SN Computer Science (2022) 3:308
https://doi.org/10.1007/s42979-022-01198-7

ORIGINAL RESEARCH

Orienting Ordered Scaffolds: Complexity and Algorithms

Sergey Aganezov[1] · Pavel Avdeyev[2] · Nikita Alexeev[3] · Yongwu Rong[4] · Max A. Alekseyev[2]

Received: 20 September 2021 / Accepted: 7 May 2022 / Published online: 26 May 2022

Abstract
Despite the recent progress in genome sequencing and assembly, many of the currently available assembled genomes come in a draft form. Such draft genomes consist of a large number of genomic fragments (*scaffolds*), whose order and/or orientation (i.e., strand) in the genome are unknown. There exist various scaffold assembly methods, which attempt to determine the order and orientation of scaffolds along the genome chromosomes. Some of these methods (e.g., based on FISH physical mapping, chromatin conformation capture, etc.) can infer the order of scaffolds, but not necessarily their orientation. This leads to a special case of the *scaffold orientation problem* (i.e., deducing the orientation of each scaffold) with a known order of the scaffolds. We address the problem of orientating ordered scaffolds as an optimization problem based on given weighted orientations of scaffolds and their pairs (e.g., coming from pair-end sequencing reads, long reads, or homologous relations). We formalize this problem using notion of a scaffold graph (i.e., a graph, where vertices correspond to the assembled contigs or scaffolds and edges represent connections between them). We prove that this problem is NP-hard, and present a polynomial-time algorithm for solving its special case, where orientation of each scaffold is imposed relatively to at most two other scaffolds. We further develop a fixed-parameter tractable algorithm for the general case of the orientation of ordered scaffolds problem.

Keywords Genome assembly · Genome scaffolding · Scaffold orientation · Computational complexity · Algorithms

Introduction

While genome sequencing technologies are constantly evolving, they are still unable to read at once complete genomic sequences from organisms of interest. Instead, they produce a large number of rather short genomic fragments, called *reads*, originating from unknown locations and strands of the genome. The problem then becomes to assemble the reads into the complete genome. Existing genome assemblers usually assemble reads based on their overlap patterns and produce longer genomic fragments, called *contigs*, which are typically interweaved with highly polymorphic and/or repetitive regions in the genome. Contigs are further assembled into *scaffolds*, i.e., sequences of contigs interspaced with gaps.[1] Assembling scaffolds into larger scaffolds (ideally representing complete chromosomes) is called the *scaffold assembly problem*.

The scaffold assembly problem is known to be NP-hard [14, 17, 23, 29, 35], but there still exists a number of methods that use heuristic and/or exact algorithmic approaches to address it. The *scaffold assembly problem* consists of two subproblems:

1. determine the order of scaffolds (*scaffold order problem*); and
2. determine the orientation (i.e., strand of origin) of scaffolds (*scaffold orientation problem*).

This article is part of the topical collection "String Processing and Combinatorial Algorithms" guest edited by Simone Faro.

✉ Max A. Alekseyev
maxal@gwu.edu

[1] Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

[2] Department of Mathematics, The George Washington University, Washington, DC, USA

[3] International Laboratory "Computer technologies", ITMO University, Saint Petersburg, Russia

[4] Department of Mathematics, CUNY Queens College, Queens, NY, USA

[1] We remark that contigs can be viewed as a special type of scaffolds with no gaps.

Some methods attempt to solve these subproblems jointly by using various types of additional data including jumping libraries [11, 15, 20, 21, 25, 27, 32], long error-prone reads [6, 7, 12, 26, 34], homology relationships between genomes [1, 3–5, 24], etc. Other methods (typically based on wet-lab experiments [13, 22, 28, 30, 31, 33]) can often reliably reconstruct the order of scaffolds, but may fail to impose their orientation.

The scaffold orientation problem is also known to be NP -hard [10, 23]. Since the scaffold order problem can often be reliably solved with wet-lab based methods, this inspires us to consider the special case of the scaffold orientation problem with the given order of scaffolds, which we refer to as the *orientation of ordered scaffolds* (OOS) problem. We formulate the OOS as an optimization problem based on given weighted orientations of scaffolds and their pairs (e.g., coming from pair-end sequencing reads, long reads, or homologous relations). We prove that the OOS is NP-hard both in the case of linear genomes and in the case of circular genomes. We present a polynomial-time algorithm for solving the special case of the OOS, where the orientation of each scaffold is imposed relatively to at most two other scaffolds, and further generalize it to an FPT algorithm for the general OOS problem. The proposed algorithms are implemented in the CAMSA [2] software that have been developed for comparative analysis and merging of scaffold assemblies.

Background

We start with a brief description of the notation which have been used in CAMSA framework. The notation provides a unifying way to represent scaffold assemblies obtained by different methods.

Let $\mathbb{S} = \{s_i\}_{i=1}^n$ be the set of scaffolds. We represent an *assembly* of scaffolds as a *set* of *assembly points*. Each assembly point is formed by an adjacency between two scaffolds. Namely, an assembly point $p = (s_i, s_j)$ tells that the scaffolds s_i and s_j are adjacent in the assembly, where $s_i, s_j \in \mathbb{S}$. Additionally, we may know the orientation of either or both of the scaffolds and thus distinguish between three types of assembly points:

1. p is *oriented* if the orientation of both scaffolds s_i and s_j is known;
2. p is *semi-oriented* if the orientation of only one scaffold among s_i and s_j is known;
3. p is *unoriented* if the orientation of neither of s_i and s_j is known.

We denote the known orientation of scaffolds in assembly points by overhead arrows. While the right arrow corresponds to the original genomic sequence, the left arrow corresponds to the reverse complement of this sequence. For example, $(\vec{s_i}, \vec{s_j})$, $(\vec{s_i}, s_j)$, and (s_i, s_j) are oriented, semi-oriented, and unoriented assembly points, respectively. We remark that assembly points $(\vec{s_i}, \vec{s_j})$ and $(\overleftarrow{s_j}, \overleftarrow{s_i})$ represent the same adjacency between oriented scaffolds; to make this representation unique we will require that in all assembly points (s_i, s_j) we have $i < j$. Another way to represent the orientation of the scaffolds in an assembly point is by using superscripts h and t denoting the head and tail extremities of the scaffold's genomic sequence, e.g., $(\vec{s_i}, \vec{s_j})$ can also be written as (s_i^h, s_j^t).

We will need an auxiliary function $\mathrm{sn}(p, i)$ defined on an assembly point p and an index $i \in \{1, 2\}$ that returns the scaffold corresponding to the component i of p (e.g., $\mathrm{sn}((\vec{s_i}, \vec{s_j}), 2) = s_j$). We define a *realization* of an assembly point p as any oriented assembly point that can be obtained from p by orienting the unoriented scaffolds. We denote the set of realizations of p by $\mathrm{R}(p)$. When p is oriented, it has a single realization equal p itself (i.e., $\mathrm{R}(p) = \{p\}$); when p is semi-oriented, it has two realizations (i.e., $|\mathrm{R}(p)| = 2$); and when p is unoriented, it has four realizations (i.e., $|\mathrm{R}(p)| = 4$). For example,

$$\mathrm{R}((s_i, s_j)) = \left\{(\vec{s_i}, \vec{s_j}), (\vec{s_i}, \overleftarrow{s_j}), (\overleftarrow{s_i}, \vec{s_j}), (\overleftarrow{s_i}, \overleftarrow{s_j})\right\}. \quad (1)$$

An assembly point p is called a *refinement* of an assembly point q if $\mathrm{R}(p) \subset \mathrm{R}(q)$. From now on, we assume that no assembly point in a given assembly is a refinement of another assembly point (otherwise we simply discard the latter assembly point as less informative). We further assume that in a given assembly there are no two assembly points $(\vec{s_i}, s_j)$ and $(s_i, \vec{s_j})$ such that s_i or s_j belongs to yet another assembly point (otherwise[2] we simply replace $(\vec{s_i}, s_j)$ and $(s_i, \vec{s_j})$ with $(\vec{s_i}, \vec{s_j})$). Similarly, we assume that no assembly points $(\vec{s_i}, \vec{s_j})$, $(\overleftarrow{s_i}, \vec{s_j})$, $(\overleftarrow{s_i}, \overleftarrow{s_j})$ can be present in a given assembly at the same time. We refer to an assembly satisfying these assumptions as a *proper assembly*.

For a given assembly $\mathbb{A}$ we will use subscripts $u/s/o$ to denote the sets of unoriented/semi-oriented/oriented assembly points in $\mathbb{A}$ (e.g., $\mathbb{A}_u \subset \mathbb{A}$ is the set of all unoriented assembly points from $\mathbb{A}$). We also denote by $\mathbb{S}(\mathbb{A})$ the set of scaffolds appearing in the assembly points from $\mathbb{A}$.

We call two assembly points *overlapping* if they involve the same scaffold, and further call them *conflicting* if they involve the same extremity of this scaffold. We generalize this notion for semi-oriented and unoriented assembly points: two assembly points p and q are *conflicting* if all pairs of their realizations $(p', q') \in \mathrm{R}(p) \times \mathrm{R}(p)$ are conflicting. If some, but not all, pairs of the realizations are

[2] It can will be seen later that any assembly realization in this case is conflicting.

