ by an edge $e \in G$ such that e' is a duplicate of e. W is thus a valid cyclical walk in G which visits every edge at least once and whose weight is the same as that of W' and of G'. □

Now, we give an algorithm for the Eulerization Problem. First, we consider the case that G is not connected. Since any extension of G will also not be connected, our algorithm can safely report that there is no Eulerian extension of G, and hence no Chinese walk. For the case that G is connected, we formulate EP as a min-cost bidirected flow problem. First, we represent an extension G' of G with $|E(G)|$ variables, where each variable f_e represents the number of additional copies of edge e in G'. It is clear that an assignment of non-negative integers to these variables corresponds to an extension of G, and vice-versa. Now, we would like to formulate EP in terms of these variables instead of in terms of an extension. The minimization criterion is the weight of the extension, which is $\sum w_e(1 + f_e)$. The criterion that G' is Eulerian is, by Lemma 1, the criteria that it is connected and balanced. The connectivity criterion is redundant since G is connected and thus any extension of G must also be connected. The balance condition for each vertex x can be stated as: $\sum_e I^G(x, e) \cdot f_e + bal^G(x) = 0$. That is, the balance of x in G' is the balance of x in G plus the contribution of all the copied edges. We are now able to formulate EP as the following integer linear program:

$$\begin{aligned}
&\text{minimize} && \sum w_e f_e \\
&\text{subject to} && \sum_e I^G(x,e) f_e = -bal^G(x) && \text{for each vertex } x \\
& && f_e \geq 0 && \text{for each edge } e
\end{aligned}$$

From the definition in Section 2.2, this is actually a minimum cost bidirected flow problem, which can be solved using Gabow's algorithm [3]. Our final algorithm for CPP on bidirected graphs is given in Figure 4. For the running time, we need to bound the size of the solution:

Lemma 2. *G has an Eulerian extension if and only if it is strongly connected. Moreover, the min-weight Eulerian extension has at most $2|E||V|$ edges.*

Proof. If G has an Eulerian extension, then it must be connected, and for every edge there is a cyclical walk containing it (namely the one induced by the Euler tour). Conversely, suppose that G is strongly connected. For every edge, we can duplicate all the other edges of the shortest cyclical walk that contain it, thus balancing the graph. Now, suppose G' is a min-weight Eulerian extension of G. We can decompose G' into a set of minimal cycles. Each cycle must contain an edge that no other cycle contains, otherwise it can be removed from G' to get a smaller weight extension. Therefore, there are at most $|E|$ cycles, and each cycle contains at most $2|V|$ edges. □

Gabow's algorithm runs in time $O(|E|^2 \log(|V|) \log(C))$, where C is the largest capacity ($C = \max c(e)$ using the definition of Section 2.2). By the above lemma, $C = O(|V|^3)$ if the graph is simple, so the running time for finding the flow, and thus for the whole algorithm, is $O(|E|^2 \log^2(|V|))$.

5.2 The Bidirected de Bruijn Graph

In an earlier work [11], Pevzner showed that the de Bruijn graph B^{k-1} can be used to represent the k-spectrum of a string, and that the (directed) Chinese postman tour on this graph corresponds to the shortest string with the given k-spectrum. When working with double-stranded DNA molecules, however, it is necessary to model k-molecules instead of k-mers in the de Bruijn graph. To do this Pevzner includes both of the k-mers associated with every k-molecule in the de Bruijn graph. He then searches for two "complementary" walks, each corresponding to one of the DNA strands (see Figure 5). Instead, we show how to construct a bidirected de Bruijn graph where each k-molecule is represented only once.

Our input is the k-molecule-spectrum of the genome. We will arbitrarily label one of the k-mers associated with each k-molecule as coming from the "positive" strand and the other from the "negative" strand. Let the nodes of the bidirected de Bruijn graph be all of the possible $(k-1)$-molecules. For every k-molecule in the spectrum, let z be one of its two k-mers. Let x and y be the $(k-1)$-molecules corresponding to $z[1..k-1]$ and $z[2..k]$, respectively. We make an edge between the vertices corresponding to x and y.

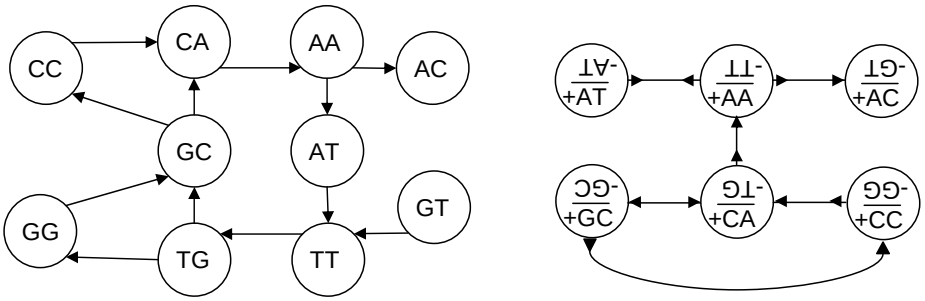


Fig. 5. Given the k-molecule-spectrum {ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT, Pevzner et al.'s [12] approach builds the graph on the left, and searches for two complementary paths. The bidirected de Bruijn graph is on the right; one tour that includes all of the edges spells ATTGCCAAC on the forward strand, and GTTGGCAAT on the reverse.

