



# Connecting de Bruijn Graphs

Giulia Bernardini  

University of Trieste, Trieste, Italy

Inge Li Gørtz  

Technical University of Denmark, Lyngby,  
Denmark

Grigorios Loukides  

King's College London, London, UK

Leen Stougie  

CWI, Amsterdam, The Netherlands



Vrije Universiteit, Amsterdam, The Netherlands

Huiping Chen  

University of Birmingham, Birmingham, UK

Christoffer Krogh  

Technical University of Denmark, Lyngby,  
Denmark

Solon P. Pissis  

CWI, Amsterdam, The Netherlands

Vrije Universiteit, Amsterdam, The Netherlands

Michelle Sweering  

CWI, Amsterdam, The Netherlands

---

## Abstract

We study the problem of making a *de Bruijn graph* (DBG), constructed from a collection of strings, *weakly connected* while minimizing the total cost of edge additions. The input graph is a DBG that can be made weakly connected by adding edges (along with extra nodes if needed) from the underlying complete DBG. The problem arises from genome reconstruction, where the DBG is constructed from a set of sequences generated from a genome sample by a sequencing experiment. Due to sequencing errors, the DBG is never Eulerian in practice and is often not even weakly connected. We show the following results for a DBG  $G(V, E)$  of order  $k$  consisting of  $d$  weakly connected components:

1. Making  $G$  weakly connected by adding a set of edges of minimal total cost is NP-hard.
2. No PTAS exists for making  $G$  weakly connected by adding a set of edges of minimal total cost (unless the *unique games conjecture* fails). We complement this result by showing that there does exist a polynomial-time  $(2 - 2/d)$ -approximation algorithm for the problem.
3. We consider a restricted version of the above problem, where we are asked to make  $G$  weakly connected by *only adding directed paths between pairs of components*. We show that making  $G$  weakly connected by adding  $d - 1$  such paths of minimal total cost can be done in  $\mathcal{O}(k|V|\alpha(|V|) + |E|)$  time, where  $\alpha(\cdot)$  is the inverse Ackermann function. This improves on the  $\mathcal{O}(k|V|\log(|V|) + |E|)$ -time algorithm proposed by Bernardini et al. [CPM 2022] for the same restricted problem.
4. An ILP formulation of polynomial size for making  $G$  Eulerian with minimal total cost.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Pattern matching

**Keywords and phrases** string algorithm, graph algorithm, de Bruijn graph, Eulerian graph

**Digital Object Identifier** 10.4230/LIPIcs.CPM.2024.6

**Funding** *Giulia Bernardini*: Supported by the MUR - FSE REACT EU - PON R&I 2014-2020.

*Inge Li Gørtz*: Supported by the Independent Research Fund Denmark (DFR-9131-00069B and 10.46540/3105-00302B).

*Solon P. Pissis*: Supported by the PANGAIA and ALPACA projects that have received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No 872539 and 956229, respectively.

*Leen Stougie*: Supported by the PANGAIA and ALPACA projects (grant agreements No 872539 and 956229) and by the Netherlands Organisation for Scientific Research (NWO) under projects OCENW.GROOT.2019.015 "Optimization for and with Machine Learning (OPTIMAL)" and Gravitation-grant NETWORKS-024.002.003.

*Michelle Sweering*: Supported by the Netherlands Organisation for Scientific Research (NWO) under the Gravitation-grant NETWORKS-024.002.003.



© Giulia Bernardini, Huiping Chen, Inge Li Gørtz, Christoffer Krogh, Grigorios Loukides, Solon P. Pissis, Leen Stougie, and Michelle Sweering;  
licensed under Creative Commons License CC-BY 4.0

35th Annual Symposium on Combinatorial Pattern Matching (CPM 2024).

Editors: Shunsuke Inenaga and Simon J. Puglisi; Article No. 6; pp. 6:1–6:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

Let us start with some basic definitions and notation following [5]. An *alphabet*  $\Sigma$  is a finite set of elements called *letters*. We consider an integer alphabet  $\Sigma = [0, \sigma)$ . Let  $x = x[0] \dots x[n-1]$  be a *string* of length  $n = |x|$  over  $\Sigma$ . By  $\Sigma^k$  we denote the set of all strings of length  $k > 0$ . For two indices  $i$  and  $j \geq i$  of  $x$ ,  $x[i..j]$  is the *fragment* of  $x$  starting at position  $i$  and ending at position  $j$ . The fragment  $x[i..j]$  is an *occurrence* of the underlying *substring*  $p = x[i] \dots x[j]$ ; we say that  $p$  occurs (or starts) at *position*  $i$  in  $x$ . A *prefix* of  $x$  is a substring of the form  $x[0..j]$  and a *suffix* of  $x$  is a substring of the form  $x[i..n-1]$ . By  $xy$  or  $x \cdot y$  we denote the *concatenation* of strings  $x$  and  $y$ :  $xy = x[0] \dots x[|x| - 1]y[0] \dots y[|y| - 1]$ . Given strings  $x$  and  $y$ , a *suffix/prefix overlap* of  $x$  and  $y$  is a suffix of  $x$  that is a prefix of  $y$ .

Let  $S$  be a collection of strings. The *order- $k$  de Bruijn graph* (DBG) of  $S$  is a directed multigraph, denoted by  $G_{S,k}(V, E)$ , such that  $V$  is the set of length- $(k-1)$  substrings of the strings in  $S$  and  $G_{S,k}$  contains an edge  $(u, v)$  with multiplicity  $m_{u,v}$  if and only if the string  $u[0] \cdot v$  is equal to the string  $u \cdot v[k-2]$  and this string occurs exactly  $m_{u,v}$  times in total in the strings in  $S$ . For instance, suppose that  $S$  is generated from a genome sample by a sequencing experiment: then any Eulerian circuit<sup>1</sup> of  $G_{S,k}(V, E)$  corresponds to a single genome reconstruction [26, 23]. In this model, due to sequencing errors,  $G_{S,k}$  would never be Eulerian in practice [24]; and it would not even be weakly connected. One could, however, try to make  $G_{S,k}$  Eulerian by duplicating some of its *existing* edges [22] or introducing *new ones* when the former do not suffice to make  $G_{S,k}$  Eulerian [5]. A natural optimization goal in either case would be to minimize the total cost of edge additions.

In this paper, we study the problem of *making any arbitrary  $G_{S,k}$  weakly connected* by introducing a set of new edges of minimal total cost (as well as the underlying set of new nodes when they do not exist in  $G_{S,k}$ ). Finding a cheapest way for making  $G_{S,k}$  weakly connected is important because one can subsequently apply the linear-time algorithm of Bernardini et al. [5] to balance it by adding a set of edges of minimal total cost, and thus making the graph Eulerian. Since making the DBG *directly* Eulerian by adding a set of new edges of minimal total cost is NP-hard (from the *shortest common superstring* problem [11]), the *connect-and-balance* approach, generally, serves as a good-performing heuristic [5]. Our work falls into a broader line of research that is concerned with algorithmic problems on strings that can be formulated as problems on DBGs [7, 6, 8, 30, 28, 29, 31, 3, 4].

By  $G_{\Sigma,k}(V_{\Sigma,k}, E_{\Sigma,k})$ , we denote the *complete* DBG of order  $k$  over alphabet  $\Sigma$  with  $|V_{\Sigma,k}| = \sigma^{k-1}$  and  $|E_{\Sigma,k}| = \sigma^k$ . Any two nodes  $u$  and  $v \neq u$  in  $V_{\Sigma,k}$  can be connected by a *super-edge* whose weight  $w_{u,v}$  is in  $[1, k)$ : this is the shortest path of  $w_{u,v}$  unit-cost edges in  $G_{\Sigma,k}$ . For example, for edge  $(abc, bcac)$  with  $abc, bcac \in V_{\Sigma,k}$  and  $k = 5$ , we have  $w_{abc, bcac} = 2$  corresponding to the following two unit-cost edges:  $abc \rightarrow abca \rightarrow bcac$ .

We next formally define the main problem in scope; see Figure 1 for an example.

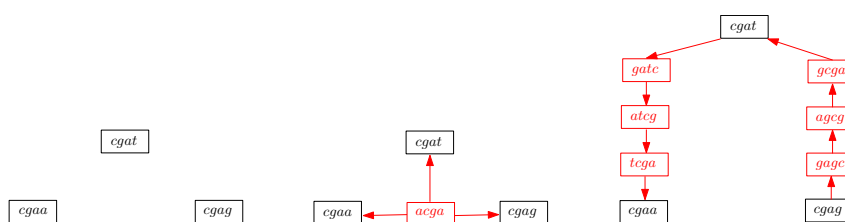
### CONNECTING DE BRUIJN GRAPHS WITH EDGES (CONNECT-DBG-E)

**Input:** A de Bruijn graph  $G(V, E)$  of order  $k$  over alphabet  $\Sigma = [0, \sigma)$ ,  $\sigma \leq (k-1)|V|$ .

