Complete Assembly of Circular and Chloroplast Genomes Based on Global Optimization

Rumen Andonov, Hristo Djidjev, Sébastien François, Dominique Lavenier

Complete Assembly of Circular and Chloroplast Genomes Based on Global Optimization*

Rumen Andonov [1†] Hristo Djidjev [2] Sebastien François [1]
Dominique Lavenier [1]
[1] Univ Rennes, Inria, CNRS, IRISA, F-35000 Rennes, France
[2] Los Alamos National Laboratory, Los Alamos, NM 87545, USA

January 17, 2019

Abstract

This paper focuses on the last two stages of genome assembly, namely scaffolding and gap-filling, and shows that they can be solved as part of a single optimization problem. Our approach is based on modeling genome assembly as a problem of finding a simple path in a specific graph that satisfies as many as possible of the distance-constraints encoding the insert-size information. We formulate it as a mixed-integer linear programming problem and apply an optimization solver to find the exact solutions on a benchmark of chloroplasts. We show that the presence of repetitions in the set of unitigs is the main reason for the existence of multiple equivalent solutions that are associated to alternative subpaths. We also describe two sufficient conditions and we design efficient algorithms for identifying these subpaths. Comparisons of the results achieved by our tool with the ones obtained with recent assemblers are presented.

keywords: de novo genome assembly, unitig, contig, scaffolding, gap-filling, weighted simple path problem, linear integer programming.

1 Introduction

Genome assembly is a challenging computational task, consisting in reconstructing the full genome from billions of short DNA sequences, called *reads*, that are generated by the modern Next-Generation Sequencing (NGS) techniques. This is a complex procedure, usually consisting of three main steps:

*Extended version of a paper published in the proceedings of the 10th International Conference on Bioinformatics and Computational Biology (BICOB), 2018.

†Corresponding author. Email: randonov@irisa.fr

(1) generation of *contigs* and/or *unitigs*, that are long contiguous DNA sequences issued from the overlapping of the reads; 2) constructing *scaffolds*– set of ordered and oriented contigs/unitigs along the genome interspaced with gaps; (3) *gap-filling* (also called *finishing*), that aims to complete the assembly by inserting DNA text in the gaps between the scaffolds. Most current genome assembly approaches consider the above mentioned steps as independent consecutive tasks and therefore do not propose an optimal global strategy.

The "easily assembled regions" of the genome are usually treated in the first step by methods using a particular data structure called *de-Bruijn graph* [21]. In our approach, we expect this step to output *unitigs* –a special kind of high-confidence contigs that represent longest non branching paths in the de-Bruijn graphs[15]. In spite of the progress done in this task, in the presence of repeats longer than the size of the reads, long regions of the genome fail to be assembled in a unique way.

As a result of the contig/unitig assembly done in the first step, the input for the second (scaffolding) step is of moderate size with respect to the number of unitigs. However, while algorithms of linear complexity exist for the first step [4], the second step needs to solve an NP-hard problem that asks for ordering and orientating the contigs for connecting them into *scaffolds*[12]. The novelty here consists in taking advantage of the ability of the sequencing technology to provide couples of reads (*paired-end* or *mate-pair*) that are separated by a known distance (called *insert size*) [25, 18]. This distance information is not used in the first assembly stage, but is essential for the second one. Consequently, longer genome subsequences are assembled here.

The scaffolding phase rarely generates the entire genome, but instead usually produces multiple scaffolds. In addition, these scaffolds may contain regions that have not been completely assembled. Two additional steps, *gap-filling* (filling the discontinuity between consecutive scaffolds using the gap length–also NP-hard in general [24]) and *scaffold extension* (elongating the scaffolds) are often needed to complete the genome.

The strategy we propose diverges from the ones usually described in the literature. While the latter tackle the different assembly stages one after another separately, our methodology consists in developing a global optimization approach where the scaffolding, gap-filling, and scaffold extension steps are simultaneously solved in the framework of a common objective function. We reduce these three steps to finding a long simple path[1] in a specific graph that satisfies as many as possible of the additional constraints encoding the insert-size information. Our approach is *repeat-aware* [16, 7]– the potential unitig repeats are taken into account by adding as many copies of repeated unitigs as needed. This requires the development of new tech-

[1] a path in which no vertex appears more than once

niques for searching of a simple path where the distances between some of the nodes are known. Furthermore, we propose a Mixed-Integer Linear Programming (MILP) formulation for the related optimization problem and solve it using a suitable solver.

We are not aware of previous approaches for genome assembly based on a similar formulation. Most previous work on scaffolding is heuristics based, e.g., SSPACE [2], GRASS [8], BESST [22] and SPAdes [1]. Such tools may find in some cases good solutions, but their accuracies cannot be guaranteed or predicted. Note that SPAdes uses an advanced abstraction of NGS data where the genome assembly is seen as the problem of reconstructing a string from a set of pairs of k-mers (also called *k-bimers*). A *k-bimers* is a triple $(a|b, d)$ consisting of k-mers a and b together with an integer d (estimated distance between particular instances of a and b in a genome). Moreover, SPAdes uses *A-Bruijn graphs* [20] where the operations are based on graph topology, coverage, and sequence lengths, but not on the sequences themselves. These concepts resemble to the main hypotheses in our model except that nodes denote for us unitigs, while edges stand for the overlaps between them. Furthermore, the formal language and the techniques we use (MILP) differ significantly from the ones in [1].

MILP approaches have already been used for targeting the scaffolding problem [19, 17, 23, 16]. These optimization approaches use various objective functions for maximizing the number/weight of links that are consistent with the paired-end/mate-pair reads. However, as far as we know, in contrast to our approach, none of them manages distances between unitigs, neither proposes sub-tour elimination strategies for searching a simple path in a graph.

Completely different direction of research is presented in [27, 26]. Focusing on the scaffolding problem, and towards the goal of designing exact algorithms for solving it, the authors study the structural properties of the contig graph that would make the corresponding problem of polynomial complexity.