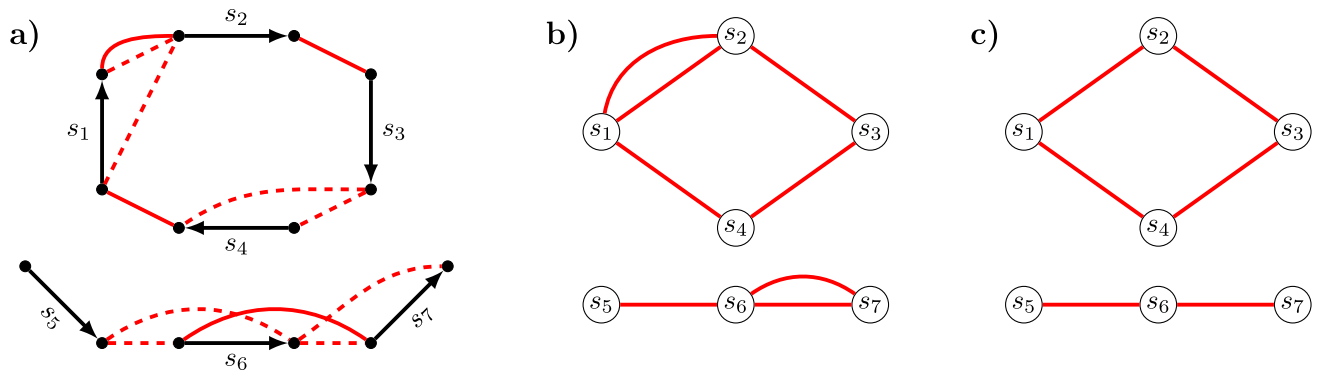

Fig. 1 For an assembly $A = \{(s_1, \overrightarrow{s_2}), (\overrightarrow{s_1}, \overrightarrow{s_2}), (\overrightarrow{s_2}, \overrightarrow{s_3}), (\overrightarrow{s_3}, s_4), (\overrightarrow{s_1}, \overrightarrow{s_4}), (\overrightarrow{s_5}, s_6), (\overrightarrow{s_6}, \overrightarrow{s_7}), (\overrightarrow{s_6}, s_7)\}$, (**a**) the scaffold assembly graph $\mathsf{SAG}(A)$, where semi-oriented assembly points, oriented assembly points, and scaffolds are represented by dashed red edges, solid red edges, and directed black edges, respectively. (**b**) The order graph $\mathsf{OG}(A)$. (**c**) The contracted order graph $\mathsf{COG}(A)$

conflicting, p and q are called *semi-conflicting*. Otherwise, p and q are called *non-conflicting*.

We extend the notion of non-/semi- conflictedness to entire assemblies as follows. A scaffold assembly $\mathbb{A}$ is *non-conflicting* if all pairs of assembly points in it are non-conflicting, and $\mathbb{A}$ is *semi-conflicting* if all pairs of assembly points are non-conflicting or semi-conflicting with at least one pair being semi-conflicting.

Methods

Assembly Realizations

For an assembly $\mathbb{A} = \{p_i\}_{i=1}^k$, an assembly $\mathbb{A}' = \{q_i\}_{i=1}^k$ is called a *realization*[3] of $\mathbb{A}$ if there exists a permutation π of order k such that $q_{\pi_i} \in \mathrm{R}(p_i)$ for all $i = 1, 2, \ldots, k$. We denote by $\mathrm{R}(\mathbb{A})$ the set of realizations of assembly $\mathbb{A}$, and by $\mathrm{NR}(\mathbb{A})$ the set of non-conflicting realizations among them.

We define the *scaffold assembly graph* $\mathsf{SAG}(\mathbb{A})$ on the set of vertices $\{s^h, s^t \ : \ s \in \mathbb{S}(\mathbb{A})\}$ and edges of two types: directed edges (s^t, s^h) that encode scaffolds from $\mathbb{S}(\mathbb{A})$, and undirected edges that encode all possible realizations of all assembly points in $\mathbb{A}$ (Fig. 1a). We further define the *order (multi)graph* $\mathsf{OG}(\mathbb{A})$ formed by the set of vertices $\mathbb{S}(\mathbb{A})$ and the set of undirected edges $\{\{\mathrm{sn}(p, 1), \mathrm{sn}(p, 2)\} \ : \ p \in \mathbb{A}\}$ (Fig. 1b). The order graph can also be obtained from $\mathsf{SAG}(\mathbb{A})$ by first contracting the directed edges, and then by substituting all edges that encode realizations of the same assembly point with a single edge (Fig. 1b). We define the *contracted order graph* $\mathsf{COG}(\mathbb{A})$ obtained from $\mathsf{OG}(\mathbb{A})$ by replacing all multi-edges edges with single edges (Fig. 1c).

[3] It can be easily seen that a realization of $\mathbb{A}$ may exist only if $\mathbb{A}$ is proper.

Let $\deg(v)$ be the degree of a vertex v in $\mathsf{OG}(\mathbb{A})$, i.e., the number of edges (counted with multiplicity) incident to v. We call the order graph $\mathsf{OG}(\mathbb{A})$ *non-branching* if $\deg(v) \leq 2$ for all vertices v of $\mathsf{OG}(\mathbb{A})$.

Lemma 1 *For a non-conflicting realization $\mathbb{A}'$ of an assembly $\mathbb{A}$, $\mathsf{OG}(\mathbb{A}')$ is non-branching.*

Proof Each vertex v in $\mathsf{OG}(\mathbb{A}')$ represents a scaffold, which has two extremities and thus can participate in at most two non-conflicting assembly points in $\mathbb{A}'$. Hence, $\deg(v) \leq 2$. □

We notice that any non-conflicting realization $\mathbb{A}'$ of an assembly $\mathbb{A}$ provides orientation for all scaffolds involved in each connected component of $\mathsf{SAG}(\mathbb{A}')$ (as well as of $\mathsf{OG}(\mathbb{A}')$ and $\mathsf{COG}(\mathbb{A}')$) relatively to each other.

Theorem 1 *An assembly $\mathbb{A}$ has at least one non-conflicting realization (i.e., $|\mathrm{NR}(\mathbb{A})| \geq 1$) if and only if $\mathbb{A}$ is non-conflicting or semi-conflicting and $\mathsf{OG}(\mathbb{A})$ is non-branching.*

Proof Suppose that $|\mathrm{NR}(\mathbb{A})| \geq 1$ and pick any $\mathbb{A}' \in \mathrm{NR}(\mathbb{A})$. Then for every pair of assembly points $p, q \in \mathbb{A}$, their realizations in $\mathbb{A}'$ are non-conflicting, implying that p and q are either non-conflicting or semi-conflicting. Hence, $\mathbb{A}$ is non-conflicting or semi-conflicting. Since $\mathbb{A}$ is a proper assembly, we have $\mathsf{OG}(\mathbb{A}) = \mathsf{OG}(\mathbb{A}')$. Taking into the account that $\mathbb{A}'$ is non-conflicting, Lemma 1 implies that $\mathsf{OG}(\mathbb{A})$ is non-branching.

Vice versa, suppose that $\mathbb{A}$ is non-conflicting or semi-conflicting and $\mathsf{OG}(\mathbb{A})$ is non-branching. To prove that $|\mathrm{NR}(\mathbb{A})| \geq 1$, we will orient unoriented scaffolds in all assembly points in $\mathbb{A}$ without creating conflicts. Every scaffold s corresponds to a vertex v in $\mathsf{OG}(\mathbb{A})$ of degree at most 2. If $\deg(v) = 1$, then s participates in one assembly point p, and s is either already oriented in p or we pick an arbitrary

orientation for it. If $\deg(v) = 2$, then s participates in two overlapping assembly points p and q. If s is not oriented in either of p, q, we pick an arbitrary orientation for it consistently across p and q (i.e., keeping them non-conflicting). If s is oriented in exactly one assembly point, we orient the unoriented instance of s consistently with its orientation in the other assembly point. Since conflicts may appear only between assembly points that share a vertex in $\mathsf{OG}(\mathbb{A})$, the constructed orientations produce no new conflicts. On other hand, the scaffolds that are already oriented in $\mathbb{A}$ impose no conflicts since $\mathbb{A}$ is non-conflicting or semi-conflicting. Hence, the resulting oriented assembly points form a non-conflicting assembly from $\mathrm{NR}(\mathbb{A})$, i.e., $|\mathrm{NR}(\mathbb{A})| \geq 1$.

□

We remark that if $\mathsf{OG}(\mathbb{A})$ is branching, the assembly $\mathbb{A}$ may be semi-conflicting but have $|\mathrm{NR}(\mathbb{A})| = 0$. An example is given by $\mathbb{A} = \{(s_1, s_{i+1})\}_{i=1}^{k}$ with $k > 2$, which contains no conflicting assembly points (in fact, all assembly points in $\mathbb{A}$ are semi-conflicting), but $|\mathrm{NR}(\mathbb{A})| = 0$.

From now on, we will always assume that assembly $\mathbb{A}$ has at least one non-conflicting realization (i.e., $|\mathrm{NR}(\mathbb{A})| \geq 1$). For an assembly $\mathbb{A}$, the orientation of some scaffolds from $\mathbb{S}(\mathbb{A})$ does not depend on the choice of a realization from $\mathrm{NR}(\mathbb{A})$ (we denote the set of such scaffolds by $\mathbb{S}_o(\mathbb{A})$), while the orientation of other scaffolds within some assembly points varies across realizations from $\mathrm{NR}(\mathbb{A})$ (we denote the set of such scaffolds by $\mathbb{S}_u(\mathbb{A})$). Trivially, we have $\mathbb{S}_u(\mathbb{A}) \cup \mathbb{S}_o(\mathbb{A}) = \mathbb{S}(\mathbb{A})$. It can be easily seen that the set $\mathbb{S}_u(\mathbb{A})$ is formed by the scaffolds for which the orientation in the proof of Theorem 1 was chosen arbitrarily, implying the following statement.

Corollary 1 *For a given assembly $\mathbb{A}$ with $|\mathrm{NR}(\mathbb{A})| \geq 1$, we have $|\mathrm{NR}(\mathbb{A})| = 2^{|\mathbb{S}_u(\mathbb{A})|}$.*

We label scaffolds from $\mathbb{S}(\mathbb{A})$ with integers $\{1, \ldots, |\mathbb{S}(\mathbb{A})|\}$. From computational perspective, we assume that we can get a scaffold from its name and vice verca in $\mathcal{O}(1)$ time.

Lemma 2 *Testing whether a given assembly $\mathbb{A}$ has a non-conflicting realization can be done in $\mathcal{O}(k)$ time, where $k = |\mathbb{S}(\mathbb{A})|$.*

Proof To test whether $\mathbb{A}$ has a non-conflicting realization, we first create a hash table indexed by $\mathbb{S}(\mathbb{A})$ that for every scaffold $s \in \mathbb{S}(\mathbb{A})$ will contain a list of assembly points that involve s. We iterate over all assembly points $p \in \mathbb{A}$ and add p to two lists in the hash table indexed by the scaffolds participating in p. If the length of some list becomes greater than 2, then $\mathbb{A}$ is conflicting and we stop. If we successfully complete the iterations, then every scaffold from $\mathbb{S}(\mathbb{A})$ participates in at most two assembly points in $\mathbb{A}$, and thus we made $\mathcal{O}(k)$ steps of $\mathcal{O}(1)$ time each.