**Output:** A set  $A \subseteq E_{\Sigma,k}$  of edges and a set  $B \subseteq V_{\Sigma,k}$  of nodes such that  $G(V \cup B, E \cup A)$  is weakly connected and  $A$  is of minimum size.

Let us remark that CONNECT-DBG-E allows for connecting two components  $C_i, C_j$  of  $G$  by a path directed from  $C_i$  to  $C_j$  but this needs not be the case in general; see Figure 1.

<sup>1</sup> An Eulerian circuit is a graph cycle using each graph edge exactly once. Such a graph is called Eulerian.



■ **Figure 1** An input DBG of order  $k = 5$  with  $d = 3$  weakly connected components (left); a solution to CONNECT-DBG-E with cost 3 (middle); a solution to CONNECT-DBG-P with cost 8 (right). The CONNECT-DBG-P problem is a restricted version of the CONNECT-DBG-E problem allowing to connect the graph *only by means of directed paths* whose endpoints are two components. In fact, the graph on the right also shows an optimal solution to making the graph on the left semi-Eulerian.

We fix throughout a DBG  $G(V, E)$  of order  $k$  over the integer alphabet  $\Sigma = [0, \sigma]$ ,  $\sigma \leq (k - 1)|V|$ , consisting of  $d$  weakly connected components. We show the following results:

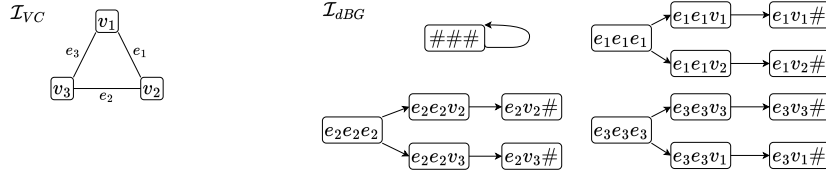
1. CONNECT-DBG-E is NP-hard. We show this via a somewhat surprising and highly non-trivial reduction from the MINIMUM VERTEX COVER problem [15]. See Section 2.
2. No *polynomial-time approximation scheme* (PTAS) exists for CONNECT-DBG-E unless the *unique games conjecture* [16] fails. We complement this result with a polynomial-time  $(2 - 2/d)$ -approximation algorithm for CONNECT-DBG-E. Our strategy relies on an existing  $(2 - 2/d)$ -approximation algorithm for the MINIMUM STEINER TREE problem [18], where  $d$  is the number of terminals of the *Steiner tree*.<sup>2</sup> See Section 3.
3. Making  $G$  weakly connected by adding a set of  $d - 1$  directed paths (between components) of *minimal total cost* can be done in  $\mathcal{O}(k|V|\alpha(|V|) + |E|)$  time, where  $\alpha(\cdot)$  is the inverse Ackermann function. We call this the CONNECT-DBG-P problem; see Figure 1 for an example. Our algorithm improves the  $\mathcal{O}(k|V|\log(|V|) + |E|)$ -time algorithm by Bernardini et al. [5]. We make use of an augmented static version of the Aho-Corasick machine [1] to select the shortest possible paths, while keeping track of the connected components as they are dynamically merged by using a *union-find* data structure [10]. See Section 4.
4. An *integer linear program* (ILP) formulation of polynomial size for making  $G$  Eulerian with *minimal total cost*. This is a flow-based formulation inspired by the above relaxation of connecting the  $d$  components with  $d - 1$  paths (CONNECT-DBG-P). Since the graph must also be balanced (the in- and out-degree for every node is the same), *an optimal solution can always be decomposed into such paths*. We complement our ILP with proof-of-concept experiments on real data showing that problem instances of around 900 nodes and edges can be solved using an off-the-shelf ILP solver within 10 hours. See Section 5.

## 2 Hardness of Connect-DBG-E

In this section, we prove that CONNECT-DBG-E is NP-hard via a reduction from MINIMUM VERTEX COVER [15]. Recall that MINIMUM VERTEX COVER asks, given an undirected graph  $G(V, E)$ , to find a smallest subset  $C$  of  $V$  such that every edge in  $E$  has at least one endpoint in  $C$ . In this section, we use the term *vertex* instead of *node* for obvious reasons.

► **Theorem 1.** *CONNECT-DBG-E is NP-hard.*

<sup>2</sup> The Steiner tree of some subset of the nodes of a graph  $G$  is a minimum-weight connected subgraph of  $G$  that includes all the nodes.



■ **Figure 2** An instance of MINIMUM VERTEX COVER (left) and the instance of CONNECT-DBG-E (right) implied by the reduction of Theorem 1.

**Proof.** Let  $\mathcal{I}_{VC} = G(V, E)$  be an instance of MINIMUM VERTEX COVER. We reduce it to an instance  $\mathcal{I}_{DBG}$  of CONNECT-DBG-E, consisting of a  $\text{DBG } \tilde{G}$  of order 4 over an alphabet  $\Sigma$  of size  $\sigma = |V| + |E| + 1$ .  $\tilde{G}$  consists of  $|E|$  edge gadgets, plus a vertex  $v_{\#} = \#\#\#$  and an edge  $(v_{\#}, v_{\#})$ . The edge gadget for  $e_i = (u, v) \in E$  has the following vertices and edges:  $V_i = \{v_{1i}, v_{2i}, v_{3i}, v_{4i}, v_{5i}\}$ , with  $v_{1i} = e_i e_i e_i$ ;  $v_{2i} = e_i e_i u$ ;  $v_{3i} = e_i e_i v$ ;  $v_{4i} = e_i u \#$ ;  $v_{5i} = e_i v \#$ ; and  $E_i = \{(v_{1i}, v_{2i}), (v_{1i}, v_{3i}), (v_{2i}, v_{4i}), (v_{3i}, v_{5i})\}$ . We call  $v_{4i}$  and  $v_{5i}$  the *terminal* vertices of the  $i$ th component; the remaining vertices are called *non-terminal*. The reduction requires polynomial time: an example is illustrated in Figure 2. Let  $OPT(\mathcal{I}_{VC})$  and  $OPT(\mathcal{I}_{DBG})$  denote the size of an optimal solution to  $\mathcal{I}_{VC}$  and  $\mathcal{I}_{DBG}$ , respectively.

▷ **Claim 2.** A solution to  $\mathcal{I}_{VC}$  of size  $\alpha$  implies a solution to  $\mathcal{I}_{DBG}$  of size  $\alpha + |E|$ .

*Proof.* Let  $C$  be a cover of  $G$  of size  $\alpha$ . For each  $v \in C$ , we add to  $\tilde{G}$  a new vertex  $v\#\#$ ; we then connect all the new vertices to  $\#\#\#$  using  $\alpha$  edges in total. Since  $C$  is a vertex cover for  $G$ , by construction, one of the two terminal vertices of each edge gadget in  $\tilde{G}$  corresponds to a vertex in  $C$  and it can thus be connected with a single edge to one of the newly added vertices, using another  $|E|$  edges in total. We can thus make  $\tilde{G}$  weakly connected by adding  $\alpha$  vertices and  $\alpha + |E|$  edges. ◁

▷ **Claim 3.** A solution to  $\mathcal{I}_{DBG}$  of size  $\beta + |E|$  implies a solution to  $\mathcal{I}_{VC}$  of size at most  $\beta$ .

*Proof.* We observe that any solution to  $\mathcal{I}_{DBG}$  must add new vertices, as by construction, no two gadgets can be connected with a single edge, nor can they be connected to  $\#\#\#$  with a single edge. Moreover, any solution that adds  $\gamma$  new vertices must add at least  $\gamma + |E|$  new edges, as this is the minimum possible number to connect  $|E| + \gamma + 1$  components (the  $|E|$  edge gadgets, the vertex  $\#\#\#$ , and the  $\gamma$  new vertices).

We further observe that the only way two distinct edge gadgets can be connected using two edges is by adding an edge from one of the terminal vertices of each gadget to a newly added vertex of the form  $v\#\lambda$ , where  $\lambda$  is any letter from the alphabet and  $v$  is a letter corresponding to a vertex of  $\mathcal{I}_{VC}$  that is an endpoint of both the edges corresponding to the two gadgets. This is because any two vertices of two distinct gadgets have no suffix/prefix overlap, thus no path of length two can connect them; and any two vertices of two distinct gadgets have no common prefix, thus there cannot be two edges out of a new vertex that reach two distinct gadgets. On the other hand, the terminal vertices of two distinct gadgets can have the same suffix  $v\#$  for some  $v$  and thus can be both connected to a vertex of the form  $v\#\lambda$  – note that when  $\lambda = \#$  these vertices can be, in turn, connected to  $\#\#\#$ .