Our paper focuses on circular genomes and, in particular, on chloroplasts. The reasons for this choice are as follows. Chloroplasts possess circular and relatively small genomes. The particularity of these genomes is the presence of numerous repetitions, which pose the main computational challenges for the modern genome assembly techniques. On the other hand, the size of the chloroplast genome permits assembling them rapidly (each one of the instances from the considered benchmark except one, Euglena-Gracilis genome, has been solved for less that 3 sec.) and so we were able to refine our strategy and to focus entirely on the quality of the obtained results.

The contributions of this study are as follows:

- We reduce the unitigs assembly to a problem of finding longest paths in specific graphs with an additional set of distance constraints between

some couples of vertices along these paths. We also propose a Mixed-Integer Linear Programming (MILP) formulation for this problem.

- Using the specificities of the circular genomes case we simplify significantly the sophisticated MILP model described in our previous paper [6].
- We deeply analyze the reasons for the existence of multiple optimal solutions. It is well known that this is mainly due to he presence of repeated unitigs. Our contribution here is to formulate two sufficient conditions for the existence of alternative subpaths that yield equivalent solutions. Furthermore, we design efficient (linear) algorithms for detecting these conditions. We thus provide formal foundations and improve the heuristics strategy described in our conference paper [5].
- Once these subpaths detected, our strategy consists in performing cuts at their endpoints. The remaining subpaths are considered as *unambiguous* portions of the genome (contigs). We call these contigs *distance-based contigs (db-contigs)* since our splits are safe (do not decrease the number of satisfied distances).
- We tested this strategy on a set of 33 chloroplast genomes and compared the results with three recent assemblers (SPAdes [1], SSPACE [2] and BESST [22]).
- Using the QUality ASsessment Tool (QUAST) [9] for quality assessment, we demonstrate that our approach produces assemblies of higher quality than the above heuristics.
- Moreover, we study the robustness of the approach in respect to the depth of sequencing coverage (read coverage). We show that our tool is able to find an acceptable assembly even for a small read coverage of 30X by choosing a suitable k-mer size. However, the results are significantly better for a read coverage above 70X.

2 Methodology

This section is organized as follows. First we give some basic definitions. Then, in subsection 2.2, we describe the graph modeling, i.e., transforming the input genomic data to a graph and reducing the assembly problem to a path search in this graph. In subsection 2.3, we present the mathematical programming formulation, which includes enhancements of the model that take into account the specifics of the circular genomes. While such enhancements make the model less general than the one in [6], they significantly increase its efficiency.

2.1 Definitions and notations

A *compacted de Bruijn graph* (CDBG) is a representation of a de Bruijn graph in which each non-branching path is merged into a single node. The sequence of each node in a CDBG is called an *unitig*[15]. It is easily seen that the following properties hold: i) two distinct unitigs do not share a k-mer; ii) any k-mer of a unitig is unique (not duplicated in the unitig).

In our approach, the unitigs are represented by nodes that keep track only of the length of the associated sequence, but not of the sequence itself. Moreover, any unitig is affiliated with its *copy-count*, defined as the number of times the unitig (or its reverse-complement) is repeated along the genome.

2.2 Modeling the assembly problem as a path search in a graph

The input data consists of the following types:

- A set $\mathcal{U}$ of *unitigs* characterized by their lengths and copy-counts. Only unitigs whose length is larger than a predefined threshold (cf section 4.1) are considered. Let us denote by w_u(resp. k_u) the length(resp. the copy-count) of the unitig u. The k_u values are computed during the unitig generation phase as explained in section 4.1.

- A list O of oriented pairs of unitigs, which we call *overlaps*. Two unitigs u and v overlap if a suffix of u is equal to a prefix of v. For each $(u, v) \in O$, the length of the overlap between them is encoded as a weight $l_{(u,v)}$ and is such that $l_{(u,v)} < 0$ and $|l_{(u,v)}| < \min(w_u, w_v)$.

- A list L of oriented pairs of unitigs, which we call *links*. Links are determined from the *paired-end* or *mate-pairs* information and encode distance information. Due to fluctuation and inaccuracy in the insert-size information, an interval $[\underline{d}_{(u,v)}, \overline{d}_{(u,v)}]$ is associated with each $(u, v) \in L$, rather than a single number, and it means that a solution candidate will be rewarded if the distance between u and v is contained in this interval. Hence, we say that the link (u, v) is *satisfied*, if the distance in the solution between u and v is a number in $[\underline{d}_{(u,v)}, \overline{d}_{(u,v)}]$.

An instance of such input data is given in Figure 1. Based on this input, we construct a directed graph $G = (V, E)$ called a *unitig graph*, where both the set V of vertices and the set E of edges are weighted. We generate the set V according to the following rules:

1. Each unitig i is represented by at least two vertices v_i and v'_i, where v'_i denotes the reverse-complement counterpart of v_i.

2. If unitig i is repeated k_i times, it generates a set of $2k_i$ vertices $v_{i1}, \ldots, v_{ik}$, $v'_{i1}, \ldots, v'_{ik}$.

Unitig and its copy-count

3_len_19212	1
5_len_7596	2
4_len_88398	1
2_len_18914	2

Overlaps

2_len_18914_R	3_len_19212_R	-124
2_len_18914_R	3_len_19212_F	-124
2_len_18914_F	5_len_7596_R	-124
3_len_19212_R	2_len_18914_F	-124
3_len_19212_F	2_len_18914_F	-124
4_len_88398_F	5_len_7596_F	-124
4_len_88398_R	5_len_7596_F	-124
5_len_7596_R	4_len_88398_F	-124
5_len_7596_R	4_len_88398_R	-124
5_len_7596_F	2_len_18914_R	-124

Distances

2_len_18914_F	5_len_7596_R	0	89
2_len_18914_R	3_len_19212_F	0	40
2_len_18914_R	3_len_19212_R	0	61
2_len_18914_F	4_len_88398_R	7267	7323
3_len_19212_R	2_len_18914_F	0	162
3_len_19212_F	2_len_18914_F	0	58
4_len_88398_F	5_len_7596_F	0	93
4_len_88398_R	5_len_7596_F	0	71
5_len_7596_R	4_len_88398_R	0	70
5_len_7596_F	2_len_18914_R	0	158
5_len_7596_R	4_len_88398_F	0	51

Figure 1: Input data for *Liriodendron Tulipifera* chloroplast. It contains three type of data : i) Set of unitigs with their copy-count number. The unitig i with length w_i is represented here as `[ i_len_w_i ]`. ; ii) List of overlaps between unitigs. The first two terms here correspond to a couple of unitigs, while the last one is the associated overlap. The orientation of a unitig (forward/ reverse-complement) is denoted by `(F, R)` respectively, and corresponds to the last character of these terms.; iii) List of distances between unitigs (also called links). As before, the first two terms correspond to a given couple of unitigs, while the last two data represent the lower and upper bound of the distance between them.