This edge is positive-incident to x if $z[1..k-1]$ is the positive strand of x, and negative-incident otherwise. It is negative-incident to y if $z[2..k]$ is the positive strand of y, and positive-incident otherwise. Note that this edge construction is identical to the one defined by Kececioglu [8] for an overlap between two DNA molecules (also see Figure 1).

The Chinese postman tour of the resulting bidirected de Bruijn graph corresponds to the shortest assembly of the DNA molecule with the given k-molecule-spectrum. The proof follows from the construction: every k-molecule from the spectrum is represented by exactly one edge in the graph. Every valid assembly of the genome corresponds to a walk in the bidirected de Bruijn graph. Because the Chinese postman tour is the shortest such walk, it is also the shortest assembly of the genome. The tour also corresponds to both of the DNA strands. Because a walk is required to use edges with opposite orientations to enter and leave every vertex, but is allowed to enter on either a positive or negative oriented edge, the Chinese postman tour can be "walked" in either of two directions. If we enter a node on a positive-incident edge we use the positive k-mer, if on the negative incident we use the negative k-mer. The two directions correspond exactly to the two strands of DNA, and the sequences "spelled" by them are reverse-complements. For the running time, because the de Bruijn graph has a constant degree at every node ($|E| \in \Theta(|V|)$), the overall running time is $O(|V|^2 \log^2(|V|))$ using the algorithm of Section 5.1.

6 Discussion

In this work we showed that both the de Bruijn graph and string graph models for sequence assembly are NP-hard. While this result makes it impractical to look for polynomial time exact algorithms for either of these problems, we believe our work suggests two important areas of investigation. The first is to characterize the computational difficulty of the genome assembly models on real-world genomes. It is well known that many NP-hard problems are efficiently solvable when restricted to particular classes of inputs. The success of both the de Bruijn and string graph models in practice indicate

that by defining a more restricted model of inputs that nevertheless covers most actual genomes, we may be able to create a model for sequence assembly that can be solved exactly in polynomial time. Simultaneously, real-life genomes contain repeats, making it unlikely that any real genome will have a unique solution under either string graph or de Bruijn graph assembly models. Consequently it is important to explore what a realistic objective function for an assembly algorithm should be. Conducting a rigorous study of these questions is a promising avenue for improving assembly programs.

In our second result we showed that the computational difficulty of sequence assembly is not due to double-strandedness of DNA. By unifying Pevzner's work on de Bruijn graphs, Kececioglu's and Myers' work on bidirected graphs in assembly and Edmonds' and Gabow's work on bidirected flow, we are able to demonstrate an optimal polynomial time assembly algorithm that explicitly deals with double-strandedness. We believe the use of bidirected flow as a technique will be fruitful for other sequence assembly problems, including for the assembly of short DNA reads coming from novel sequencing technologies such as Illumina and 454.

Acknowlegments

We would like to thank Allan Borodin for helpful comments and careful reading of the manuscript. This work was partially supported by an NSERC Discovery Grant to MB.

References

1. J. Edmonds. An introduction to matching. Notes of engineering summer conference, University of Michigan, Ann Arbor, 1967.
2. J. Edmonds and E.L. Johnson. Matching, Euler tours, and the Chinese postman. *Mathemetical Programming*, 5:88–124, 1973.
3. Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *STOC*, pages 448–456, 1983.
4. John Gallant, David Maier, and James A. Storer. On finding minimal length superstrings. *J. Comput. Syst. Sci.*, 20(1):50–58, 1980.
5. M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
6. John D. Kececioglu and Eugene W. Myers. Combinatiorial algorithms for DNA sequence assembly. *Algorithmica*, 13(1/2):7–51, 1995.
7. John D. Kececioglu and David Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1/2):180–210, 1995.
8. John Dimitri Kececioglu. *Exact and approximation algorithms for DNA sequence reconstruction*. PhD thesis, Tucson, AZ, USA, 1992.
9. Eugene W. Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290, 1995.
10. Eugene W. Myers. The fragment assembly string graph. In *ECCB/JBI*, page 85, 2005.
11. P A Pevzner. 1-Tuple DNA sequencing: computer analysis. *J Biomol Struct Dyn*, 7(1):63–73, Aug 1989.
12. P.A. Pevzner, H. Tang, and M.S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98:9748–9753, 2001.
13. Pavel A. Pevzner, Haixu Tang, and Glenn Tesler. *De novo* repeat classification and fragment assembly. In *RECOMB*, pages 213–222, 2004.
14. Alexander Schrijver. *Combinatorial Optimization*, volume A. Springer, 2003.