Now consider a solution to  $\mathcal{I}_{DBG}$  that adds  $\beta$  new vertices. We construct a cover for  $\mathcal{I}_{VC}$  as follows. For every newly added vertex  $u\#\lambda$  that is adjacent to more than one gadget, add the corresponding vertex  $u$  to the cover: this covers all the edges corresponding to the adjacent gadgets. For the edge gadgets that are connected to some new vertex which is not adjacent to any other gadget, add one of the endpoints of the corresponding edge of  $E$  to the vertex cover: this covers the remaining edges of  $\mathcal{I}_{VC}$ . The cover is thus of size at most  $\beta$ . ◁

Let us now prove that  $OPT(\mathcal{I}_{VC}) = \ell \iff OPT(\mathcal{I}_{DBG}) = |E| + \ell$ .

$\Rightarrow$ ) By Claim 2, an optimal solution to  $\mathcal{I}_{VC}$  of size  $\ell$  implies a solution to  $\mathcal{I}_{DBG}$  of size  $\ell + |E|$ . Suppose for a contradiction that this solution is not optimal, i.e., there exists another solution to  $\mathcal{I}_{DBG}$  of size  $\ell' + |E|$  with  $\ell' < \ell$  new vertices. By Claim 3, this would imply a cover for  $\mathcal{I}_{VC}$  of size at most  $\ell' < \ell$ , a contradiction.

$\Leftarrow$ ) By Claim 3, an optimal solution to  $\mathcal{I}_{DBG}$  of size  $|E| + \ell$  implies a solution to  $\mathcal{I}_{VC}$  of size at most  $\ell$ . Suppose for a contradiction that  $OPT(\mathcal{I}_{VC}) = \ell' < \ell$ : by Claim 2, this would imply a solution to  $\mathcal{I}_{DBG}$  of size  $\ell' + |E|$ , a contradiction.  $\blacktriangleleft$

The above reduction is not approximation preserving (because  $OPT(\mathcal{I}_{VC}) = \ell \iff OPT(\mathcal{I}_{DBG}) = |E| + \ell$ ), which would have allowed us to directly obtain a constant-factor approximation algorithm for CONNECT-DBG-E from MINIMUM VERTEX COVER [14], and to prove its inapproximability from the inapproximability of MINIMUM VERTEX COVER [25].

### 3 Approximating Connect-DBG-E

We start by proving that the existence of a PTAS for CONNECT-DBG-E is excluded under the *unique games conjecture* [16]. To achieve this, we restrict to a specific class of graphs.

**► Theorem 4.** *There exists no PTAS for CONNECT-DBG-E unless the unique games conjecture fails.*

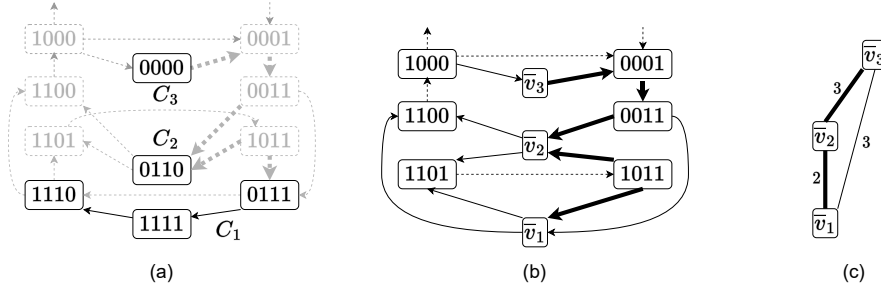
**Proof.** Consider the same reduction as in the proof of Theorem 1. The sizes of the solutions to the two problem instances  $\mathcal{I}_{VC}$  and  $\mathcal{I}_{DBG}$  always differ by a term of exactly  $|E|$ , implying that the reduction preserves the inapproximability of CONNECT-DBG-E in the case where the size of the minimum vertex cover is  $\Omega(|E|)$ . Indeed, suppose for a contradiction that there exists a PTAS for CONNECT-DBG-E. Then given any instance  $\mathcal{I}_{VC}$ , we could obtain a solution of size  $d$  by reducing it to  $\mathcal{I}_{DBG}$ , running the PTAS, and subtracting  $|E|$  from the result. Let  $d_{OPT} + |E|$  denote the size of an optimal solution to  $\mathcal{I}_{DBG}$  (thus  $d_{OPT}$  is the size of an optimal solution to  $\mathcal{I}_{VC}$ ), and  $d + |E|$  the solution returned by the PTAS. We have that  $d + |E| \leq (1 + \epsilon)(d_{OPT} + |E|)$ , for some input parameter  $\epsilon > 0$ , and thus

$$d \leq (1 + \epsilon)(d_{OPT} + |E|) - |E| = (1 + \epsilon)d_{OPT} + \epsilon|E|. \quad (1)$$

When  $d_{OPT} = \Omega(|E|)$ , let  $c > 1$  be a constant such that  $\frac{|E|}{c} \leq d_{OPT} \leq |E|$  (as the size of any vertex cover is bounded by  $|E|$ ). From Equation 1, we obtain that  $d \leq (1 + (1 + c)\epsilon)d_{OPT}$ , which contradicts the inapproximability of MINIMUM VERTEX COVER. An example of graphs for which the size of the minimum vertex cover is  $\Omega(|E|)$  are bounded-degree graphs: indeed, they have at most  $|V| \cdot \Delta/2$  edges and a minimum vertex cover of size at least  $|V|/(\Delta + 1) = \Omega(|E|)$ , where  $\Delta$  is the bounded maximum degree. MINIMUM VERTEX COVER is hard to approximate (unless the unique games conjecture fails) on bounded degree graphs to within a factor  $2 - (2 + o_\Delta(1)) \frac{\log \log \Delta}{\log \Delta}$  for a sufficiently large integer  $\Delta$  [2].

This implies that there is no PTAS for CONNECT-DBG-E (conditioned on the unique games conjecture) when restricted to the very specific instances obtained, via the reduction of Theorem 1, from instances of bounded-degree MINIMUM VERTEX COVER. We can thus conclude that, in general, there exists no PTAS for CONNECT-DBG-E conditioned on the unique games conjecture.  $\blacktriangleleft$

Motivated by Theorem 4, we next present a  $(2 - 2/d)$ -approximation algorithm for CONNECT-DBG-E. Our strategy relies on an existing  $(2 - 2/d)$ -approximation algorithm for the MINIMUM STEINER TREE problem, where  $d$  is the number of terminals. Recall that MINIMUM STEINER TREE asks, given a graph  $G'(V', E')$  with non-negative edge weights and a subset of *terminal* nodes, to compute a tree of minimum weight that contains all terminals.



■ **Figure 3** Construction of Theorem 6. The input  $G$  consists of 3 weakly connected components shown in (a) with black solid lines; grey dashed lines represent nodes and edges of the underlying complete DBG (only the portion directly connected to nodes of  $G$  is depicted); grey thick dashed edges form a solution to CONNECT-DBG-E, which in this case would be returned by the approximation algorithm.  $G'$  is shown in (b): solid edges are in  $\bar{E}$ , thick edges represent the same solution as in (a). The metric closure of  $G'$  is shown in (c): thick edges represent the same solution as in (a).

Given a DBG  $G(V, E)$  of order  $k$  consisting of  $d$  weakly connected components  $C_1, \dots, C_d$ , let  $G'(V', E')$  be the graph obtained from the complete DBG  $G_{\Sigma, k}$  collapsing each component  $C_i$  into one super-node  $\bar{v}_i$ : an example is in Figure 3. More formally,  $V' = (V_{\Sigma, k} \setminus V) \cup \bar{V}$ , where  $\bar{V} = \{\bar{v}_1, \dots, \bar{v}_d\}$  is a set of unlabeled nodes s.t.  $\bar{v}_i \notin V_{\Sigma, k}$  corresponds to  $C_i$  for all  $i \in [1, d]$ ; and  $E' = (E_{\Sigma, k} \cap ((V_{\Sigma, k} \setminus V) \times (V_{\Sigma, k} \setminus V))) \cup \bar{E}$ , where  $E_{\Sigma, k} \cap ((V_{\Sigma, k} \setminus V) \times (V_{\Sigma, k} \setminus V))$  are simply the edges of the complete DBG connecting pairs of nodes that are both not in  $G$ ; the edges in  $\bar{E}$  are s.t. there is an edge from a super-node  $\bar{v}_i$  to a node  $v \in (V_{\Sigma, k} \setminus V)$  if and only if at least one of the nodes of  $C_i$  would be connected to  $v$  by an edge in the complete DBG; and likewise for edges  $(v, \bar{v}_i)$ . Two super-nodes are connected by an edge if and only if two nodes in the respective components would be connected by an edge in the complete DBG. Formally,  $\bar{E} = \bar{E}_1 \cup \bar{E}_2 \cup \bar{E}_3$ , where

$$\begin{aligned}
 \bar{E}_1 &= \{(\bar{v}_i, v) \mid \exists u \in C_i \text{ and } v \in (V_{\Sigma, k} \setminus V) \text{ s.t. } (u, v) \in E_{\Sigma, k}\}, \\
 \bar{E}_2 &= \{(v, \bar{v}_i) \mid \exists u \in C_i \text{ and } v \in (V_{\Sigma, k} \setminus V) \text{ s.t. } (v, u) \in E_{\Sigma, k}\}, \\
 \bar{E}_3 &= \{(\bar{v}_i, \bar{v}_j) \mid \exists u \in C_i \text{ and } v \in C_j \text{ s.t. } (v, u) \in E_{\Sigma, k}\}.
 \end{aligned}$$