Denote $W = \sum_{i \in V} w_i$ and $N = \sum_{i \in \mathcal{U}} k_i$; hence $|V| = 2N$.

By construction, there are two kinds of edges: edges corresponding to the overlaps between unitigs and denoted by O, and edges corresponding to the distances and denoted by L. Hence $E = O \cup L$.

Moreover, for any edge in G, its forward/reverse-complement is also in G, i.e., if e_{ij} is an edge joining vertices v_i and v_j, then its reverse-complement counterpart $e_{j'i'}$ is also in G.

In our approach, a genome assembly corresponds to a simple path in G that satisfies the maximum number of distances and is of maximum length. The graph and the solution associated with the input from Figure 1 are given in Figure 2.

In the next section, we give a Mixed Integer Linear Programming formulation for the above graph optimization problem.

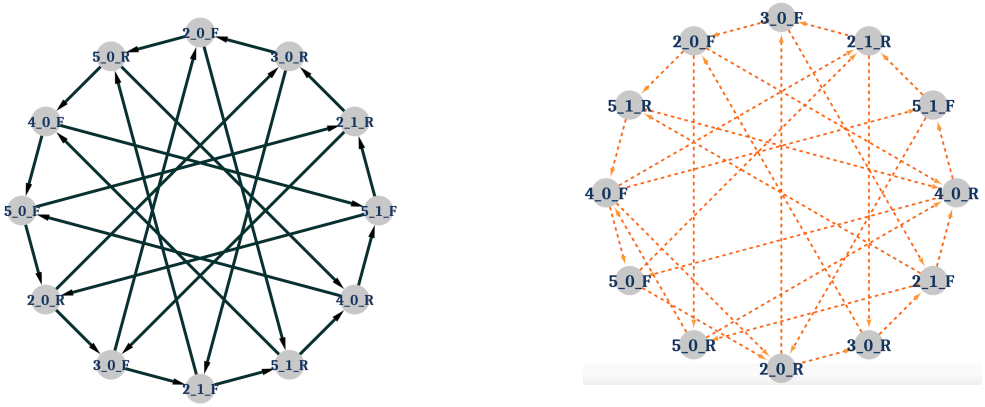

Figure 2: The unitig graph for *Liriodendron Tulipifera* chloroplast generated from the input data given in Fig. 1. **Left:** Since the initial set is composed of four vertices, and two of them have copy number two, the graph contains 12 vertices totally, according to the above explained formula. The forward (resp. reverse-complement) of the ith occurrence of the vertex v is denoted by `(v_i_F)` (resp. `(v_i_R)`). The directed edges visualize the overlaps. **Right:** The same graph where the set of links(distances to be satisfied) are represented by dashed directed edges.

2.3 A Mixed Integer Linear Programming Formulation

In order to define a path in a graph in an optimization problem, one needs typically to designate a start and end vertices for the path by appropriate variables. However, in our case, the beginning and the end of the path are unknown. Using extra decision variables in order to resolve this issue leads to the sophisticated model described in [6]. Here we are making use of the following facts/assumptions for chloroplast genomes that allows us to simplify the above general approach:

1. Chloroplast genomes are circular, implying that the beginning and the end of the path should coincide;

2. The largest unitig is part of the genome;

3. The entire genome is sufficiently covered (no gaps in its sequence).

Consistent with (1) and (2), we choose the largest unitig (say x) to play the role of the beginning and the end of the genome. Consequently, for the unitig graph, we introduce new vertices s and t to replace x, where s gets all outgoing edges of x and t gets all incoming edges from x. Specifically, we replace each edge (x, v) by an edge (s, v), each edge (v, x) by an edge (v, t) and define $\delta^+(t) = \delta^-(s) = \emptyset$, where $\delta^+(v) \subset E$ (resp. $\delta^-(v) \subset E$) denote the sets of edges outgoing from (resp. incoming to) v, while $\emptyset$ denotes the

empty set. Vertices s and t will be used as the source (start) and the sink (end), respectively, of the path we are looking for.

Furthermore, we associate a variable i_v with any vertex $v \in V \setminus \{s,t\}$ such that

$$0 \leq i_v \leq 1 \tag{1}$$

encoding whether v is or isn't in the solution path. Moreover, for each vertex v, we cannot visit both v and its reverse-complement counterpart v', which we encode as

$$\forall v : i_v + i_{v'} \leq 1. \tag{2}$$

We also associate a binary variable with each edge of the graph, i.e.,

$$\forall e \in O : x_e \in \{0,1\} \text{ and } \forall e \in L : g_e \in \{0,1\}. \tag{3}$$

If a vertex v not in $\{s,t\}$ is an intermediate vertex in the path, then there should be exactly one edge from $\delta^+(v)$ and one edge from $\delta^-(v)$ in the path, and if v is not in the path, then there will be no edges from $\delta^+(v)$ and no edges from $\delta^-(v)$ in the path. This is enforced by the following constraints

$$i_v = \sum_{e \in \delta^+(v)} x_e = \sum_{e \in \delta^-(v)} x_e. \tag{4}$$

It is then obvious that although the variables $i_v, \forall v \in V$ are defined in (1) as reals, they take binary values in each feasible solution.