Next, for every scaffold whose list in the hash table has length 2, we check whether the corresponding assembly points are either non-conflicting or semi-conflicting. If not, then $\mathbb{A}$ is conflicting and we stop. If the check completes successfully, then $\mathbb{A}$ has a non-conflicting realization by Theorem 1. The check takes $\mathcal{O}(k)$ steps of $\mathcal{O}(1)$ time each, and thus the total running time comes to $\mathcal{O}(k)$. □

A pseudocode for the test described in the proof of Lemma 2 is given in Algorithm 3 in the Appendix.

Lemma 3 *For a given assembly $\mathbb{A}$ with $|\mathrm{NR}(\mathbb{A})| \geq 1$, the set $\mathbb{S}_u(\mathbb{A})$ can be computed in $\mathcal{O}(k)$ time, where $k = |\mathbb{S}(\mathbb{A})|$.*

Proof We will construct the set $S = \mathbb{S}_u(\mathbb{A})$ iteratively. Initially we let $S = \emptyset$. Following the algorithm described in the proof for Lemma 2, we construct a hash table that for every scaffold $i \in \mathbb{S}(\mathbb{A})$ contains a list of assembly points that involve i (which takes $\mathcal{O}(k)$ time). Then for every $i \in \mathbb{S}(\mathbb{A})$, we check if either of the corresponding assembly points provides an orientation for i; if not, we add i to S. This check for each scaffolds takes $\mathcal{O}(1)$ time, bringing the total running time to $\mathcal{O}(k)$. □

A pseudocode for the computation of $\mathbb{S}_u(\mathbb{A})$ described in the proof of Lemma 3 is given in Algorithm 4 in the Appendix.

Problem Formulations

For a non-conflicting assembly $\mathbb{A}$ composed only of oriented assembly points, an assembly point p on scaffolds $s_i, s_j \in \mathbb{S}(\mathbb{A})$ has a *consistent orientation with* $\mathbb{A}$ if for some $p' \in \mathrm{R}(p)$ there exists a path connecting edges s_i and s_j in $\mathsf{SAG}(\mathbb{A})$ such that direction of edges s_i and s_j at the path ends is consistent with p' (e.g., in Fig. 1a, the assembly point $(\overrightarrow{s_1}, \overrightarrow{s_3})$ has a consistent orientation with the assembly $\mathbb{A}$). Furthermore, for a non-conflicting assembly $\mathbb{A}$ that has at least one non-conflicting realization, an assembly point p has a *consistent orientation with* $\mathbb{A}$ if p' has a consistent orientation with $\mathbb{A}'$ for some $p' \in \mathrm{R}(p)$ and $\mathbb{A}' \in \mathrm{NR}(\mathbb{A})$.

We formulate the orientation of ordered scaffolds problem as follows.

Orientation of Ordered Scaffolds (OOS) Let $\mathbb{A}$ be an assembly and $\mathbb{O}$ be a set[4] of assembly points such that $|\mathrm{NR}(\mathbb{A})| \geq 1$ and $\mathbb{S}(\mathbb{O}) \subset \mathbb{S}(\mathbb{A})$. Find a non-conflicting realization

[4] More generally, $\mathbb{O}$ may be a multiset whose elements have real positive multiplicities (weights).

$\mathbb{A}' \in \mathrm{NR}(\mathbb{A})$ that maximizes the number (total weight) of assembly points from $\mathbb{O}$ having consistent orientations with $\mathbb{A}'$.

From the biological perspective, the OOS can be viewed as a formalization of the case where (sub)orders of scaffolds have been determined (which defines $\mathbb{A}$), while there exists some information (possibly coming from different sources and conflicting) about their relative orientation (which defines $\mathbb{O}$). The OOS asks to orient unoriented scaffolds in the given scaffold orders in a way that is most consistent with the given orientation information.

We also remark that the OOS can be viewed as a fine-grained variant of the scaffold orientation problem studied in [10]. In our terminology, the latter problem concerns an artificial circular genome $\mathbb{A}$ formed by the given scaffolds in an arbitrary order (so that there is a path connecting any scaffold or its reverse complement to any other scaffold in $\mathsf{OG}(\mathbb{A})$), and $\mathbb{O}$ formed by unordered pairs of scaffolds supplemented with the binary information on whether each such pair come from the same or different strands of the genome. In contrast, in the OOS, the assembly $\mathbb{A}$ is given and $\mathsf{OG}(\mathbb{A})$ does not have to be connected or non-branching, while $\mathbb{O}$ may provide a pair of scaffolds with up to four options (as in (1)) of their relative orientation.

At the latest stages of genome assembly, the constructed scaffolds are usually of significant length. If (sub)orders for these scaffolds are known, it is rather rare to have orientation-imposing information that would involve non-neighboring scaffolds. Or, more generally, it is rather rare to have orientation imposing information for one scaffold with respect to more than two other scaffolds. This inspires us to consider a special case of the OOS problem:

Non-branching Orientation of Ordered Scaffolds (NOOS) Given an OOS instance $(\mathbb{A}, \mathbb{O})$ such that the graph $\mathsf{COG}(\mathbb{O})$ is non-branching. Find $\mathbb{A}' \in \mathrm{NR}(\mathbb{A})$ that maximizes the number of assembly points from $\mathbb{O}$ having consistent orientations with $\mathbb{A}'$.

NP-Hardness of the OOS

We consider two important partial cases of the OOS, where the assembly $\mathbb{A}$ represents a linear or circular genome up to unknown orientations of the scaffolds. In these cases, the graph $\mathsf{OG}(\mathbb{A})$ forms a collection of paths or cycles, respectively. Below we prove that the OOS in both these cases is NP-hard.

Lemma 4 *The OOS for linear genomes is* NP*-hard.*

Proof We will construct a polynomial-time reduction from the MAX 2-DNF problem, which is known to be NP-hard [8, 16]. An instance of MAX 2-DNF consists of clauses $C = \{c_i\}_{i=1}^{k}$ each formed by either a single variable or a conjunction of two variables from $X = \{x_i\}_{i=1}^{n}$, each of which may or may not be negated. The goal is to determine the maximum number of clauses that can be simultaneously satisfied by a 0/1 assignment to the variables from X. For a given instance $I = (C, X)$ of MAX 2-DNF, we construct an assembly

$$\mathbb{A} = \{(0, x_1)\} \cup \{(x_i, x_{i+1}) \ : \ i = 1, 2, \ldots, n-1\}.$$

Then we construct a set of assembly points $\mathbb{O}$ from the clauses in C as follows. For each conjunction $c \in C$ of variables x_i and x_j $(i < j)$, we add an oriented assembly point on scaffolds x_i, x_j to $\mathbb{O}$ with the orientation depending on the presence of negation of these variables in c (e.g., a conjunction $x_i \wedge \overline{x_j}$ is translated into an assembly point $(\overrightarrow{x_i}, \overleftarrow{x_j})$). For each clause $c \in C$ with a single variable x, we add an assembly point $(\vec{0}, \vec{x})$ or $(\vec{0}, \overleftarrow{x})$ depending whether x is negated in c.

It is easy to see that the constructed assembly $\mathbb{A}$ is semi-conflicting and $\mathsf{OG}(\mathbb{A})$ is a path, and thus by Theorem 1 $\mathbb{A}$ has a non-conflicting realization. Hence, $\mathbb{A}$ and $\mathbb{O}$ form an instance of the OOS for linear genomes. A solution $\mathbb{A}'$ to this OOS provides an orientation for each $x \in \mathbb{S}$ that maximizes the number of assembly points from $\mathbb{O}$ having consistent orientations with $\mathbb{A}'$. A solution to I is obtained from $\mathbb{A}'$ as the assignment of 0 or 1 to each variable x depending on whether the orientation of scaffold x in $\mathbb{A}'$ is forward or reverse. Indeed, since each assembly point in $\mathbb{O}$ having consistent orientation with $\mathbb{A}'$ corresponds to a truthful clause in I, the number of such clauses is maximized.

Since the OOS instance and the solution to I can be computed in polynomial time, the above construction represents a polynomial-time reduction from the MAX 2-DNF to the OOS for linear genomes. □

Lemma 5 *The OOS for circular genomes is* NP*-hard.*

Proof We construct a polynomial-time reduction from the MAX-CUT problem, which is known to be NP-hard [18, 19]. An instance I of MAX-CUT for a given a graph (V, E) asks to partition the set of vertices $V = \{v_i\}_{i=1}^{n}$ into two disjoint subsets V_1 and V_2 such that the number of edges $\{u, v\} \in E$ with $u \in V_1$ and $v \in V_2$ is maximized. For a given instance I of MAX-CUT problem, we define the assembly

$$\mathbb{A} = \left\{(v_i, v_{i+1}) \ : \ i = 1, 2, \ldots, n-1\right\} \cup \left\{(v_1, v_n)\right\}$$

and the set of assembly points

$$\mathbb{O} = \left\{(\overrightarrow{v_i}, \overleftarrow{v_j}) \ : \ \{v_i, v_j\} \in E\right\}.$$

It is easy to see that $\mathbb{A}$ has a non-conflicting realization and $\mathsf{OG}(\mathbb{A})$ is a cycle, i.e., $\mathbb{A}$ and $\mathbb{O}$ form an instance of the OOS for circular genomes. A solution $\mathbb{A}'$ to this OOS instance provides orientations for all elements $\mathbb{S}(\mathbb{A}) = V$ that maximizes the number of assembly points from $\mathbb{O}$ having consistent orientations with $\mathbb{A}'$. A solution to I is obtained as the partition of V into two disjoint subsets, depending on the orientation of scaffolds in $\mathbb{A}'$ (forward vs reverse). Indeed, since each assembly point in $\mathbb{O}$ having a consistent orientation with $\mathbb{A}'$ corresponds to an edge from E whose endpoints belong to distinct subsets in the partition, the number of such edges is maximized.

It is easy to see that the OOS instance and the solution to I can be computed in polynomial time, thus we constructed a polynomial-time reduction from the MAX-CUT to the OOS for circular genomes. □

As a trivial consequence of Lemmas 4 and 5, we obtain that the general OOS problem is NP-*hard*.

Theorem 2 *The OOS is* NP-*hard*.

Properties of the OOS

In this subsection, we formulate and prove some important properties of the OOS.