Inspect Figure 3(b): the edge  $(1011, \bar{v}_1)$  belongs to set  $\bar{E}_2$ ; the edge  $(\bar{v}_3, 0001)$  belongs to  $\bar{E}_1$ ; no edges belong to  $\bar{E}_3$  in this example.

It is easy to see that solving CONNECT-DBG-E for  $G$  is equivalent to solving an instance of MINIMUM STEINER TREE on  $G'$  with  $\bar{v}_1, \dots, \bar{v}_d$  as terminals. Any polynomial-time approximation algorithm for MINIMUM STEINER TREE can therefore be applied to solve CONNECT-DBG-E with the same approximation ratio. Unfortunately, when applied naively, this strategy does not give a polynomial-time algorithm for CONNECT-DBG-E, because  $G'$  has an exponential size and thus constructing it requires, in general, exponential time.

To overcome this issue, we focus on a specific approximation algorithm for the MINIMUM STEINER TREE problem which does not require computing the whole graph  $G'$  but rather only its *metric closure*, defined as a weighted complete graph on the set of terminals  $\bar{v}_1, \dots, \bar{v}_d$  such that the weight on edge  $(\bar{v}_i, \bar{v}_j)$  is the length of the shortest *undirected* path between  $\bar{v}_i$  and  $\bar{v}_j$  in  $G'$ . An example of the metric closure of  $G'$  is in Figure 3(c). Note, in particular, that the length of the shortest undirected path between two nodes (i.e., a sequence of edges that form a path if their direction is ignored) is smaller or equal to the length of the shortest *directed* path: for instance, the shortest undirected path between 0110 and 0111 in Figure 3 is of length 2 (through node 1011), while the shortest directed path is of length 3 (through nodes 1101 and 1011). In contrast to explicitly constructing the whole  $G'$ , computing only its metric closure can be done in polynomial time, as stated by the following lemma.

► **Lemma 5.** *For any dBG  $G(V, E)$  of order  $k$ , computing the metric closure of  $G'$  can be done in  $\mathcal{O}(k|V|^2)$  time.*

**Proof.** Let  $G$  consist of the weakly connected components  $C_1, \dots, C_d$ . By the definition of  $G'$ , computing its metric closure requires computing the length of the shortest undirected path in  $G_{\Sigma, k}$  between any pair of nodes that lie in two different components of  $G$ . An algorithm to compute shortest undirected paths in dBGs in  $\mathcal{O}(k)$  time per pair has been proposed in [20]: this algorithm only relies on computing common substrings for each pair of nodes and it does not require to construct  $G'$ .

The weight of an edge  $(\bar{v}_i, \bar{v}_j)$  in the metric closure of  $G'$  is thus obtained by computing the length of the shortest undirected path between every pair of nodes  $u \in C_i, v \in C_j$  and taking the minimum over such values. Over all edges  $(\bar{v}_i, \bar{v}_j)$ , this requires time  $\mathcal{O}(k \sum_{i,j \in [1,d]} |C_i||C_j|) = \mathcal{O}(k|V|^2)$ . ◀

► **Theorem 6.** *For any dBG  $G(V, E)$  of order  $k$  consisting of  $d$  weakly connected components, there exists an  $\mathcal{O}(k|V|^2)$ -time  $(2 - 2/d)$ -approximation algorithm for CONNECT-DBG-E.*

**Proof.** The algorithm, which is an adaptation of [18] to dBGs, consists of three steps:

- (i) Construct the metric closure of  $G'$ .
- (ii) Compute a minimum spanning tree of the metric closure.
- (iii) Convert the minimum spanning tree into a set of nodes and a set of edges to be added to  $G$  to make it weakly connected.

The correctness follows directly from the fact that a minimum spanning tree for the metric closure of  $G'$  is a  $(2 - 2/d)$ -approximation for the minimum Steiner tree [18], where  $d$  is the number of terminals and thus the number of weakly connected components of  $G$ .

Step (i) requires  $\mathcal{O}(k|V|^2)$  time as per Lemma 5. Step (ii) can be done in  $\mathcal{O}(d^2)$  time by applying, e.g., Prim's algorithm [27]. Finally, Step (iii) can be done by applying again the algorithm from [20] to compute the shortest undirected path between every pair  $\bar{v}_i, \bar{v}_j$  such that the edge  $(\bar{v}_i, \bar{v}_j)$  is in the minimum spanning tree of the metric closure of  $G'$  and taking the union of the nodes and edges in such paths. This requires  $\mathcal{O}(k|V|^2)$  total time. ◀

## 4 Connecting de Bruijn Graphs with Paths in Essentially Optimal Time

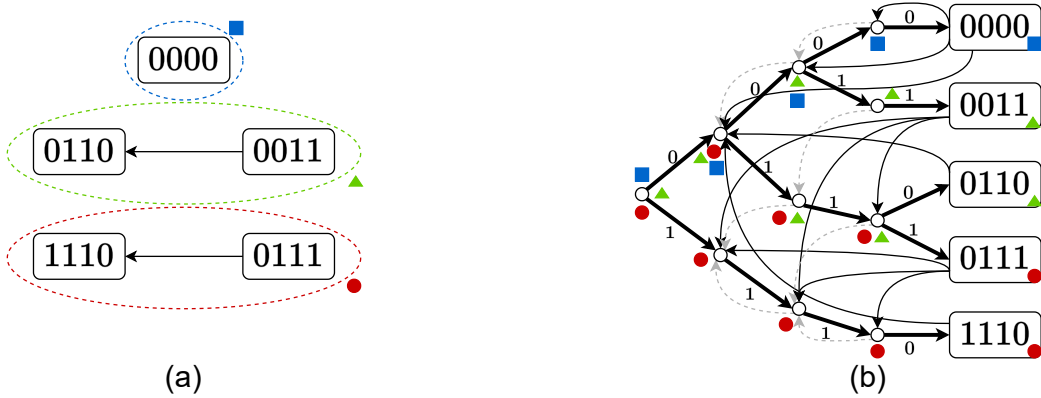
In this section, we present an exact algorithm for a restricted version of CONNECT-DBG-E, in which we are asked to make a dBG  $G(V, E)$  of order  $k$  weakly connected by adding a set of *directed paths* (between components) of minimum total length. This problem was already considered and solved in (nearly optimal) polynomial time in [5]; here we propose a much simpler and essentially time-optimal algorithm. To formally define the restricted problem we consider, we first need the following definition of a *condensed graph* of a dBG from [5].

► **Definition 7** (Condensed Graph). *Given a dBG  $G(V, E)$  of order  $k$  over an alphabet  $\Sigma$  with a set  $\mathcal{C}$  of weakly connected components, its condensed graph  $\widehat{G}(\widehat{V}, \widehat{E})$  is a weighted directed multigraph whose nodes  $\widehat{V}$  are in a bijection with  $\mathcal{C}$ . The edges have integer weights in  $[1, k]$ : there is an edge  $(i, j) \in \widehat{E}$  for each pair of nodes  $u_i \in C_i, u_j \in C_j$ , with  $C_i, C_j \in \mathcal{C}$ , and its weight is the length of the shortest path from  $u_i$  to  $u_j$  in the complete dBG  $G_{\Sigma, k}$ .*

CONNECTING DE BRUIJN GRAPHS WITH PATHS (CONNECT-DBG-P)

**Input:** A de Bruijn graph  $G(V, E)$  of order  $k$  over alphabet  $\Sigma = [0, \sigma], \sigma \leq (k - 1)|V|$ .

**Output:** A minimum-weight spanning tree  $\mathcal{T}$  of the condensed graph  $\widehat{G}$  of  $G$ .



■ **Figure 4** (a) An instance of CONNECT-DBG-P consisting of 3 components. (b) The modified AC machine built in the preprocessing phase of Algorithm 1. Dashed arrows are the backward edges of the original AC machine; solid, curved arrows are the backward edges of the modified AC machine (backward edges to the root are omitted). Symbols close to the states represent their lists of colors.