The constraints so far help us to define a subgraph of G of degree two for all its vertices except s and t, which have degree one. But in order to be a path, such subgraph should also be connected. For that end, we model a flow in the graph and introduce a continuous variable $f_e \in R^+$ to express the quantity of the flow circulating along the edge $e \in O$. Without this variable, the solution may contain some loops and hence may not be a simple path. We put a requirement that no flow can use an edge e when $x_e = 0$, which can be encoded as

$$\forall e \in O : 0 \leq f_e \leq W x_e, \tag{5}$$

where W is as defined above ($W = \sum_{v \in V} w_v$).

We require that, when passing through a vertex v or an edge e of the path, the flow is reduced by w_v or f_e, respectively. Hence, the flow f_e should satisfy the following constraints: for any intermediate vertex we have

$$\forall v \in V \setminus \{s,t\} : \sum_{e \in \delta^-(v)} f_e - \sum_{e \in \delta^+(v)} f_e = i_v(w_v + \sum_{e \in \delta^-(v)} l_e x_e), \tag{6}$$

while for the source vertex s we require

$$\sum_{e \in \delta^+(s)} f_e = W. \tag{7}$$

We furthermore observe that the constraint (6) can be simplified. Using (4), the constraint (6) can be transformed into the equivalent simpler constraint

$$\forall v \in V \setminus \{s,t\} : \sum_{e \in \delta^-(v)} f_e - \sum_{e \in \delta^+(v)} f_e = i_v w_v + \sum_{e \in \delta^-(v)} l_e x_e. \tag{8}$$

More importantly, the constraint (8) is linear, unlike (6) which contains quadratic terms.

The model so far (constraints (1)–(8)) defines a path from s to t. But we also want such path to satisfy as many distance constraints as possible. Hence, we need also to add to the model information related to the links distances. For this end, we define a binary variable g_e for each edge e of G. For each $(u,v) \in L$, the value of $g_{(u,v)}$ should be 1 if and only if both vertices u and v belong to the selected path and the length $dist_{(u,v)}$ of the considered path between them is in the given interval $[\underline{d}_{(u,v)}, \overline{d}_{(u,v)}]$. This is accomplished by the following constraints:

$$g_{(u,v)} \leq i_u \text{ and } g_{(u,v)} \leq i_v \tag{9}$$

$$\forall (u,v) \in L : \sum_{e \in \delta^+(u)} f_e - \sum_{e \in \delta^-(v)} f_e + \sum_{e \in \delta^-(v)} l_e x_e \geq \underline{d}_{(u,v)} g_{(u,v)} - M(1 - g_{(u,v)}), \tag{10}$$

$$\forall (u,v) \in L : \sum_{e \in \delta^+(u)} f_e - \sum_{e \in \delta^-(v)} f_e + \sum_{e \in \delta^-(v)} l_e x_e \leq \overline{d}_{(u,v)} g_{(u,v)} + M(1 - g_{(u,v)}), \tag{11}$$

where M is some big constant (in our case we set $M = 100W$).

Our objective for the assembly problem is to find a long path in G such that as many as possible link distances are satisfied. The number of the satisfied link distances is $\sum_{e \in L} g_e$. Hence, the objective function of the optimization problem becomes

$$\text{maximize} \left(\sum_{e \in O} x_e l_e + \sum_{v \in V} w_v i_v + p \sum_{e \in L} g_e \right), \tag{12}$$

where p is a parameter to be chosen as appropriate (we use $p = W$).

3 Dealing with multiple optimal solutions

The information contained in the overlaps and the given set of distances is not always sufficient for identifying a unique assembly. The reasons for that are multiple–the unitig graph G is symmetric by construction, e.g., if there is an edge (v,w) between vertices v and w, then there is an edge (w',v') between their reverse-complements w' and v'. Such symmetry usually allows multiple optimal solutions. Moreover, the data contains repeated

identical unitigs, which are modeled by different vertices of G. As a consequence, for each optimal solution (path) p^* found by our algorithm, there are typically multiple (exponential in the worst case) number of equivalent solutions (paths). Such paths are different from p^* as sequences of vertices of G, but correspond to the same set of unitigs (and their reverse-complement copies) and satisfy the same number of links, and hence, are equally "optimal" from the point of view of the optimization problem (1)–(12). This issue is especially pronounced for chloroplasts due to their high number of repeated/symmetrical regions.

Choosing just any arbitrary path from the set of equivalent optimal ones can give an assembly different from the genome that serves as a reference for evaluating the accuracy of the obtained result. Our strategy to tackle this issue is as follows: i) find an optimal solution(path) p^*; ii) identify in p^* sources of multiple solutions (i.e. portions for which multiple equivalent subpaths exist), which we call *ambiguous* subpaths; iii) cut any of these ambiguous subpaths at the two interior edges adjacent to their endpoints (see formal definition below). The pieces remaining from the optimal path after such splits represent the contigs output by the algorithm. Their quality was assessed using QUAST [9] based on the following key metrics proposed by this tool (see QUAST 5.0.2 manuel `http://quast.bioinf.spbau.ru/manual.html` for more details) :

- **# contigs** – the total number of contigs in the assembly;
- **Genome fraction (%)** – the percentage of aligned bases in the reference genome. A base in the reference genome is aligned if there is at least one contig with at least one alignment to this base. Contigs from repetitive regions may map to multiple places, and thus may be counted multiple times;
- **#misassemblies** – the number of positions in the contigs (breakpoints) that satisfy one of the following criteria: i) the left flanking sequence aligns over 1 kbp away from the right flanking sequence on the reference; ii) flanking sequences overlap on more than 1 kbp;
- **#local misassemblies** – the number of positions in the contigs (breakpoints) that satisfy the following conditions: i) The gap or overlap between left and right flanking sequences is less than 1 kbp, and larger than the maximum indel length (85 bp); ii) The left and right flanking sequences both are on the same strand of the same chromosome of the reference genome.
- **Largest contig** – the length of the longest contig in the assembly.
- **Total length** – the total number of bases in the assembly.
- **Reference length** – the total number of bases in the reference genome.