Connected Components of $\mathsf{OG}(\mathbb{A})$

Below we show that an OOS instance can also be solved independently for each connected component of $\mathsf{OG}(\mathbb{A})$. We start with the following lemma that trivially follows from the definition of consistent orientation.

Lemma 6 *Let $\mathbb{A}$ be an assembly such that $|\mathrm{NR}(\mathbb{A})| \geq 1$. An assembly point on scaffolds $s_i, s_j \in \mathbb{S}(\mathbb{A})$ may have a consistent orientation with $\mathbb{A}$ only if both s_i and s_j belong to the same connected component in $\mathsf{OG}(\mathbb{A})$.*

Theorem 3 *Let $(\mathbb{A}, \mathbb{O})$ be an OOS instance, and $\mathbb{A} = \mathbb{A}_1 \cup \cdots \cup \mathbb{A}_k$ be the partition such that $\mathsf{OG}(\mathbb{A}_1), \ldots, \mathsf{OG}(\mathbb{A}_k)$ represent the connected components of $\mathsf{OG}(\mathbb{A})$. For each $i = 1, 2, \ldots, k$, define $\mathbb{O}_i = \{p \in \mathbb{O} : \mathrm{sn}(p, 1), \mathrm{sn}(p, 2) \in \mathbb{S}(\mathbb{A}_i)\}$ and let $\mathbb{A}'_i$ be a solution to the OOS instance $(\mathbb{A}_i, \mathbb{O}_i)$. Then $\mathbb{A}'_1 \cup \cdots \cup \mathbb{A}'_k$ is a solution to the OOS instance $(\mathbb{A}, \mathbb{O})$.*

Proof Lemma 6 implies that we can discard from $\mathbb{O}$ all assembly points that are formed by scaffolds from different connected components in $\mathsf{OG}(\mathbb{A})$. Hence, we may assume that $\mathbb{O} = \mathbb{O}_1 \cup \cdots \cup \mathbb{O}_k\}$.

Lemma 6 further implies that an assembly point from $\mathbb{O}_i$ may have a consistent orientation with $\mathbb{A}_j$ only if $i = j$. Therefore, any solution to the OOS instance $(\mathbb{A}, \mathbb{O})$ is formed by the union of solutions to the OOS instances $(\mathbb{A}_i, \mathbb{O}_i)$. □

Theorem 3 allows us focus on instances of the OOS, where $\mathsf{OG}(\mathbb{A})$ is connected and thus forms a path or a cycle (by Theorem 1).

Connected Components of $\mathsf{OG}(\mathbb{O})$

Below we show that an OOS instance can also be solved independently for each connected component of $\mathsf{OG}(\mathbb{O})$. We need the following lemma that trivially holds.

Lemma 7 *Let $\mathbb{A}$ be an assembly such that $|\mathrm{NR}(\mathbb{A})| \geq 1$, and s_i, s_j be scaffolds from the same connected component C in $\mathsf{OG}(\mathbb{A})$. Then an unoriented assembly point (s_i, s_j) has a consistent orientation with $\mathbb{A}$. Furthermore, if C is a cycle, then any semi-oriented assembly point on s_i, s_j has a consistent orientation with $\mathbb{A}$.*

By Lemma 7, we can assume that $\mathbb{O}$ does not contain any unoriented assembly points (i.e., $\mathbb{O} = \mathbb{O}_o \cup \mathbb{O}_s$). Furthermore, if $\mathsf{OG}(\mathbb{A})$ is a cycle, we can assume that $\mathbb{O} = \mathbb{O}_o$ (i.e., $\mathbb{O}$ consists of oriented assembly points only). We consider two cases depending on whether $\mathsf{OG}(\mathbb{A})$ forms a path or a cycle.

$\mathsf{OG}(\mathbb{A})$ *is a path.* Suppose that $\mathsf{OG}(\mathbb{A}) = (s_1, s_2, \ldots, s_n)$ is a path and $\mathbb{O} = \mathbb{O}_o \cup \mathbb{O}_s$. Let $\mathcal{C}$ be the set of connected components of $\mathsf{OG}(\mathbb{O})$.

Consider any $C \in \mathcal{C}$. Let $(s_{j_1}, \ldots, s_{j_m})$ be a vertex sequence of C such that $j_1 < j_2 < \cdots < j_m$, where m is the number of vertices in C. We define an assembly $\mathbb{A}_C$ such that $\mathsf{OG}(\mathbb{A}_C)$ is the path $(x, s_{j_1}, \ldots, s_{j_m}, y)$, where x and y are artificial vertices, and the assembly points in $\mathbb{A}_C$ (corresponding to the edges in $\mathsf{OG}(\mathbb{A}_C)$) are oriented or semi-oriented as follows.

- The edges $\{x, s_{j_1}\}$ and $\{s_{j_m}, y\}$ correspond to semi-oriented assembly points $(\vec{x}, s_{j_1})$ and $(s_{j_m}, \vec{y})$, respectively;
- For each $l \in \{1, \ldots, m-1\}$, orientation of the assembly point corresponding to the edge $\{s_{j_l}, s_{j_{l+1}}\}$ is imposed from the orientations of s_{j_l} and $s_{j_{l+1}}$ in the assembly points in $\mathbb{A}$ corresponding to the edges $\{s_{j_l}, s_{j_l+1}\}$ and $\{s_{j_{l+1}-1}, s_{j_{l+1}}\}$ at the ends of a path connecting s_{j_l} and $s_{j_{l+1}}$ in $\mathsf{SAG}(\mathbb{A})$. For example, assembly points $(\vec{s}_{j_l}, \vec{s}_{j_l+1})$ and $(\vec{s}_{j_{l+1}-1}, \overleftarrow{s}_{j_{l+1}})$ in $\mathbb{A}$ impose the assembly point $(\vec{s}_{j_l}, \overleftarrow{s}_{j_{l+1}})$ in $\mathbb{A}_C$.

We further define $\mathbb{O}_C$ as a set formed by the assembly points from C and the following assembly points. For each semi-oriented assembly point $p \in \mathbb{O}$ formed by scaffolds s_i and s_j $(i < j)$, $\mathbb{O}_C$ contains:

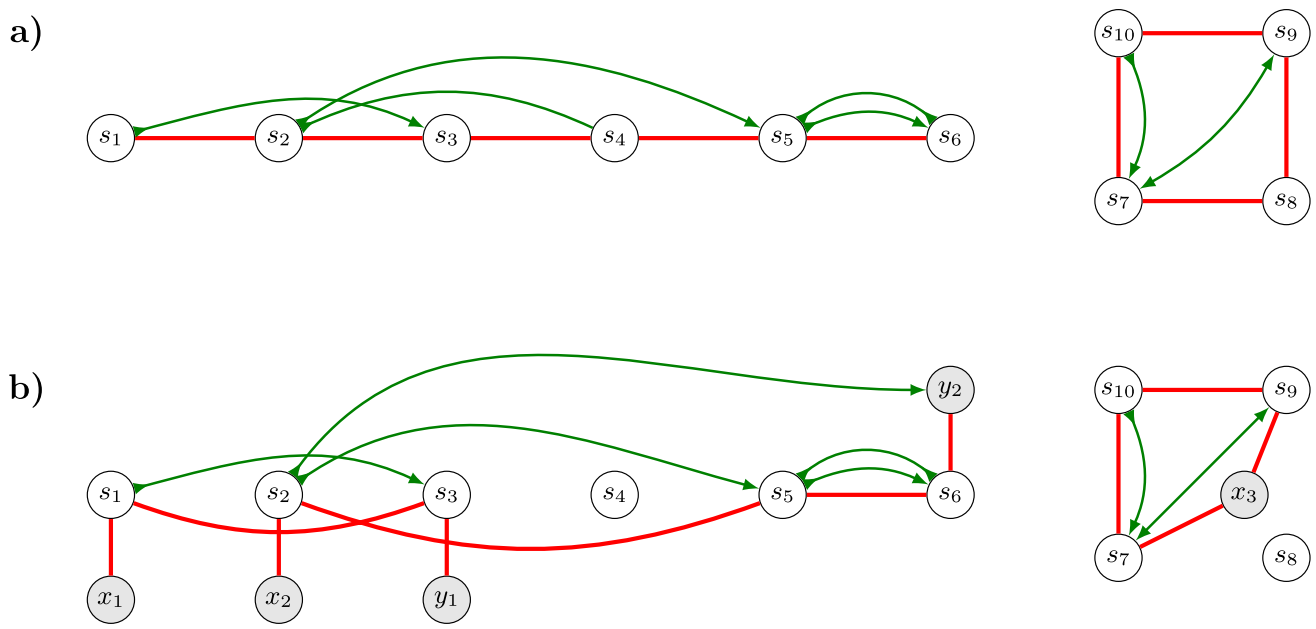

Fig. 2 Decomposition of an OOS problem instance $(\mathbb{A}, \mathbb{O})$ based on the connected components of $\mathsf{OG}(\mathbb{O}_o)$. (**a**) The superposition of $\mathsf{OG}(\mathbb{A})$ (red edges) and $\mathsf{OG}(\mathbb{O})$ (green edges), where arrows (if present) at the ends of green edges encode the orientation of the scaffolds in the corresponding assembly points. (**b**) The superposition of five graphs $\mathsf{OG}(\mathbb{A}_i)$ (red edges) and three graphs $\mathsf{OG}(\mathbb{O}_j)$ (green edges) constructed based on the connected components of $\mathsf{OG}(\mathbb{O}_o)$. Unless $\mathsf{OG}(\mathbb{A}_i)$ is formed by an isolated vertex, it contains artificial vertices x_i and y_i, which coincide if $\mathsf{OG}(\mathbb{A}_i)$ is a cycle

- an oriented point p' formed by s_i and $\vec{y}$ whenever s_i is oriented in p and belongs to C (and its orientation in p' is inherited from p);
- an oriented point p'' formed by $\vec{x}$ and s_j whenever s_j is oriented in p and belongs to C (and its orientation in p'' is inherited from p) (Fig. 2).