A solution  $\mathcal{T}$  to CONNECT-DBG-P naturally corresponds to a set  $\mathcal{P}$  of paths on  $G_{\Sigma,k}$  that make  $G$  weakly connected: an edge  $(i, j)$  of  $\mathcal{T}$  corresponds to the shortest path from some node  $u_i \in C_i$  to some  $u_j \in C_j$ , and in turn, by the definition of DBG, such a path is determined by the longest suffix/prefix overlap of  $u_i$  and  $u_j$ .

The algorithm for CONNECT-DBG-P proposed in [5] makes use of a dynamic version of the Aho-Chorasick (AC) machine of the nodes of  $G$  to find the shortest connecting paths and to keep track of the connected components as they are progressively united by these paths. Here we will use an augmented but static version of the same AC machine to select the paths, and we will keep track of the connected components as they are dynamically merged by employing a union-find data structure.

Before describing our solution, let us recall that AC machines generalize the Knuth-Morris-Pratt [17] algorithm for a set of strings. Informally, AC machines are finite-state machines that resemble a trie with additional *backward edges* (also called *failure transitions*) between the states. There is exactly one failure transition  $f(u) = v$  from each state  $u$  (except for the root state) to some state  $v$ . Backward edges encode suffix/prefix overlaps between the strings represented by the AC machine, as specified by the following lemma.

► **Lemma 8** (Aho-Corasick lemma [1]). *Let  $u$  and  $v$  be two strings representing two distinct states of the AC machine, and identify the states with such strings. Then,  $f(u) = v$  if and only if  $v$  is the longest proper suffix of  $u$  that is also a prefix of some string in the machine.*

In a preprocessing step, we compute the  $d$  weakly connected components of  $G$ , choose a representative node for each component, and assign it a unique color: we will identify each color with the connected component and with the representative node it is associated with. To store the weakly connected components of  $G$ , we construct a union-find data structure [10]. Union-find data structures allow to efficiently perform any sequence of operations of the following two kinds on disjoint sets:  $\text{union}(A, B)$  merges sets  $A$  and  $B$ ; and  $\text{find}(x)$  returns the representative element of the unique set containing  $x$ .

We then construct the AC machine of the nodes of  $G$  and preprocess it as follows; see Figure 4 for an example. We color each terminal state with the color of the connected component of the node of  $G$  it represents. Each internal state is assigned the union of the colors of its descendants. From each terminal state  $s$ , we follow the unique path of backward edges to the root and, for each state  $u$  on this path, we add to the machine a backward edge  $(s, u)$ . We finally prune all the backward edges connecting two non-terminal states.



Once we are done with the preprocessing phase, we start performing a reverse BFS of the machine (which begins from the deepest internal states and proceeds level-by-level towards the root) and check whether the overlap encoded by the backward edges incoming to each of the visited states can be used to unite some components. This is because the deeper the state  $u$  reached by a backward edge  $(s, u)$ , the longer the overlap encoded by the edge; and the longer the overlap, the shorter the path connecting  $s$  with all the nodes represented by the terminal states below  $u$ . The idea is to greedily select the backward edges encoding paths that connect two currently separate components, using the union-find data structure both to check which components are still separate and to unite them when we select a shortest path.

**Implementation Details.** We associate two lists to each state  $u$ : one for the colors; and one for the incoming backward edges. The colors of  $u$  are stored in a list  $LC_u$  of ordered pairs  $\langle c, p_c \rangle$ , where  $c$  is a distinct color and  $p_c$  is a pointer to any terminal state of color  $c$  below  $u$ . The backward edges incoming to  $u$  are stored using a list  $LE_u$  of their tails (recall that all the tails are terminal states). We will need to keep track of the states of the AC machine visited during the execution of the algorithm, therefore we set up a visited/unvisited flag for each internal state, initially set to “unvisited”.

When we visit a state  $u$  for the first time, we select the first backward edge  $(s, u)$  of the list  $LE_u$  (if any). Let  $c$  be the color of the terminal state  $s$ . For each color  $\alpha$  in the list  $LC_u$ , we compare the representative of the current connected component of the node associated with color  $c$  and the representative of the current connected component of the node associated with  $\alpha$ , that is, we compare the results of operations  $\text{find}(c)$  and  $\text{find}(\alpha)$ . If they differ, it means that the components of the two nodes are still separate, thus we can unite them by adding the path linking  $s$  to the node pointed by  $p_\alpha$ , and keep track of the fact that they now constitute a single connected component by performing  $\text{union}(c, \alpha)$  (recall that we identify colors and connected components). If  $\text{find}(c) = \text{find}(\alpha)$ , then the two components were united in a previous step, thus we just move on to the next color in  $LC_u$ . At the end of the scan of  $LC_u$ ,  $c$  and all the colors of  $u$  will represent the same connected component.

We then select each subsequent backward edge in  $LE_u$ , and we compare just the color of its tail and the first color in  $LC_u$ , again by performing two  $\text{find}$  operations. We merge the two components and add the appropriate path if they differ, or move on to the next backward edge in  $LE_u$  (or to the next state, if  $LE_u$  is exhausted) if they are the same.

The whole procedure is summarized in Algorithm 1.

► **Theorem 9.** *Algorithm 1 solves CONNECT-DBG-P in  $\mathcal{O}(k|V|\alpha(|V|) + |E|)$  time.*

**Proof.**

**Correctness.** Algorithm 1 is essentially Kruskal’s algorithm [19] applied to the condensed graph  $\hat{G}$ . Indeed, the longest suffix/prefix overlap between any two nodes  $u_1, u_2$  (which determines the weight of the corresponding edge in  $\hat{G}$ ) is encoded in the AC machine by a path of backward edges starting from  $u_1$  and ending at an ancestor of  $u_2$  [33, Theorem 4]. Thus, by construction, the backward edges we add to the AC machine encode the edges of  $\hat{G}$  (a single backward edge may correspond to multiple edges of  $\hat{G}$ ), plus some shorter suffix/prefix overlaps that are discarded: they have no correspondence in  $\hat{G}$ . In particular, a backward edge  $(s, u)$  of the modified AC machine encodes overlaps of length  $d(u)$ , where  $d(u)$  is the depth of state  $u$ , i.e., the length of the string it represents [33, Lemma 3].

Algorithm 1 always returns a feasible solution. Indeed, every time a state  $u$  is visited, all the components (i.e., nodes from  $\hat{V}$ ) descending from  $u$  are connected (Lines 9-11); since all the components descend from the root, at the end the whole  $\hat{G}$  is connected; and the union

## 6:10 Connecting de Bruijn Graphs

### Algorithm 1 CONNECT DBG WITH PATHS.

---

```

1: Compute the  $d$  weakly connected components of  $G(V, E)$  and the union-find data structure over
   the components; identify each component with a distinct color and each color with a node of
   that component. Construct and preprocess the AC machine of  $V$ .
2:  $\mathcal{P} \leftarrow \emptyset$ ; comp-count  $\leftarrow d$ ;
3: while comp-count  $> 1$  do
4:    $u \leftarrow$  next state of the AC machine in a reverse BFS order;
5:   for all  $(s, u)$  in  $LE_u$  do
6:      $c \leftarrow \text{color}(s)$ ;
7:     if  $u$  is unvisited then
8:       Flag  $u$  as visited;
9:       for all  $\langle \alpha, p_\alpha \rangle$  in  $LC_u$  do
10:        if  $\text{find}(c) \neq \text{find}(\alpha)$  then
11:           $\text{union}(c, \alpha)$ ; comp-count  $\leftarrow \text{comp-count} - 1$ ;  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\text{path from } s \text{ to } p_\alpha\}$ ;
12:        else
13:           $\langle \alpha, p_\alpha \rangle \leftarrow$  first element of  $LC_u$ ;
14:          if  $\text{find}(c) \neq \text{find}(\alpha)$  then
15:             $\text{union}(c, \alpha)$ ; comp-count  $\leftarrow \text{comp-count} - 1$ ;  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\text{path from } s \text{ to } p_\alpha\}$ ;
16: return  $\mathcal{P}$ 

```

---

and `find` operations ensure that no loop is created. Moreover, the algorithm only returns paths corresponding to backward edges that encode maximal suffix/prefix overlaps, thus edges of  $\hat{G}$ . Suppose for a contradiction that the algorithm uses an edge  $(s, u)$  to connect  $s$  with a descendant  $s'$  of  $u$  using an overlap of length  $d(u)$ , while the longest overlap between  $s$  and  $s'$  is of length  $\ell > d(u)$ . Then, by construction, there is a lower ancestor  $v$  of  $s'$  with  $d(v) = \ell$  and another backward edge  $(s, v)$ . Since the states are visited in a reverse BST order,  $v$  is visited before  $u$ ,  $(s, v)$  is considered before  $(s, u)$  and it is used to connect  $s$  to  $s'$ , thus uniting their components; when  $u$  is visited afterwards and  $(s, u)$  is considered,  $s$  and  $s'$  are already in the same component, so the shorter overlap is discarded, a contradiction.