QUAST requires as input a set of unitigs/contigs without indication for their repetition and orientation and maps any of them to the reference genome in order to assess its quality.

There are two main challenges in implementing the above strategy: i) how to detect splits; and ii) how to split the optimal path into subpaths without degrading its optimality (the number of satisfied distances). Details of our approach are presented below. Since we split safely in the sense that the number of satisfied distances in the set of remaining disconnected subpaths equals the number of satisfied distances in the original/optimal path, we call the contigs obtained in this way *distance-based contigs* (*db-contigs*).

Formally, we call two paths $p_1 = (v_1, \ldots, v_k)$ and $p_2 = (w_1, \ldots, w_k)$ of G *equivalent*, if they satisfy the same number of links and their vertices are permutations of the same set of unitigs (and their reverse-complement counterparts). Obviously, if p_1 and p_2 are equivalent, then $length(p_1) = length(p_2)$.

Given an optimal path p^*, we search in this section to detect pairs of vertices $(v, w) \in p^*$ that represent endpoints of ambiguous subpaths. More formally, denote by p_1 the subpath $\in p^*$ between v and w. We call the couple (v, w) *split* of p^*, if there exists an equivalent path to p_1 (say p_2) such that the path (v, p_2, w) is also optimal subpath of p^*.

Next, we describe two methods for identifying such splits. Note that both are related to repeated unitigs.

3.1 Link-closed reversible subpaths

We call a path p *link-closed*, if for any link that has as an endpoint a vertex of p, its other endpoint is also in p. Consider an unitig v_s with copy-count at least two. Amongst the link-closed paths, we are interested in those having as one endpoint such a unitig v_s and as another an occurrence of its reverse-complement v'_s. We show below that if such a link-closed path p is part of the optimal solution p^*, then the couple (v_s, v'_s) is a split of p^*. We will call then p *link-closed reversible subpath*.

Indeed, according to the graph-generation rules, at least four vertices are associated with v_s : two (v_{s0} and v_{s1}) in the forward orientation, and two others (v'_{s0} and v'_{s1}) in the reverse-complement orientation. Denote now by $p = (v_k, v_{k+1}, \ldots, v_{r-1}, v_r)$ the link-closed subpath where $v_k = v_{s0}$ and $v_r = v'_{s1}$. We then show that its reverse-complement subpath $p' = \text{inv}(p) = (v'_r, v'_{r-1}, \ldots, v'_k)$ is also part of an optimal solution (see for illustration on Figure 3 where the upper (lower) path corresponds respectively to p (p').)

Obviously, $length(p) = length(p')$. Since $v'_r = v_{s1} = v_{s0}$ and $v'_k = v'_{s0} = v'_{s1}$, the paths p and p' have identical vertices/unitigs as their endpoints. Furthermore, for each link (g, h) in p, its reverse-complement counterpart (h', g') is in p' and both links simultaneously satisfy (or do not satisfy) the

associated distance since the subpaths between g and h (respectively h' and g') contain the same vertices (or their reverse-complements). Hence, both paths satisfy the same number of links. The subsequences p and p' are then two alternative subpaths in the optimal solution, while the vertices v_{s0} and $v_{s1}^{'}$ represent splits in the optimal path. Cutting it in these points (i.e. eliminating the edges connecting these vertices with the subpaths p and p') does not modify the number of satisfied distances since p and p' are link-closed subpaths. It turns out that such type of optimal subpaths are quite common.

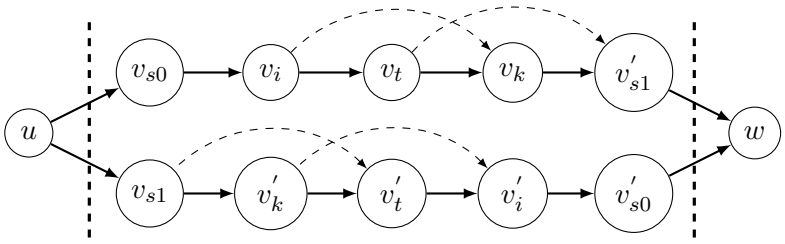

Figure 3: The copy-count number of v_{s0} equals 2. The upper subpath $p = (v_{s0}, v_i, \ldots, v_{s1}^{'})$ contains two links visualized with dashed lines. The lower path $p' = \text{inv}(p) = (v_{s1}, v'_k, \ldots, v'_{s0})$ is its reversible path. v_{s0} is identical to v_{s1}, while $v_{s0}^{'}$ is identical to $v_{s1}^{'}$. These two alternative subpaths are of the same length, satisfy the same number of links, and thus are equally optimal. The vertices v_{s0} and $v_{s1}^{'}$ limit this ambiguity zone. To manage such case, we split the optimal path at any of the interior edges adjacent to these endpoints (visualized by vertical dashed lines) and concatenate the unitigs (v_i, v_t, v_k) in one single contig. This new contig is a db-contig as it satisfies the link (v_i, v_k) while its reverse-complement satisfies the link $(v_k^{'}, v_i^{'})$. The position of this contig in the assembly is fixed (between v_{s0} and $v_{s1}^{'}$), but the given distances do not allow to determine its orientation in a unique way. Trying to fix it may result in an elimination of some eligible optimal solution.

Implementation details: Given an optimal path p^*, let us denote $L^* = \{(u, w) \in L : g(u, w) = 1\}$ (i.e. only the distances satisfied in p^* are considered). We say that a vertex $v \in p^*$ is *covered* if there exists a couple $(u, w) \in L^*$ such that v is an intermediate vertex in the subpath between u and w (i.e. $v \neq u$ and $v \neq w$). Otherwise, v is considered *uncovered*. Furthermore, let $\text{pos}(v)$ be the position in p^* of the vertex v. This position can be assigned to any $v \in p^*$ simply by traversing linearly p^* once the optimal path has been found (i.e. in $O(|V|)$ time).

Let us denote by $\tilde{R}_v$ the set of all reversed-complements of all occurrences of v belonging to p^* and by LC the set of endpoints of all link-closed reversible subpaths. The set LC can be generated by the following simple algorithm.