Now, for each $C \in \mathcal{C}$, we assume that $\mathbb{A}_C$ and $\mathbb{O}_C$ are defined as above and let $\mathbb{A}'_C$ be a solution to the OOS instance $(\mathbb{A}_C, \mathbb{O}_C)$. We construct a non-conflicting realization $\mathbb{A}' \in \mathrm{NR}(\mathbb{A})$ as follows:

- for a scaffold s *present* in some $\mathbb{A}'_C$, $\mathbb{A}'$ inherits the orientation of s from $\mathbb{A}'_C$;
- for a scaffold s *not present* in any $\mathbb{A}'_C$, if s is oriented in any assembly point of $\mathbb{A}$, then $\mathbb{A}'$ inherits that orientation of s; otherwise s is arbitrarily oriented in $\mathbb{A}'$.

The following theorem shows that constructed $\mathbb{A}'$ is a solution to the OOS instance $(\mathbb{A}, \mathbb{O})$.

Theorem 4 *Let* $(\mathbb{A}, \mathbb{O})$ *be an OOS instance, and* $\mathbb{A}' \in \mathrm{NR}(\mathbb{A})$ *be defined as above. Then* $\mathbb{A}'$ *is a solution to the OOS instance* $(\mathbb{A}, \mathbb{O})$.

Proof The graph $\mathsf{SAG}(\mathbb{A}')$ can be viewed as an ordered sequence of directed scaffold edges (interweaved with undirected edges encoding assembly points). Then each $\mathsf{SAG}(\mathbb{A}'_i)$, with the exception of scaffold edges x_i and y_i, corresponds to a subsequence of this sequence.

Each oriented assembly point $p \in \mathbb{O}$ is formed by scaffolds u, v from C_i for some $i \in \{1, \ldots, k\}$. Then $p \in \mathbb{O} \cap \mathbb{O}_i$ and there exist a unique path in $\mathsf{SAG}(\mathbb{A}'_i)$ and a unique path in $\mathsf{SAG}(\mathbb{A}')$ having the same directed edges u, v at the ends. Hence, if p has a consistent orientation with one of assemblies $\mathbb{A}'$ or $\mathbb{A}'_i$, then it has a consistent orientation with the other.

Each semi-oriented assembly point $p \in \mathbb{O}$ formed by scaffold u, v corresponds to an oriented assembly point $q \in \mathbb{O}_i$ (for some i) formed by u and y_i (in which case $u \in C_i$ and u is oriented in p), or by x_i and v (in which case $v \in C_i$ and v is oriented in p). Without loss of generality, we assume the former case. Then there exists a unique path Q in $\mathsf{SAG}(\mathbb{A}'_i)$ connecting directed edges u and y_i, and there exists a unique path P in $\mathsf{SAG}(\mathbb{A}')$ connecting directed edges u and v, where the orientation of u is the same in the two paths. By construction, the orientation of y_i in q matches that in Q. Hence, q has a consistent orientation with $\mathbb{A}'_i$ if and only if the orientation of u in q matches that in Q, which happens if and only if the orientation of u in p matches its orientation in P, i.e., p has a consistent orientation with $\mathbb{A}'$. We proved that the number of assembly points from $\mathbb{O}$ having consistent orientation with $\mathbb{A}'$ equals the total number of assembly points from $\mathbb{O}_i$ having consistent orientation with $\mathbb{A}'_i$ for

all $i = 1, 2, \ldots, k$. It remains to notice that this number is maximum possible, i.e., $\mathbb{A}'$ is indeed a solution to the OOS instance $(\mathbb{A}, \mathbb{O})$ (if it is not, then the sets $\mathbb{A}_i$ constructed from $\mathbb{A}$ being an actual solution to the OOS will give a better solution to at least one of the subproblems). □

$\mathsf{OG}(\mathbb{A})$ *is a cycle*. In this case, we can construct subproblems based on the connected components of $\mathsf{OG}(\mathbb{O})$ similarly to Case 1, with the following differences. First, by Lemma 7, we assume that $\mathbb{O} = \mathbb{O}_o$ (discarding all unoriented and semi-oriented assembly points from $\mathbb{O}$). Second, we assume that $x_i = y_i$ and thus $\mathsf{OG}(\mathbb{A}_i)$ forms a cycle. Theorem 4 still holds in this case.

Articulation Vertices in $\mathsf{OG}(\mathbb{O})$

While Theorem 4 allows us to divide the OOS problems into subproblems based on the connected components of $\mathsf{OG}(\mathbb{O})$, we show below that similar division is possible when $\mathsf{OG}(\mathbb{O})$ is connected but contains an articulation vertex.[5]

A vertex v in $\mathsf{OG}(\mathbb{O})$ (or in $\mathsf{COG}(\mathbb{O})$) is called *oriented* if $v \in \mathbb{S}_o(\mathbb{A})$. Otherwise, v is called *unoriented*. Let $(\mathbb{A}, \mathbb{O})$ be an instance of the OOS problem such that both $\mathsf{OG}(\mathbb{A})$ and $\mathsf{OG}(\mathbb{O})$ are connected. Let v be an oriented articulation vertex in $\mathsf{OG}(\mathbb{O})$, defining a partition of $\mathbb{S}(\mathbb{O})$ into disjoint subsets:

$$\mathbb{S}(\mathbb{O}) = \{v\} \cup V_1 \cup V_2 \cup \cdots \cup V_k, \tag{2}$$

where $k > 1$ and the V_i represent the vertex sets of the connected components resulted from removal of v from $\mathsf{OG}(\mathbb{O})$. To divide the OOS instance $(\mathbb{A}, \mathbb{O})$ into subinstances, we construct a new OOS instance $(\hat{\mathbb{A}}, \hat{\mathbb{O}})$ as follows.

We introduce copies $v_1, \ldots, v_k$ of v, and construct $\hat{\mathbb{A}}$ from $\mathbb{A}$ by replacing a path (u, v, w) in $\mathsf{OG}(\mathbb{A})$ with a path $(u, v_1, v_2, \ldots, v_k, w)$ where all v_i inherit the orientation from v. Then we construct $\hat{\mathbb{O}}$ from $\mathbb{O}$ by replacing in each assembly point p formed by v and $u \in V_i$ (for some $i \in \{1, 2, \ldots, k\}$) with an assembly point formed by v_i and u (keeping their orientations intact).

The OOS instance $(\hat{\mathbb{A}}, \hat{\mathbb{O}})$ enables application of Theorem 4. Indeed, by construction, the vertex sets of the connected components of $\mathsf{OG}(\hat{\mathbb{O}})$ are $\{v_i\} \cup V_i$, where $i \in \{1, 2, \ldots, k\}$. Hence, by Theorem 4 the OOS instance $(\hat{\mathbb{A}}, \hat{\mathbb{O}})$ can solved by dividing into OOS subinstances corresponding to the connected components of $\mathsf{OG}(\hat{\mathbb{O}})$.

Now, we assume that we have a solution to the OOS instance $(\hat{\mathbb{A}}, \hat{\mathbb{O}})$. We construct a non-conflicting realization $\mathbb{A}' \in \mathsf{NR}(\mathbb{A})$ from a solution to the OOS instance $(\hat{\mathbb{A}}, \hat{\mathbb{O}})$ by replacing every scaffold v_i with v.

[5] We remind that a vertex is *articulation* if its removal from the graph increases the number of connected components.

The following theorem shows that the constructed $\mathbb{A}'$ is a solution to the OOS instance $(\mathbb{A}, \mathbb{O})$.

Theorem 5 *Let* $(\mathbb{A}, \mathbb{O})$ *be an OOS instance such that both* $\mathsf{OG}(\mathbb{A})$ *and* $\mathsf{OG}(\mathbb{O})$ *are connected, and* $\mathbb{A}'$ *be defined as above. Then* $\mathbb{A}'$ *is a solution to the OOS instance* $(\mathbb{A}, \mathbb{O})$.

Proof Let $\hat{\mathbb{A}}'$ be a solution to the OOS instance $(\hat{\mathbb{A}}, \hat{\mathbb{O}})$, and $\mathbb{A}'$ be obtained from $\hat{\mathbb{A}}'$ by replacing every v_i with v. We remark that $\mathbb{O}$ can be obtained from $\hat{\mathbb{O}}$ by similar replacement.

This establishes an one-to-one correspondence between the assembly points in $\hat{\mathbb{A}}'$ and $\mathbb{A}'$, as well as between the assembly points in $\hat{\mathbb{O}}'$ and $\mathbb{O}'$. It remains to show that consistent orientations are invariant under this correspondence.

We remark that $\mathsf{SAG}(\mathbb{A}')$ can be obtained from $\mathsf{SAG}(\hat{\mathbb{A}}')$ by replacing a sequence of edges $(r_1, v_1, r_2, v_2, \ldots, r_k, v_k, r_{k+1})$, where r_i are assembly edges, with a sequence of edges (r_1, v, r_2). Therefore, if there exists a path in one graph proving existence of consistent orientation for some assembly point, then there exists a corresponding path in the other graph (having the same orientations of the end edges).

□

Algorithms for the NOOS and the OOS

In this section, by Theorems 3 and 4, we can assume that both $\mathsf{OG}(\mathbb{A})$ and $\mathsf{OG}(\mathbb{O})$ are connected.

A Polynomial-Time Algorithm for NOOS

Theorem 6 *The NOOS is in* P.

Proof Since $\mathsf{COG}(\mathbb{O})$ is non-branching, we consider two cases depending on whether it is a path or a cycle.

If $\mathsf{COG}(\mathbb{O})$ is a path, then every vertex in it is an articulation vertex in both $\mathsf{COG}(\mathbb{O})$ and $\mathsf{OG}(\mathbb{O})$. Our algorithm will process this path in a divide-and-conquer manner. Namely, for a path of length greater than 2, we pick a vertex v closest to the path middle. If v is oriented, we proceed as in Theorem 5. If v is unoriented, we fix each of the two possible orientations, proceed as in Theorem 5 to obtain two candidate solutions, from which we pick one with the larger number of assembly points with consistent orientations.

A path of length at most 2 can be solved in $\mathcal{O}(|\mathbb{O}|)$ time by brute-forcing all possible orientations of the scaffolds in the path and counting how many assembly points in $\mathbb{O}$ get consistent orientations.

The running time $T(l)$ for recursive part of the algorithm satisfies the formula:

$$T(l) = \begin{cases} 4 \cdot T\left(\frac{l}{2}\right) + \mathcal{O}(1), & \text{if } |\mathbb{O}| > 2; \\ \mathcal{O}(|\mathbb{O}|), & \text{if } |\mathbb{O}| \leq 2. \end{cases}$$

From the Master theorem [9], we conclude that the total running time for the proposed recursive algorithm is $\mathcal{O}\left(|\mathbb{O}|^2\right)$ (or $\mathcal{O}\left(|\mathbb{S}(\mathbb{A})|^2\right)$ since $\mathsf{COG}(\mathbb{O})$ is a path).