Finally, optimality follows directly from the correctness of Kruskal's algorithm [19].

**Complexity.** Computing the connected components of  $G$  and assigning each a color  $c \in [1, d]$  requires  $\mathcal{O}(|V| + |E|)$  time, where  $|E|$  is the number of *distinct* edges of  $G$  [13]. Building the AC machine of  $V$  takes  $\mathcal{O}(k|V|)$  time because each string is of length  $k - 1$  [1, 9]. Initializing a union-find data structure for the weakly connected components of  $G$  requires  $\mathcal{O}(|V|)$  time [10].

During the execution of the algorithm, we perform exactly  $d - 1 < |V|$  union operations; moreover, at each visited state  $u$ , we perform a number of `find` operations proportional to the sum of the number of colors of  $u$  and the number of backward edges incoming to  $u$ . The total size of lists  $LE_u$  and  $LC_u$  over all non-terminal states  $u$  is bounded by  $\mathcal{O}(k|V|)$ , because the color of each of the  $|V|$  terminal states propagates to at most  $k - 2$  non-terminal states (the depth of the AC machine is  $k - 1$ ), and by construction, there are up to  $k - 2$  backward edges from each terminal state.

Since the cost of each `find` and `union` operation amortizes to  $\mathcal{O}(\alpha(|V|))$  [10], the total cost of this procedure is  $\mathcal{O}(\alpha(|V|)(k|V| + d - 1)) = \mathcal{O}(k|V|\alpha(|V|))$ . Since the preprocessing phase requires  $\mathcal{O}(k|V| + |E|)$  time, the statement follows.  $\blacktriangleleft$

## 5 Making a dBG Eulerian through ILP

Let us recall some basic definitions. An *Eulerian trail* is a trail in a finite graph that visits every edge exactly once allowing for revisiting nodes. An *Eulerian circuit* is an Eulerian trail that starts and ends on the same node. A graph with an Eulerian circuit is called *Eulerian*. A graph with an Eulerian trail but with no Eulerian circuit is called *semi-Eulerian*.<sup>3</sup>

Recall that, by  $G_{\Sigma,k}(V_{\Sigma,k}, E_{\Sigma,k})$ , we denote the complete dBG of order  $k$  over alphabet  $\Sigma$ . Here, we present an ILP formulation for making any arbitrary dBG  $G(V, E)$  Eulerian (or semi-Eulerian) by adding a multiset of edges from  $E_{\Sigma,k}$ ; this problem is NP-hard via a simple reduction from the shortest common superstring problem [11]. Instead of explicitly adding nodes, we assume that any two nodes can be connected with a (super-)edge whose cost is in  $[1, k)$ , and try to make  $G$  Eulerian by adding edges with a minimum total cost.

### 5.1 The ILP Formulation

Let  $\mathcal{E}$  be the set of edges  $(u, v)$  between nodes  $u, v \in V$  for which there is a path from  $u$  to  $v$  in  $G_{\Sigma,k}$ . This is possible for every pair of nodes in  $V$ . We define the set  $\mathcal{V}^-(u)$  of in-neighbors of node  $u \in V$  as  $\mathcal{V}^-(u) = \{v \in V \mid (v, u) \in \mathcal{E}\}$ . Similarly, we define the set  $\mathcal{V}^+(u)$  of out-neighbors of node  $u \in V$  as  $\mathcal{V}^+(u) = \{v \in V \mid (u, v) \in \mathcal{E}\}$ . We also define a weight function  $W : \mathcal{E} \rightarrow \mathbb{Z}_{\geq 0}$ , which assigns a weight  $w_{u,v}$  to an edge  $(u, v) \in \mathcal{E}$  equal to the length of the shortest directed path from  $u$  to  $v$  in  $G_{\Sigma,k}$ . In particular, we have  $w_{u,v} = k - 1 - \text{MO}(u, v)$ , where  $k$  is the order of  $G$  and  $\text{MO}(u, v)$  is the length of the longest overlap between a suffix of  $u$  and a prefix of  $v$ . For example, for edge  $(aabc, bcac) \in \mathcal{E}$  with  $aabc, bcac \in V$  and  $k = 5$ , we have  $w_{aabc, bcac} = k - 1 - |bc| = 5 - 1 - 2 = 2$ , corresponding to the following two unit-cost edges:  $aabc \rightarrow abca \rightarrow bcac$ . All  $\Theta(|V|^2)$  weights of the  $|V|$  nodes can be precomputed in the optimal  $\mathcal{O}(k|V| + |V|^2)$  time [12, 21]. By Euler's theorem, making  $G$  Eulerian (resp. semi-Eulerian) reduces to finding a minimum weight multiset of edges  $\mathcal{A}'$  from  $\mathcal{E}$  such that  $G' = (V, E \cup \mathcal{A}')$  is weakly connected and balanced (resp. semi-balanced).

To compute multiset  $\mathcal{A}'$ , we employ the ILP formulation presented in Figure 5. Each edge  $(u, v)$  is associated to a non-negative integer variable  $a_{u,v}$  whose value corresponds to the increase of the multiplicity  $m_{u,v}$  of  $(u, v)$  (where  $m_{u,v} = 0$  for any  $(u, v) \in \mathcal{E} \setminus E$ ). Thus,  $a_{u,v} + m_{u,v}$  denotes the actual multiplicity of  $(u, v)$  in  $G'$ . Each node  $v$  is associated to binary variables  $x_v$  and  $y_v$  which determine if  $v$  is a source and/or a target node in an Eulerian trail of  $G'$ , respectively. Specifically, if  $x_v = 1, y_v = 0$ , then  $v$  is a source node and not a target node; if  $x_v = 0, y_v = 1$  then  $v$  is a target node but not a source; and if  $x_v = y_v = 0$  then  $v$  is either (i) both a source and a target node in an Eulerian trail; or (ii) neither a source nor a target node in an Eulerian trail. Due to the constraint in Equation (2e), it cannot be that  $x_v = y_v = 1$ , and due to the constraints in Equation (2c) and Equation (2d) there is up to one source and one target node in  $G'$ .

Since the existence of a single component in  $G'$  is necessary for  $G'$  to be Eulerian or semi-Eulerian, we introduce a non-negative integer variable  $b_{u,v}$  for each edge  $(u, v) \in \mathcal{E}$  to check the connectivity of  $G'$ , through constraints that will be explained in detail later. Let us provide the main idea behind modeling connectedness. Suppose there are  $d = r + 1$  components in  $G$ . We select one node from each component of  $G$  arbitrarily, such that we have a set  $\mathcal{S} = \{s_1, \dots, s_r\}$  of start nodes and one destination node which we denote by  $dn$ . Let  $C_i$  be the component containing  $s_i$ , where  $i \in [1, r]$ , and  $C_{dn}$  be the component

<sup>3</sup> Note that both Eulerian and semi-Eulerian graphs are required to be weakly connected.