Algorithm 1 Link-closed reversible subpaths generation

1: $LC = \emptyset$
2: To any vertex $v \in p^*$ associate its position $\text{pos}(v)$ in p^*
3: Compute the set of all uncovered vertices in p^* .
4: **for all** $v \in p^*$ in increasing order of $\text{pos}(v)$ **do**
5: **if** v is uncovered and is a repeated vertex **then**
6: **for all** uncovered $u \in \tilde{R}_v$ s.t. $\text{pos}(u) > \text{pos}(v)$ **do**
7: add (v, u) to LC;
8: **end for**
9: **end if**
10: **end for**

The most costly operation in the above algorithm is line 3 (computing the set of uncovered vertices in p^*). In order to implement that efficiently, we sort the set L^* in increasing order of the position in p^* of the beginning of the links (which can be done in $O(|L^*| \log |L^*|)$).

We then traverse in increasing order of the vertices positions the path p^*, as well as the sorted list L^*. Comparing the position of any vertex $v \in p^*$ with the beginning and the end of the closest link we detect if v is covered or uncovered. This algorithm can be done in linear time $O(|V| + |L^*|)$. The total complexity of algorithm 1 is then $O(|V| + |L| \log |L|)$.

3.2 Reversible symmetry splits

Consider vertex $v \in p^*$ and denote by $prev(v)$ (resp. $next(v)$) its previous (resp. next) neighbor on the path p^*. We call v *reversible* if $(prev(v), v') \in O$ and $(v', next(v)) \in O$.

Let us denote

$$\Gamma_v^- = \{u \in p^* : g(u, v) = 1\} \text{ and } \Gamma_v^+ = \{w \in p^* : g(v, w) = 1\}. \tag{13}$$

We call $v \in p^*$ *symmetrical reversible* if the following conditions are satisfied: i) v is reversible; ii) $|\Gamma_v^-| = |\Gamma_v^+|$, and, the sets Γ_v^- and Γ_v^+ are composed of occurrences of vertices with repetitions (i.e. with copy-counts at least 2), and such that for any $u \in \Gamma_v^-$, (resp. Γ_v^+) $\exists w \in \Gamma_v^+$ (resp. Γ_v^-) such that $w \in \tilde{R}_v$ and $dist(u, v) = dist(v, w)$. It is easy to see then that the vertices v and v' are interchangeable (i.e. they are two alternative subpaths in the optimal path p^*). The vertices $prev(v)$ and $next(v)$ are the associated splits, while the vertex v will be disconnected from them and

will be considered as a separate contig in our approach (see Figure 4 for an illustration).

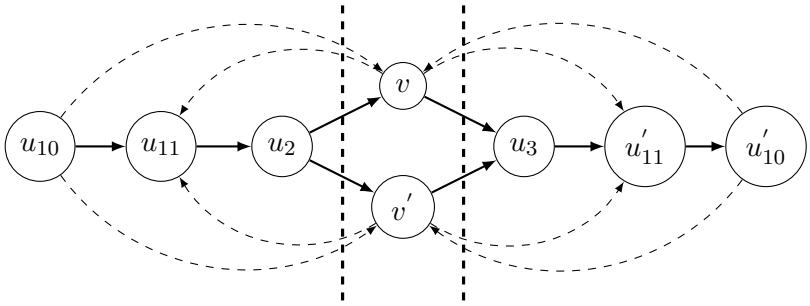

Figure 4: In this example $\Gamma_v^- = \Gamma_{v'}^- = \{u_{10}, u_{11}\}$, $\Gamma_v^+ = \Gamma_{v'}^+ = \{u'_{10}, u'_{11}\}$. Moreover, $dist(u_{10}, v) = dist(v, u'_{10})$ and $dist(u_{11}, v) = dist(v, u'_{11})$. The vertex v is then symmetrical reversible and the vertices v and v' are two alternative subpaths in the optimal solution. The position of the vertex v in the computed assembly is fixed (between the vertices u_2 and u_3), but its orientation cannot be determined in a unique way. Both orientations of this vertex are equally optimal. To manage this situation we cut the path at the edges adjacent to v and v'.

The algorithm for generating the list of reversible symmetry splits simply consists in verifying the above conditions for any reversible link endpoint $v \in p^*$. This reduces to a search operation in the sets Γ_v^-, Γ_v^+, which can be done in $O(|L| \log |L|)$ time complexity.

Both above splitting conditions are illustrated on Figure 5 for the case of Liriodendron Tulipifera chloroplast. Note that the partition obtained here by our approach corroborates with the common knowledge that many chloroplast DNAs contain two inverted repeats (here contigs 2_0_R and 2_1_F), that separate a long single copy section (LSC) (here 5_1_R $\rightarrow$ 4_0_F $\rightarrow$ 5_0_F) from a short single copy section (SSC) (here 3_0_F) [13].

In the next section we report some experimental results comparing our tool with some of the best existing scaffolding tools. All the results have been obtained by applying the two above described splitting criteria.

4 Numerical Results

4.1 Data Generation

Using 33 chloroplast reference genomes (see the first three columns in tables 1 and 2) obtained from the NCBI website (https://www.ncbi.nlm.nih.gov/genome),