If $\mathsf{COG}(\mathbb{O})$ is a cycle, we can reduce the corresponding NOOS instance to the case of a path as follows. First, we pick a random vertex w in $\mathsf{COG}(\mathbb{O})$ and replace it with new vertices w_1 and w_2 such that the edges $\{u, w\}, \{w, v\}$ in $\mathsf{COG}(\mathbb{O})$ are replaced with $\{u, w_1\}, \{w_2, v\}$. Then we solve the NOOS for the resulting path one or two times (depending on whether $w \in \mathbb{S}_o(\mathbb{A})$): once for each of possible orientations of scaffold w (inherited by w_1 and w_2), and then select the orientation for w that produces the largest number of assembly points having consistent orientations with the input assembly. □

A pseudocode for the algorithm described in the proof of Theorem 6 is given in Algorithm 1 in the Appendix.

An Exact Algorithm for the OOS

Below we show how to solve OOS instance $(\mathbb{A}, \mathbb{O})$ in general case, i.e., when $\mathsf{COG}(\mathbb{O})$ is neither a path or a cycle.

First we assume that there are no articulation vertices in the $\mathsf{OG}(\mathbb{O})$, while the case when articulation vertices are present is addressed in the next section. Let $\mathrm{BV}(\mathbb{O})$ be the set of unoriented *branching vertices* (i.e., unoriented vertices of degree greater than 2) in $\mathsf{COG}(\mathbb{O})$. We define a non-branching path as a path for which the endpoints are in $\mathrm{BV}(\mathbb{O})$, and all internal vertices have degree 2 (e.g., $\{s_{18}, s_{23}, s_{24}, s_{25}\}$ is a non-branching path in Fig. 3a). Similarly, we define a non-branching cycle as a cycle in which all vertices have degree 2, except for one vertex (called *endpoint*) that belongs to $\mathrm{BV}(\mathbb{O})$ and thus has degree greater than 2 (e.g., $\{s_7, s_4, s_3, s_1, s_2, s_6, s_5, s_7\}$ is a non-branching cycle in Fig. 3a).

Each OOS instance induced by a non-branching path and a non-branching cycle in $\mathsf{COG}(\mathbb{O})$ represents an NOOS instance, and thus can be solved in polynomial time. We iterate over all possible orientations for the endpoints of the underlying paths/cycles in the corresponding NOOS instances and solve them. A solution to the OOS instance is obtained by iterating over all possible orientations of the scaffolds represented by branching vertices in $\mathsf{COG}(\mathbb{O})$ (i.e., $\mathrm{BV}(\mathbb{O})$) and merging the solutions to the corresponding NOOS instances, and picking the best result. Then, the following lemma trivially holds:

Lemma 8 *The running time for the proposed algorithm is bounded by* $\mathcal{O}\left(2^{|\,\mathrm{BV}(\mathbb{O})|} \cdot |\mathbb{S}(\mathbb{A})|^2\right)$.

An FPT Algorithm for the OOS

Thanks to Theorem 5, we can partition a given OOS instance $(\mathbb{A}, \mathbb{O})$ into subinstances using the oriented articulation vertices. By Theorem 6, we also know how to efficiently orient scaffolds that correspond to unoriented articulation vertices of degree 2. In this section, we address the remaining type of articulation vertices, namely unoriented articulation vertices of degree at least 3.

Let $\mathrm{AV}(\mathbb{O}) \subseteq \mathrm{BV}(\mathbb{O})$ be the set of unoriented articulation vertices of degree at least 3. A straightforward solution to this problem is to iterate over all possible $2^{|AV(\mathbb{O})|}$ orientations of the scaffolds in $\mathrm{AV}(\mathbb{O})$, and then use Theorem 5 to partition the OOS instance $(\mathbb{A}, \mathbb{O})$ into subinstances. Each such subinstance, in turn, can be solved using Theorem 6 or Lemma 8. Below we show how one can orient the scaffolds in $\mathrm{AV}(\mathbb{O})$ more efficiently based on the dependencies between the connected subgraphs flanked by the corresponding vertices.

The set $\mathrm{AV}(\mathbb{O})$ defines a set $\mathrm{C}(\mathbb{O})$ of connected subgraphs (*components*) of $\mathsf{COG}(\mathbb{O})$ by breaking it at the vertices from $\mathrm{AV}(\mathbb{O})$, introducing copies of each articulation vertex in the resulting components (Fig. 3a). We distinguish between two types of components in $\mathrm{C}(\mathbb{O})$:

- *path bridges* forming the set $\mathrm{PB}(\mathbb{O}) \subseteq \mathrm{C}(\mathbb{O})$, i.e., components that do not contain cycles (e.g., pb_1 in Fig. 1a);
- *complex components* forming the set $\mathrm{CC}(\mathbb{O}) \subseteq \mathrm{C}(\mathbb{O})$, i.e., components that contain at least one cycle (e.g., cc_2 in Fig. 3a).

Trivially we have $\mathrm{CC}(\mathbb{O}) \cup \mathrm{PB}(\mathbb{O}) = \mathrm{C}(\mathbb{O})$. We denote by $V(c)$ the set of vertices in a component $c \in \mathrm{C}(\mathbb{O})$. Now, we show how to solve the OOS instances induced by elements of $\mathrm{C}(\mathbb{O})$:

Case $c \in \mathrm{PB}(\mathbb{O})$ The OOS instance induced by c can be solved as follows. We iterate over all possible orientations of the unoriented articulation vertices in c (i.e., we need solve the OOS instance induced by c at most 4 times). For each fixed orientation, since c is non-branching, the OOS instance induced by c is an instance of NOOS and can be solved as in Theorem 6.

Case $c \in \mathrm{CC}(\mathbb{O})$ The OOS instance induced by c can be solved as follows. We iterate over all possible orientations of the vertices in $\mathrm{AV}(\mathbb{O}) \cap V(c)$. For each fixed orientation, a solution to the OOS instance induced by c can be obtained as in Theorem 8 by iterating over all possible orientations of the scaffolds represented by the unoriented branching vertices in c (i.e., $(\mathrm{BV}(\mathbb{O}) \setminus \mathrm{AV}(\mathbb{O})) \cap V(c)$).

Now, we outline how we iterate over the orientations of scaffolds in $\mathrm{AV}(\mathbb{O})$. Our algorithm constructs a *subproblem tree* $\mathrm{ST}(\mathbb{O}) = (V, E)$ (Fig. 3b), where $V = \mathrm{C}(\mathbb{O})$ is the set of vertices corresponding to the set of components

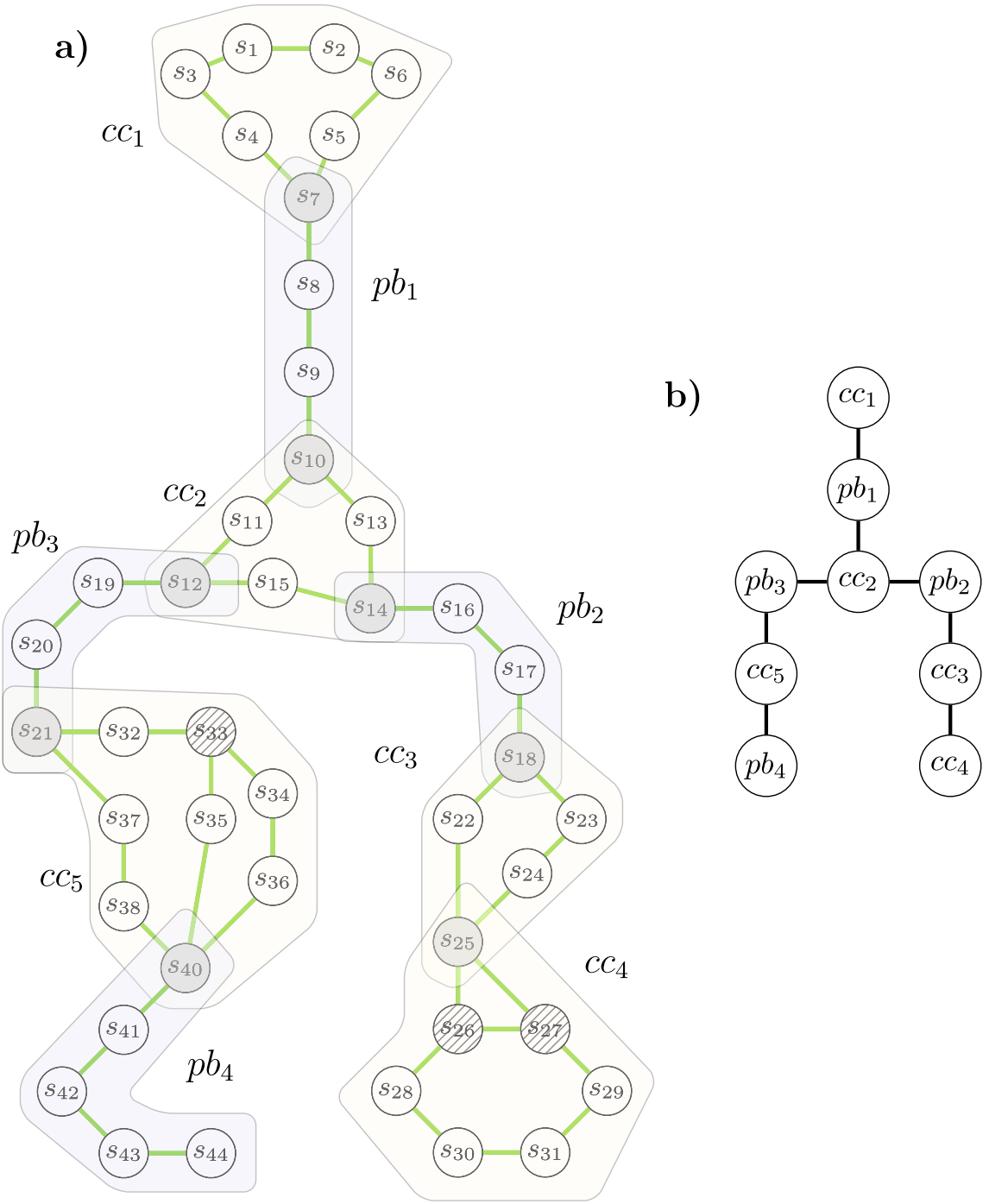

Fig. 3 (**a**) Contracted ordered graph $\mathsf{COG}(\mathbb{O})$ of a set of assembly points $\mathbb{O}$. Branching articulation vertices $\mathrm{AV}(\mathbb{O}) = \{s_7, s_{10}, s_{12}, s_{14}, s_{21}, s_{40}\}$ are shown as filled with gray. Branching vertices that are not articulation vertices $\mathrm{BV}(\mathbb{O}) \setminus \mathrm{AV}(\mathbb{O}) = \{s_{33}, s_{26}, s_{27}\}$ are shown as filled with line pattern. Yellow areas highlight elements of $\mathrm{CC}(\mathbb{O}) = \{cc_1, cc_2, cc_3, cc_4, cc_5\}$. Blue areas highlights elements of $\mathrm{PB}(\mathbb{O}) = \{pb_1, pb_2, pb_3, pb_4\}$. (**b**) The subproblem tree $\mathrm{ST}(\mathbb{O})$

induced by $\mathrm{AV}(\mathbb{O})$, and E is the set of edges constructed iteratively. We start with $E = \emptyset$ and populate E as follows: for each vertex $v \in V$ and all vertices $u \in V$, add an edge $\{v, u\}$ if the following two conditions hold:

1. v and u share an articulation vertex in $\mathsf{COG}(\mathbb{O})$ (e.g., cc_2 and pb_1 in Fig. 1a); and
2. u is not an endpoint of any edge in E.

A subproblem tree $\mathrm{ST}(\mathbb{O})$ allows us to solve the original OOS instance in the bottom-up fashion. Indeed, the OOS instance corresponding to any disjoint subtrees of $\mathrm{ST}(\mathbb{O})$ can be solved independently. We start with solving OOS instances that correspond to the leaves, producing solutions corresponding to different orientations of the scaffolds corresponding to articulation vertices. When the OOS instances for all children of an internal vertex c in $\mathrm{ST}(\mathbb{O})$ are solved, we iterate over the orientations for the scaffolds that correspond to articulation vertices in c (i.e., $\mathrm{AV}(\mathbb{O}) \cap V(c)$) and merge the OOS solutions for c with the corresponding solutions for its children. Eventually, we obtain the OOS solution for the root of $\mathrm{ST}(\mathbb{O})$ and thus for the original OOS problem.

The following theorem states the running time of the proposed algorithm.

Theorem 7 *The running time for the proposed algorithm for solving OOS instance* $(\mathbb{A}, \mathbb{O})$ *is bounded by*

$$\mathcal{O}\left(2^{\alpha} \cdot |\mathbb{S}(\mathbb{A})|^2 \cdot |\,\mathrm{CC}(\mathbb{O})|\right), \tag{3}$$

where $\alpha = \max_{c \in \mathrm{CC}(\mathbb{O})} |\,\mathrm{BV}(\mathbb{O}) \cap V(c)|$.

Proof The construction time of $\mathsf{SAG}(\mathbb{A})$, $\mathrm{AV}(\mathbb{O})$, $\mathrm{BV}(\mathbb{O})$, $\mathsf{OG}(\mathbb{O})$, $\mathsf{COG}(\mathbb{O})$, $\mathrm{ST}(\mathbb{O})$, and $\mathrm{C}(\mathbb{O})$ is bounded by $\mathcal{O}\left(|\mathbb{S}(\mathbb{A})|^2\right)$.

The OOS instances induced by each non-branching path or cycle in $\mathsf{COG}(\mathbb{O})$ are solved at most 4 times for different orientations of the endpoints. By Theorem 6, the total running time for processing all non-branching paths/cycles in $\mathsf{COG}(\mathbb{O})$ is bounded by $\mathcal{O}\left(|\mathbb{S}(\mathbb{A})|^2\right)$.

By Lemma 8, each OOS instance induced by a complex component $c \in \mathrm{CC}(\mathbb{O})$ can be solved in $\mathcal{O}\left(2^m \cdot |\mathbb{S}(\mathbb{A})|^2\right)$ time, where $m = |(\mathrm{BV}(\mathbb{O}) \setminus \mathrm{AV}(\mathbb{O})) \cap V(c)|$. The running time of the bottom-up algorithm is bounded by $|\,\mathrm{C}(\mathbb{O})|$ (i.e., the number of vertices in $\mathrm{ST}(\mathbb{O})$) times the running time of the merging procedure bounded by $\mathcal{O}\left(2^{|\,\mathrm{AV}(\mathbb{O}) \cap V(c)|} \cdot \deg(c)\right)$, where $\deg(c)$ is the degree of c in $\mathrm{ST}(\mathbb{O})$.

Thus, the proposed algorithm can be bounded by $\mathcal{O}\left(2^{\alpha} \cdot |\mathbb{S}(\mathbb{A})|^2 \cdot |\,\mathrm{CC}(\mathbb{O})|\right)$, where $\alpha = \max_{c \in \mathrm{CC}(\mathbb{O})} |\,\mathrm{BV}(\mathbb{O}) \cap V(c)|$. □

The proposed algorithm is an FPT algorithm. Indeed, instead of finding the best orientation by iterating over all possible orientations of the scaffolds in $\mathbb{S}_u(\mathbb{A})$, we iterate over all possible orientations of the scaffolds that correspond to branching vertices in $\mathsf{COG}(\mathbb{O})$. Furthermore, we reduced running time of an FPT algorithm by partitioning the problem into connected components and solving them independently.

The exponential term in (3) accounts for the number of articulation vertices in the complex components of $\mathsf{COG}(\mathbb{O})$. For real data, the exponent can become large only if many scaffolds have relative orientation with respect to three or more other scaffolds, which we expect to be a rare situation, especially when the scaffolds are long (e.g., produced by scaffolders combining paired-end and long-read data, a popular approach for the genome assembly).

Conclusions

In the present study, we posed the orientation of ordered scaffolds (OOS) problem as an optimization problem based on given weighted orientations of scaffolds and their pairs. We further addressed it within the earlier introduced CAMSA framework [2], taking advantage of the simple yet powerful concept of assembly points describing (semi-/un-) oriented adjacencies between scaffolds. This approach allows one to uniformly represent both orders of oriented and/or unoriented scaffolds and orientation-imposing data.

We proved that the OOS problem is NP-hard when the given scaffold order represents a linear or circular genome. We also described a polynomial-time algorithm for the special case of non-branching OOS (NOOS), where the orientation of each scaffold is imposed relatively to at most two other scaffolds. Our algorithm for the NOOS problem and Theorems 3, 4, and 5 further enabled us to develop an FPT algorithm for the general OOS problem. The proposed algorithms are implemented in the CAMSA software version 2 (https://github.com/compbiol/CAMSA).

Appendix: Pseudocodes

In the algorithms below we do not explicitly describe the function `OrConsCount`, which takes 4 arguments:

1. a subgraph c from $\mathsf{COG}(\mathbb{O})$ with 1 or 2 vertices;
2. a hash table *so* with scaffolds as keys and their orientations as values;
3. a set of orientation imposing assembly points $\mathbb{O}$;
4. an assembly $\mathbb{A}$

and counts the assembly points from $\mathbb{O}$ that have consistent orientation with $\mathbb{A}$ in the case where scaffold(s) corresponding to vertices from c were to have orientation from *so* in $\mathbb{A}$. With simple hash-table based preprocessing of $\mathbb{A}$ and $\mathbb{O}$ this function runs in $\mathcal{O}(n)$ time, where n is a number of assembly points in $\mathbb{O}$ involving scaffolds

that correspond to vertices in c. So, total running time for all invocations of this function will be $\mathcal{O}(|\mathbb{O}|)$ (i.e., $\mathcal{O}\big(|\mathbb{S}(\mathbb{A})|^2\big)$).

Algorithm 1 Solving the NOOS for complex component

```
1: function SolveNOOSforCC(c, 𝕆, so, 𝔸)
2:     score ← 0
3:     if c has less than 3 vertices then
4:         return OrConsCount(c, so, 𝕆, 𝔸)
5:     end if
6:     p1, p2 ← split c into two paths of equal length, at vertex s
7:     var ← empty hash table
8:     for or in {→, ←} do
9:         so[s] ← or
10:        var[or] = SolveNOOSforCC(p1, 𝕆, so)
11:        var[or] += SolveNOOSforCC(p2, 𝕆, so)
12:        var[or] += OrConsCount(s, so, 𝕆, 𝔸)
13:    end for
14:    score ← maximum value in var
15:    or ← key corresponding to the maximum value in var
16:    so[s] ← or
17:    return score
18: end function
```

Algorithm 2 Solving the NOOS for instance $(\mathbb{A}, \mathbb{O})$

```
1:
2: function SolveNOOS(𝔸, 𝕆)
3:     so ← hash table with 𝕊(𝔸) as keys and their orientation as values
4:     cog ← COG(𝕆_o)
5:     var ← empty hash table
6:     for each connected component c in cog do
7:         if c is a cycle then
8:             pick a random vertex w in c
9:             remove two edges {u, w}, {w, v} from c
10:            add two edges {u, w1}, {w2, u} to c
11:            for or in {→, ←} do
12:                so[w1], so[w2] ← or, or
13:                var[(w, or)] ← SolveNOOSforCC(c, 𝕆, so, 𝔸)
14:            end for
15:        else
16:            s1, s2 ← scaffolds, corresponding extremities of c
17:            for or1 in {→, ←} do
18:                for or2 in {→, ←} do
19:                    so[s1], so[s2] ← or1, or2
20:                    r ← SolveNOOSforCC(c, 𝕆, so, 𝔸)
21:                    var[(s1, or1)], var[(s2, or2)] ← r, r
22:                end for
23:            end for
24:        end if
25:    end for
26:    return var
27: end function
```

Algorithm 3 Checking if a given assembly has a non-conflicting realization

```
1: procedure HasNonConflictingRealization(𝔸)
2:     t ← hash table with 𝕊(𝔸) as keys and empty linked lists as values
3:     for p in 𝔸 do
4:         append p to the list at t[sn(p, 1)]
5:         append p to the list at t[sn(p, 2)]
6:         if length of list at t[sn(p, 1)] > 2 or length of list at t[sn(p, 2)] > 2 then
7:             return False
8:         end if
9:     end for
10:    for s in 𝕊(𝔸) do
11:        if length of list at t[s] = 2 and assembly points in t[s] are conflicting then
12:            return False
13:        end if
14:    end for
15:    return True
16: end procedure
```