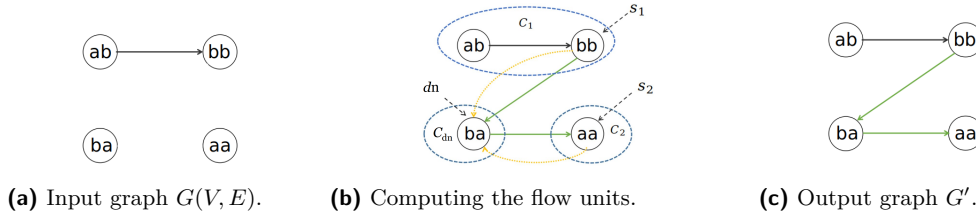
## 6:12 Connecting de Bruijn Graphs

$$\begin{aligned}
& \text{minimize} && \sum_{(u,v) \in \mathcal{E}} a_{u,v} \cdot w_{u,v} && (2a) \\
& \text{subject to} && \sum_{u \in \mathcal{V}^-(v)} (a_{u,v} + m_{u,v}) - \sum_{u \in \mathcal{V}^+(v)} (a_{v,u} + m_{v,u}) + x_v - y_v = 0, && v \in V && (2b) \\
& && 0 \leq \sum_{v \in V} x_v \leq 1, && && (2c) \\
& && \sum_{v \in V} x_v = \sum_{v \in V} y_v, && && (2d) \\
& && x_v + y_v \leq 1, && v \in V && (2e) \\
& && b_{u,v} \leq r \cdot (a_{u,v} + m_{u,v} + a_{v,u} + m_{v,u}), && (u,v) \in \mathcal{E} && (2f) \\
& && \sum_{v \in \mathcal{V}^+(s_i)} b_{s_i,v} - \sum_{u \in \mathcal{V}^-(s_i)} b_{u,s_i} = 1 && \forall i \in [1, r] && (2g) \\
& && \sum_{u \in \mathcal{V}^-(dn)} b_{u,dn} - \sum_{v \in \mathcal{V}^+(dn)} b_{dn,v} = r && && (2h) \\
& && \sum_{u \in \mathcal{V}^-(v)} b_{u,v} = \sum_{u \in \mathcal{V}^+(v)} b_{v,u} && v \in V \setminus (S \cup \{dn\}) && (2i) \\
& && b_{u,v} \in \mathbb{Z}_{\geq 0}, && (u,v) \in \mathcal{E} && (2j) \\
& && a_{u,v} \in \mathbb{Z}_{\geq 0}, && (u,v) \in \mathcal{E} && (2k) \\
& && x_v \in \{0, 1\}, && v \in V && (2l) \\
& && y_v \in \{0, 1\}, && v \in V && (2m)
\end{aligned}$$

■ **Figure 5** The complete ILP formulation for making a dBG Eulerian or semi-Eulerian.

containing  $dn$ .  $C_i$  and  $C_{dn}$  are connected if there exists a (positive) flow from  $s_i$  to  $dn$ . Assume that each connection between  $C_i$  and  $C_{dn}$  provides one unit of flow. There are  $r$  start nodes in  $G$ , so  $dn$  must absorb  $r$  units of flow in total from all start nodes.

Equation (2a) seeks to minimize the cost of multiset  $\mathcal{A}'$  (the sum of weights for all edges added to  $G$  to make it Eulerian or semi-Eulerian). All other equations seek to guarantee that the graph  $G'$  is Eulerian or semi-Eulerian by ensuring that all its nodes are balanced (Equation (2b) to Equation (2e)) and that all its nodes with nonzero degree belong to a single strongly connected component (Equation (2f) to Equation (2i)). Let  $\delta^-(v)$  and  $\delta^+(v)$  denote the in- and out-degree of node  $v$ , respectively. Recall that a weakly connected graph is Eulerian if  $\delta^-(v) = \delta^+(v)$  for each  $v \in V$ , and semi-Eulerian if  $\delta^-(s) = \delta^+(s) - 1$ ,  $\delta^-(t) = \delta^+(t) + 1$ , and  $\delta^-(v) = \delta^+(v)$  for each  $v \in V \setminus \{s, t\}$ , where  $s$  and  $t$  are the source and target nodes, respectively. Equation (2b) enforces that  $G'$  is Eulerian by requiring  $x_v = 0, y_v = 0$  such that  $\delta^-(v) = \delta^+(v)$  for each  $v \in V$ , or that  $G'$  is semi-Eulerian by requiring  $x_s = 1, y_s = 0$  for source node,  $x_t = 0, y_t = 1$  for target node and  $x_v = 0, y_v = 0$  for all other nodes, respectively. Equation (2c), Equation (2d) and Equation (2e) enforce that there exists at most only one source node and one target node in  $G'$ . Equation (2f) bounds the value of  $b_{u,v}$ . In particular, if nodes  $u$  and  $v$  are not connected in  $G'$  (i.e.,  $a_{u,v} + m_{u,v} + a_{v,u} + m_{v,u} = 0$ ), then Equation (2f) together with Equation (2j) ensure that both  $b_{u,v} = 0$  and  $b_{v,u} = 0$ ; otherwise,  $b_{u,v} \geq 0$  and  $b_{v,u} \geq 0$ . Equation (2g) enforces that each  $s_i$  provides one unit of flow; and Equation (2h) enforces that  $dn$  absorbs  $r$  units of flow from all  $s_i$ 's together. Last, Equation (2i) enforces that the amount of flow that enters each node that is not in  $S \cup \{dn\}$  is equal to the amount of flow that exits this node.



■ **Figure 6** An example of making the DBG on the left semi-Eulerian. The units of flow are depicted with yellow edges and the edges we add to make the graph semi-Eulerian are colored green.

► **Example 10.** Consider the subgraph  $G(V, E)$  of the complete order-3 DBG shown in Figure 6a, where  $V = \{aa, ab, ba, bb\}$  and  $E = \{(ab, bb)\}$ . By adding edge  $(bb, ba)$  and edge  $(ba, aa)$  in  $G$ , we find an Eulerian trail where the source node is  $ab$  and the target node is  $aa$ . The weights of the added edges are both 1, i.e.,  $w_{bb,ba} = w_{ba,aa} = 1$ , and the total cost of this solution is 2, which is minimal (since anyway we must connect  $d = 3$  connected components).

We show that the solution we found satisfies the constraints of the ILP from Figure 5. For the source node  $ab$ , we have  $x_{ab} = 1$ ,  $y_{ab} = 0$ , and it is semi-balanced since  $\delta^-(ab) - \delta^+(ab) + x_{ab} - y_{ab} = 0 - 1 + 1 - 0 = 0$ . Similarly, for the target node  $aa$ , we have  $x_{aa} = 0$ ,  $y_{aa} = 1$ , and it is semi-balanced. For all other nodes,  $v \in \{bb, ba\}$ ,  $\delta^-(v) = \delta^+(v) = 1$  and  $x_v = y_v = 0$ . Also, the values of  $x_v$  and  $y_v$ ,  $\forall v \in V$ , satisfy Equation (2c) to Equation (2e).

Next, we show the connectivity of  $G'$ . There are  $d = 3$  components in  $G$  (namely,  $C_1, C_2$  and  $C_{dn}$ ). We select one node from each component arbitrarily, such that we have  $\mathcal{S} = \{bb, aa\}$  and  $dn = ba$ . The destination node  $dn$  needs to absorb two units of flow from  $C_1$  and  $C_2$ . Since  $a_{ba,aa} = 1 > 0$ ,  $b_{aa,ba} \geq 0$ , there exists a flow starting at node  $aa$  and ending at node  $dn$  (Equation (2f)). Similarly, node  $dn$  absorbs another unit of flow from  $bb$ ; the two units of flow are represented with yellow lines in Figure 6b. Thus, the output graph  $G'$  is semi-Eulerian: the source node is  $ab$  and the target node is  $aa$ ; see Figure 6c.

## 5.2 Proof-of-concept Experiments

We implemented our ILP formulation in C++ using Gurobi 9.5.2. We will refer to this algorithm as ILP. Our code is available at <https://bitbucket.org/eulerian-ext/cpm2024/>.

We present proof-of-concept experiments using ILP on small samples of the *Staphylococcus aureus* (STA) whole-genome shotgun benchmark dataset. This dataset is available from <http://gage.cbcb.umd.edu/data/index.html>. The number of reads in the STA dataset is 1, 294, 104 (Library 1), the average read length is 101 base pairs (bp) and the insert length is 180bp. All experiments ran on an AMD EPYC 7702 CPU with 256GB RAM.

We first applied ILP on DBGs of varying order  $k$ . We constructed these order- $k$  DBGs, one for each  $k$  value, using the first 10 reads of STA. To compute the weight  $w_{u,v}$  of each edge given as input to ILP, we used the implementation of [32] for computing  $\text{MO}(u, v)$ ; and then set  $w_{u,v} = k - 1 - \text{MO}(u, v)$ . Observe in Table 1a that, as expected, increasing  $k$  slightly reduced the number of edges  $|E|$  of the input DBG and that it also generally increased the number of connected components. As can be seen, making a graph with more connected components Eulerian incurred a larger total cost and required more time. For example, for  $k = 8$ , there are 2 components extended to an Eulerian graph with a cost of 47 in less than 30 minutes, while for  $k = 13$ , there are 10 components extended to an Eulerian graph with a cost of 92 in about 6.5 hours. This is because when  $r$  increases, the number of distinct possible combinations of values of the variables  $b_{u,v}$  increases exponentially with  $d = r + 1$ ,

■ **Table 1** Runtime of ILP on dBGs with varying  $k$  and  $|E|$  on the STA dataset. Note that the time to compute all constants in ILP is not included in the reported runtimes.

(a) Runtime of ILP on dBGs with varying  $k$  constructed from the first 10 reads of the STA dataset.