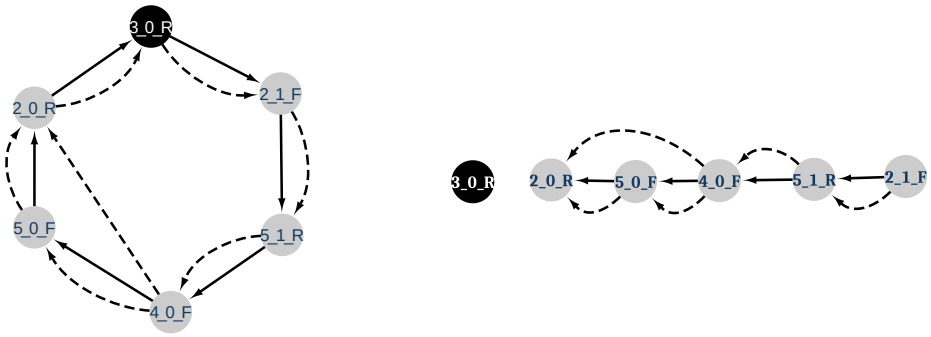

Figure 5: **Left:** The optimal solution found by the model for Liriodendron Tulipifera chloroplast (see the input data in Fig. 1) in which 7 (out of 28) distances are satisfied. They are visualized here by dashed edges, while normal directed edges represent overlaps. We observe that, for this particular instance, both splitting conditions are applicable. More precisely : on one hand, we detect that $2\_1\_F \rightarrow 5\_1\_R \rightarrow 4\_0\_F \rightarrow 5\_0\_F \rightarrow 2\_0\_R$ is a link-closed reversible subpath. Hence, we concatenate these five unitigs in a new db-contig and cut it from the path. On the other hand, we notice that the unitig 3_0_R is symmetrical reversible and can be chopped off. These splits coincide with the previously detected link-closed reversible subpath splits. **Right:** The two contigs of the final assembly. QUAST completely approves the result. The excerpt of its assessment states : Genome size 159886; Genome fraction (%) 100 ; # misassemblies 0; # local misassemblies 0; # contigs 2; N50 140922.

we have generated synthetic sequencing reads using ART simulator Illumina [11]. In order to evaluate the paired-end and the mate-pair technology behavior, two data sets have been produced respectively for each chloroplast. The coverage has been set to 100X, the read size to 250bp and the insert size to 600bp (paired-end) and 8Kbp (mate-pair). For each genome, the following three tasks have been performed: (1) generation of unitigs; (2) overlap computation; (3) link computation.

Unitigs generation

The set of unitigs was generated with the `Minia` assembler [4], while the best k-mer size was chosen based on the tool `KmerGenie` [3]. An estimation of the copy-count of each unitig is also returned by `Minia` following the strategy described next.

The *copy-count* of a k-mer is defined as the number of times the k-mer (or its reverse-complement) is repeated along the genome, while the *abundance* of a k-mer is defined as the number of times it (or its reverse-complement) appears in the multiset of k-mers [3]. The abundance of any k-mer is directly counted by `Minia`. Since, by definition, any k-mer is uniquely associated with a unitig, the copy-count (resp. the abundance) of any unitig equals the copy-count (resp. the abundance) of any of its k-mers. However, this is the ideal case. Since in practice the reads are not uniformly distributed, we compute the abundance of a unitig as the average of the abundances of all k-mers associated with it.

On the other hand, the *abundance* of a non-repeated unitig is theoretically equal to the average depth of sequencing coverage (`dsc`) which, by definition, equals $\frac{lr \times nr}{gl}$, where lr is the read length, nr is the number of reads, and gl is the genome length. A unitig that is repeated M times along the genome (which is it copy-count value) has an abundance of $M \times$`dsc`. Here, we assume the longest unitig is not duplicated, and its abundance should be then equal to the `dsc`. Based on this assumption, `Minia` computes the copy-count of each unitig as its abundance divided by the longest unitig's abundance, rounded to the nearest upper integer value.

This strategy provides an estimation of the copy-count, but its accuracy strongly depends of the length of the unitigs. The longer the unitigs, the better the estimation is. Actually, for very short unitigs, we can only provide an upper bound for the copy-count value.

Overlap computation

Let $(u, v) \in V \times V \setminus \{u\}$ be a couple of unitigs. We say that u and v overlap, if a suffix of u is equal to a prefix of v. More formally, denote by length(u) the length of u. For a given integer $k \geq 2$ (which corresponds to the k-mer size), we search for the biggest integer $ov \geq 1$ such that:

- $\lfloor k/2 \rfloor \leq ov \leq k-1$
- $\forall i = 0, \ldots, ov,\ u[\text{length}(u) - ov + i] = v[i]$ (the last ov characters of the sequence u are identical to the first ov character of the sequence v)

If such an integer exists, we consider there exists an overlap between u and v of length ov .

Link computation

Each mate-pair or paired-end read is individually mapped to the unitigs with the *Minimap* mapper tool [14]. Reads that map ambiguously to several locations are discarded. To minimize false positives, only links that are validated by at least 5 pairs are kept. The link size is estimated with the given inserts size value, and averaged over all pairs that confirm the link.

The first six columns of tables 1 and 2 summarize the set of chloroplasts used here together with their associated features corresponding to the mate-pair case. Figure 1 is an example of input data (unitigs, overlaps, links) generated from the *Liriodendron Tulipifera* chloroplast in the mate-pair.

4.2 Comparison with other tools

The optimization model has been implemented using the AMPL (A Modeling Language for Mathematical Programming) language and the Gurobi solver (version 7.0) [10]. The computational results were obtained on a Laptop Intel(R) Core(TM) i7-6700HQ (2.6 GHz, 16 GB RAM) running Linux Mint. 18.3.

This section compares the results of the tool we have developed (denoted here by GAT: GenScale Assembly Tool) with state-of-the-art assembly tools: Spades[1], SSPACE [2], BESST [22]. The assemblies were evaluated by QUAST by comparison with the reference genome that was used for the simulation. We used here the following four major metrics proposed by QUAST : **# contigs**, **genome fraction (%)** , **#misassemblies** and **#local misassemblies**.

Figures 6 and 7 synthesize the first two metrics in the case of mate-pair and paired-end evaluation respectively. On the left side on these figures the average values of the assembled contigs are plotted. The lower this number, the better the assembly. On the right side the average fractions of the genome left out (i.e. 100-genome fraction %) are reported. Again, the lower the fraction, the better the assembly. In both cases (mate-pair and paired-end), GAT clearly provides the best results in respect to these two measures.