Algorithm 4 Computing $\mathbb{S}_u(\mathbb{A})$

```
1: procedure UnorientedScaffolds(𝔸)
2:     S ← empty set
3:     if not HasNonConflictingRealization(𝔸) then
4:         return S
5:     end if
6:     t ← hash table with 𝕊(𝔸) as keys and empty linked lists as values
7:     for p in 𝔸 do
8:         append p to the list at t[sn(p, 1)]
9:         append p to the list at t[sn(p, 2)]
10:    end for
11:    for s in 𝕊(𝔸) do
12:        flag ← True
13:        for p in the list at t[s] do
14:            if p involves s with orientation then
15:                flag ← False
16:            end if
17:        end for
18:        if flag then
19:            add s to S
20:        end if
21:    end for
22:    return S
23: end procedure
```

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Aganezov S, Alekseyev MA. In: Bourgeois A, Skums P, Wan X, Zelikovsky A, editors. Multi-genome scaffold co-assembly based on the analysis of gene orders and genomic repeats, vol. 9683. Cham: Springer; 2016. pp. 237–49. https://doi.org/10.1007/978-3-319-38782-6_20.
2. Aganezov SS, Alekseyev MA. CAMSA: a tool for comparative analysis and merging of scaffold assemblies. BMC Bioinform. 2017;18(15):496. https://doi.org/10.1186/s12859-017-1919-y.
3. Anselmetti Y, Berry V, Chauve C, Chateau A, Tannier E, Bérard S. Ancestral gene synteny reconstruction improves extant species scaffolding. BMC Genom. 2015;16(Suppl 10):S11. https://doi.org/10.1186/1471-2164-16-S10-S11.
4. Assour LA, Emrich SJ. Multi-genome synteny for assembly improvement multi-genome synteny for assembly improvement. In: Proceedings of 7th international conference on bioinformatics and computational biology, 2015. pp. 193–199
5. Avdeyev P, Jiang S, Aganezov S, Hu F, Alekseyev MA. Reconstruction of ancestral genomes in presence of gene gain and loss. J Comput Biol. 2016;23(3):150–64. https://doi.org/10.1089/cmb.2015.0160.
6. Bankevich A, Nurk S, Antipov D, Gurevich AA, Dvorkin M, Kulikov AS, Lesin VM, Nikolenko SI, Pham S, Prjibelski AD, Pyshkin AV, Sirotkin AV, Vyahhi N, Tesler G, Alekseyev MA, Pevzner PA. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. J Comput Biol. 2012;19(5):455–77. https://doi.org/10.1089/cmb.2012.0021.
7. Bashir A, Klammer AA, Robins WP, Chin CS, Webster D, Paxinos E, Hsu D, Ashby M, Wang S, Peluso P, Sebra R, Sorenson J, Bullard J, Yen J, Valdovino M, Mollova E, Luong K, Lin S, LaMay B, Joshi A, Rowe L, Frace M, Tarr CL, Turnsek M, Davis BM, Kasarskis A, Mekalanos JJ, Waldor MK, Schadt EE. A hybrid approach for the automated finishing of bacterial genomes. Nat Biotechnol. 2012;30(7):701–7. https://doi.org/10.1038/nbt.2288.
8. Bazgan C, Paschos VT. Differential approximation for optimal satisfiability and related problems. Eur J Oper Res. 2003;147(2):397–404. https://doi.org/10.1016/S0377-2217(02)00299-0.
9. Bentley JL, Haken D, Saxe JB. A general method for solving divide-and-conquer recurrences. ACM SIGACT News. 1980;12(3):36–44. https://doi.org/10.1145/1008861.1008865.
10. Bodily PM, Fujimoto MS, Snell Q, Ventura D, Clement MJ. ScaffoldScaffolder: solving contig orientation via bidirected to directed graph reduction. Bioinformatics. 2015;32(1):17–24. https://doi.org/10.1093/bioinformatics/btv548.
11. Boetzer M, Henkel CV, Jansen HJ, Butler D, Pirovano W. Scaffolding pre-assembled contigs using SSPACE. Bioinformatics. 2011;27(4):578–9. https://doi.org/10.1093/bioinformatics/btq683.
12. Boetzer M, Pirovano W. SSPACE-LongRead: scaffolding bacterial draft genomes using long read sequence information. BMC Bioinform. 2014;15(1):211. https://doi.org/10.1186/1471-2105-15-211.
13. Burton JN, Adey A, Patwardhan RP, Qiu R, Kitzman JO, Shendure J. Chromosome-scale scaffolding of de novo genome assemblies based on chromatin interactions. Nat Biotechnol. 2013;31(12):1119–25. https://doi.org/10.1038/nbt.2727.
14. Chen ZZ, Harada Y, Guo F, Wang L. Approximation algorithms for the scaffolding problem and its generalizations. Theor Comput Sci. 2017. https://doi.org/10.1016/j.tcs.2017.03.042.
15. Dayarian A, Michael TP, Sengupta AM. SOPRA: scaffolding algorithm for paired reads via statistical optimization. BMC Bioinform. 2010;11:345. https://doi.org/10.1186/1471-2105-11-345.
16. Escoffier B, Paschos VT. Differential approximation of min sat, max sat and related problems. Eur J Oper Res. 2007;181(2):620–33. https://doi.org/10.1016/j.ejor.2005.04.057.
17. Gao S, Sung WK, Nagarajan N. Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. J Comput Biol. 2011;18(11):1681–91. https://doi.org/10.1089/cmb.2011.0170.
18. Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness, vol. 58. San Francisco: Freeman; 1979.
19. Garey MR, Johnson DS, Stockmeyer L. Some simplified NP-complete graph problems. Theor Comput Sci. 1976;1(3):237–67.
20. Gritsenko AA, Nijkamp JF, Reinders MJT, de Ridder D. GRASS: a generic algorithm for scaffolding next-generation sequencing assemblies. Bioinformatics. 2012;28(11):1429–37. https://doi.org/10.1093/bioinformatics/bts175.
21. Hunt M, Newbold C, Berriman M, Otto TD. A comprehensive evaluation of assembly scaffolding tools. Genome Biol. 2014. https://doi.org/10.1186/gb-2014-15-3-r42.
22. Jiao WB, Garcia Accinelli G, Hartwig B, Kiefer C, Baker D, Severing E, Willing EM, Piednoel M, Woetzel S, Madrid-Herrero E, Huettel B, Hümann U, Reinhard R, Koch MA, Swan D, Clavijo B, Coupland G, Schneeberger K. Improving and correcting the contiguity of long-read genome assemblies of three plant species using optical mapping and chromosome conformation capture data. Genome Res. 2017;27(5):116. https://doi.org/10.1101/gr.213652.116.
23. Kececioglu JD, Myers EW. Combinatorial algorithms for DNA sequence assembly. Algorithmica. 1995;13(1–2):7–51. https://doi.org/10.1007/BF01188580.
24. Kolmogorov M, Armstrong J, Raney BJ, Streeter I, Dunn M, Yang F, Odom D, Flicek P, Keane T, Thybert D, Paten B, Pham S. Chromosome assembly of large and complex genomes using multiple references. Preprint bioRxiv. 2016. https://doi.org/10.1101/088435.
25. Koren S, Treangen TJ, Pop M. Bambus 2: scaffolding metagenomes. Bioinformatics. 2011;27(21):2964–71. https://doi.org/10.1093/bioinformatics/btr520.
26. Lam KK, Labutti K, Khalak A, Tse D. FinisherSC: a repeat-aware tool for upgrading de novo assembly using long reads. Bioinformatics. 2015;31(19):3207–9. https://doi.org/10.1093/bioinformatics/btv280.
27. Luo R, Liu B, Xie Y, Li Z, Huang W, Yuan J, Wang J. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. Gigascience. 2012;1(1):18. https://doi.org/10.1186/2047-217X-1-18.
28. Nagarajan N, Read TD, Pop M. Scaffolding and validation of bacterial genome assemblies using optical restriction maps. Bioinformatics. 2008;24(10):1229–35. https://doi.org/10.1093/bioinformatics/btn102.
29. Pop M, Kosack DS, Salzberg SL. Hierarchical scaffolding with Bambus. Genome Res. 2004;14(1):149–59. https://doi.org/10.1101/gr.1536204.
30. Putnam NH, O'Connell BL, Stites JC, Rice BJ, Blanchette M, Calef R, Troll CJ, Fields A, Hartley PD, Sugnet CW, Haussler D, Rokhsar DS, Green RE. Chromosome-scale shotgun assembly using an in vitro method for long-range linkage. Genome Res. 2016;26(3):342–50. https://doi.org/10.1101/gr.193474.115.
31. Reyes-Chin-Wo S, Wang Z, Yang X, Kozik A, Arikit S, Song C, Xia L, Froenicke L, Lavelle DO, Truco MJ, Xia R, Zhu S, Xu C, Xu H, Xu X, Cox K, Korf I, Meyers BC, Michelmore RW. Genome assembly with in vitro proximity ligation data and whole-genome triplication in lettuce. Nat Commun. 2017. https://doi.org/10.1038/ncomms14953.

32. Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ, Birol I. ABySS: a parallel assembler for short read sequence data. Genome Res. 2009;19(6):1117–23. https://doi.org/10.1101/gr.089532.108.
33. Tang H, Zhang X, Miao C, Zhang J, Ming R, Schnable JC, Schnable PS, Lyons E, Lu J. ALLMAPS: robust scaffold ordering based on multiple maps. Genome Biol. 2015;16(1):3. https://doi.org/10.1186/s13059-014-0573-1.
34. Warren RL, Yang C, Vandervalk BP, Behsaz B, Lagman A, Jones SJM, Birol I. LINKS: scalable, alignment-free scaffolding of draft genomes with long reads. GigaScience. 2015;4(1):35. https://doi.org/10.1186/s13742-015-0076-3.
35. Zimin AV, Smith DR, Sutton G, Yorke JA. Assembly reconciliation. Bioinformatics. 2008;24(1):42–5. https://doi.org/10.1093/bioinformatics/btm542.