$k$	$ V $	$ E $	$d$	Cost	Time (s)
8	709	724	2	47	1,766
9	720	714	7	56	9,260
10	713	704	9	65	24,378
11	704	694	10	78	26,982
12	694	684	10	87	34,989
13	684	674	10	92	23,119

(b) Runtime of ILP on dBGs with varying number  $|E|$  of edges constructed from the first 8, 9, 10, 11, and 12 reads of the STA dataset.

$k$	$ V $	$ E $	$d$	Cost	Time (s)
9	562	554	8	43	13,779
9	641	634	7	49	9,205
9	720	714	7	56	11,748
9	798	794	7	61	13,183
9	876	875	6	65	19,751

as each possible combination corresponds to a different weighted spanning tree of the  $d$  components; and there are exponentially many possible spanning trees. In other words, there are exponentially many ways to form the sums in Equations 2g, 2h, and 2i.

We then applied ILP on dBGs of fixed order  $k = 9$  and varying number  $|E|$  of edges. We started with a dBG  $G_1$ , constructed as explained before from the first 8 reads of STA with  $k = 9$ .  $G_1$  corresponds to the first row in Table 1b. Then, we constructed dBGs  $G_2$ ,  $G_3$ ,  $G_4$ , and  $G_5$ , with a larger number of edges than  $G_1$ , by adding into  $G_1$  nodes and edges corresponding to the next 1, 2, 3, and 4 reads in STA, respectively. That is,  $G_2$  is constructed from the first 9 reads of STA with  $k = 9$ . Observe in Table 1b that, as expected, increasing  $|E|$  also increases  $|V|$  and generally reduces the number of components. As expected, making a graph with more edges Eulerian, while keeping the number of components the same, incurred a larger total cost and required more time. Indeed, the main factor that determines the runtime is the number of components. For example, it took 50% more time to make the dBG in the first row of Table 1b Eulerian compared to the time in the second row of the same table because the former has more components, although it has fewer edges and nodes.

These results show that despite the NP-hardness of the problem, ILP can be used to obtain optimal solutions for small graphs within a reasonable amount of time. These graphs may be specific subgraphs of a much larger graph that need to be made Eulerian.

---

## References

- 1 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.
- 2 Per Austrin, Subhash Khot, and Muli Safra. Inapproximability of vertex cover and independent set in bounded degree graphs. In *2009 24th Annual IEEE Conference on Computational Complexity*, pages 74–80, 2009. doi:10.1109/CCC.2009.38.
- 3 Giulia Bernardini, Huiping Chen, Gabriele Fici, Grigorios Loukides, and Solon P. Pissis. Reverse-safe data structures for text indexing. In *Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 199–213. SIAM, 2020. doi:10.1137/1.9781611976007.16.
- 4 Giulia Bernardini, Huiping Chen, Gabriele Fici, Grigorios Loukides, and Solon P. Pissis. Reverse-safe text indexing. *ACM J. Exp. Algorithmics*, 26:1.10:1–1.10:26, 2021. doi:10.1145/3461698.
- 5 Giulia Bernardini, Huiping Chen, Grigorios Loukides, Solon P. Pissis, Leen Stougie, and Michelle Sweering. Making de Bruijn graphs Eulerian. In *33rd Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 223 of *LIPICs*, pages 12:1–12:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CPM.2022.12.

- 6 Giulia Bernardini, Alessio Conte, Estéban Gabory, Roberto Grossi, Grigorios Loukides, Solon P. Pissis, Giulia Punzi, and Michelle Sweering. On strings having the same length- $k$  substrings. In *33rd Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 223 of *LIPICs*, pages 16:1–16:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CPM.2022.16.
- 7 Giulia Bernardini, Alberto Marchetti-Spaccamela, Solon P. Pissis, Leen Stougie, and Michelle Sweering. Constructing strings avoiding forbidden substrings. In *32nd Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 191 of *LIPICs*, pages 9:1–9:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CPM.2021.9.
- 8 Karel Brinda, Michael Baym, and Gregory Kucherov. Simplitigs as an efficient and scalable representation of de Bruijn graphs. *Genome biology*, 22:1–24, 2021. doi:10.1186/s13059-021-02297-z.
- 9 Shiri Dori and Gad M. Landau. Construction of Aho Corasick automaton in linear time for integer alphabets. *Inf. Process. Lett.*, 98(2):66–72, 2006. doi:10.1016/j.ipl.2005.11.019.
- 10 Zvi Galil and Giuseppe F Italiano. Data structures and algorithms for disjoint set union problems. *ACM Computing Surveys (CSUR)*, 23(3):319–344, 1991. doi:10.1145/116873.116878.
- 11 John Gallant, David Maier, and James A. Storer. On finding minimal length superstrings. *J. Comput. Syst. Sci.*, 20(1):50–58, 1980. doi:10.1016/0022-0000(80)90004-5.
- 12 Dan Gusfield, Gad M. Landau, and Baruch Schieber. An efficient algorithm for the all pairs suffix-prefix problem. *Inf. Process. Lett.*, 41(4):181–185, 1992. doi:10.1016/0020-0190(92)90176-V.
- 13 John E. Hopcroft and Robert Endre Tarjan. Efficient algorithms for graph manipulation [H] (algorithm 447). *Commun. ACM*, 16(6):372–378, 1973. doi:10.1145/362248.362272.
- 14 George Karakostas. A better approximation ratio for the vertex cover problem. *ACM Trans. Algorithms*, 5(4):41:1–41:8, 2009. doi:10.1145/1597036.1597045.
- 15 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computation*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2\_9.
- 16 Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 767–775. ACM, 2002. doi:10.1145/509907.510017.
- 17 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.
- 18 Lawrence T. Kou, George Markowsky, and Leonard Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981. doi:10.1007/BF00288961.
- 19 Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956. doi:10.1090/S0002-9939-1956-0078686-7.
- 20 Zhen Liu. Optimal routing in the De Bruijn networks. Research Report RR-1130, INRIA, 1990. URL: <https://hal.inria.fr/inria-00075429>.
- 21 Grigorios Loukides and Solon P. Pissis. All-pairs suffix/prefix in optimal time using Aho-Corasick space. *Inf. Process. Lett.*, 178:106275, 2022. doi:10.1016/J.IPL.2022.106275.
- 22 Paul Medvedev, Konstantinos Georgiou, Gene Myers, and Michael Brudno. Computability of models for sequence assembly. In *7th WABI*, volume 4645 of *Lecture Notes in Computer Science*, pages 289–301. Springer, 2007. doi:10.1007/978-3-540-74126-8\_27.
- 23 Paul Medvedev and Mihai Pop. What do Eulerian and Hamiltonian cycles have to do with genome assembly? *PLOS Computational Biology*, 17(5):1–5, May 2021. doi:10.1371/journal.pcbi.1008928.
- 24 Jason R. Miller, Sergey Koren, and Granger Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, 2010. doi:10.1016/j.ygeno.2010.03.001.

## 6:16 Connecting de Bruijn Graphs

- 25 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991. doi:10.1016/0022-0000(91)90023-X.
- 26 Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proc Natl Acad Sci*, 98(17):9748–9753, 2001. doi:10.1073/pnas.171285098.
- 27 Robert C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957. doi:10.1002/j.1538-7305.1957.tb01515.x.
- 28 Amatur Rahman and Paul Medvedev. Representation of k-mer sets using spectrum-preserving string sets. *J. Comput. Biol.*, 28(4):381–394, 2021. doi:10.1089/CMB.2020.0431.
- 29 Sebastian Schmidt, Shahbaz Khan, Jarno N Alanko, Giulio E Pibiri, and Alexandru I Tomescu. Matchtigs: minimum plain text representation of k-mer sets. *Genome Biology*, 24(1):136, 2023. doi:10.1186/s13059-023-02968-z.
- 30 Sebastian S. Schmidt and Jarno N. Alanko. Eulertigs: minimum plain text representation of k-mer sets without repetitions in linear time. *Algorithms Mol. Biol.*, 18(1):5, 2023. doi:10.1186/S13015-023-00227-1.
- 31 Ondřej Sladký, Pavel Veselý, and Karel Břinda. Masked superstrings as a unified framework for textual k-mer set representations. *bioRxiv*, pages 2023–02, 2023.
- 32 William H.A. Tustumi, Simon Gog, Guilherme P. Telles, and Felipe A. Louza. An improved algorithm for the all-pairs suffix–prefix problem. *Journal of Discrete Algorithms*, 37:34–43, 2016. 2015 London Stringology Days and London Algorithmic Workshop (LSD & LAW). doi:10.1016/j.jda.2016.04.002.
- 33 Esko Ukkonen. A linear-time algorithm for finding approximate shortest common superstrings. *Algorithmica*, 5(3):313–323, 1990. doi:10.1007/BF01840391.