Tables 3 and 4 give the exact values for any of the mate pairs and paired-end instance. In the case of mate pairs comparison, we observe that

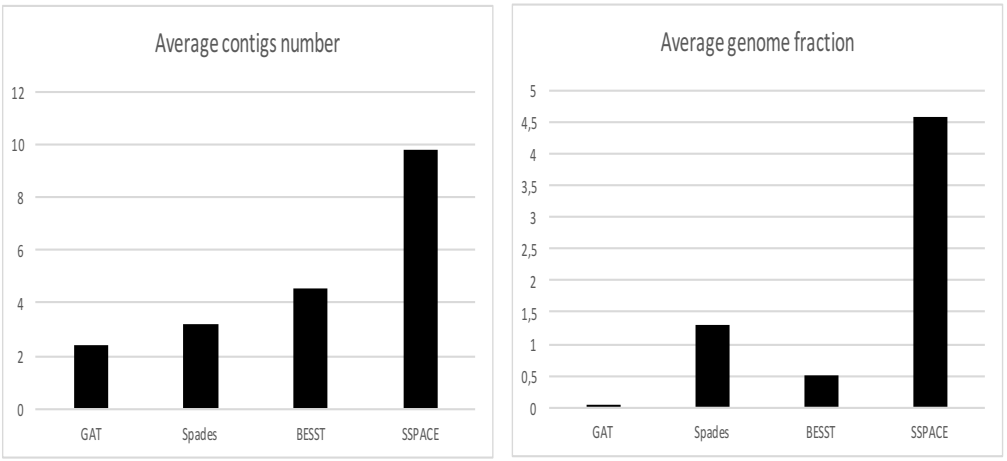

Figure 6: Mate-pair data: **Left:** Average number of contigs comparison. **Right:** Average fraction of genome left-out comparison.

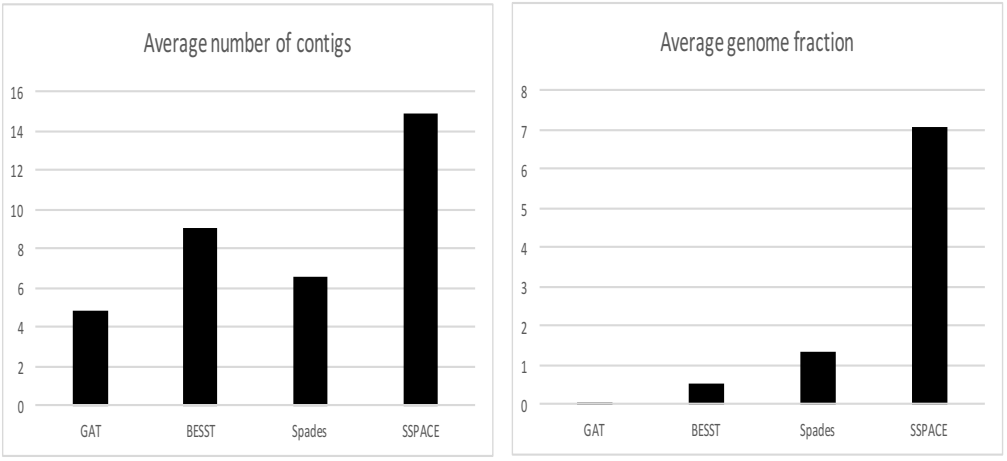

Figure 7: paired-end data: **Left:** Average number of contigs comparison. **Right:** Average fraction of genome left out comparison.

GAT ensures, with one exception, the ideal coverage of the reference genome (second column in GAT results). The unique exception here concerns the Euglena Gracilis instance (No. 21). This appears to be the most difficult instance in the benchmark–its input data contains one big unitig that covers 82% of the genome, and 64 small unitigs with a lot of repetitions. Note that BESST was unable to solve it.

Similar is the situation with the number of contigs, where GAT obtains, with one exception, the smallest number of contigs. This exception concerns the Atropa genome, where SPAdes obtains 3 contigs (versus 5 obtained by GAT). However, we observed that the $N50$ value[2] obtained by SPAdes for this instance is worser than the one given by GAT. In total, SSPACE failed to assemble 3 genomes out of 33 (see the corresponding **#misassemblies** column–the last one in Table 3), BESST–one genome (the above mentioned Euglena Gracilis instance). SPAdes and GAT assembled all 33 instances, but SPAdes performed two **#local misassemblies** errors, while GAT is the only tool without any kind if misassemblies (cf. Table 3).

The paired-end comparison is equally in favor of our tool. The obtained results for all benchmark instances are reported in table 4.

4.3 Robustness

The method described in this paper specifically targets circular genomes and requires that all the regions of the genome are covered by unitigs. Otherwise, the corresponding graph is decomposed into connected components and obviously a circular path does not exist. If this is the case, the model described in [6] should be used.

The number of non-covered regions is directly linked to the sequencing depth (read coverage). The lower the read coverage is, the bigger the number of non-covered regions is. Usually, a coverage of at least 30X is required to expect a "good assembly" (without non-covered regions).

On the other hand, low coverage can be compensated by reducing the k-mer length. Finding the best k-mer length is not an easy task. Tools like `KmerGenie` aim to analyze the sequencing data and to estimate the "best" k-mer size [3]. In the experiments reported in the table below, we used `KmerGenie` in order to find the right k-mer length as a function of the read coverage value.

Table 5 illustrates how finding a convenient k-mer length allows the model to find a solution even in the case of low read coverage. The best k-mer length is proportional to the value of the read coverage: the smaller the read coverage value is, the smaller the best k-mer length should be. As we observe on the table, when the read coverage is too small (10X), the conditions for the validity of the model are not satisfied, and GAT does not

[2] another QUAST metric

No.	Genomes	Size	$\|V\|$	$\|O\|$	$\|L\|$	NSL	`Running time`
1	Acorus Calamus	153821	8	16	16	3	1.406s
2	AdiantumCapillus Veneris	150568	20	24	24	5	1.138s
3	Agrostis Stolonifera	136584	20	52	24	6	1.284s
4	Angiopteris Evecta	153901	34	78	70	12	2.538s
5	Anthoceros Formosae	161162	16	32	24	5	1.209s
6	Arabidopsis Thaliana	161162	20	40	32	7	1.187s
7	Arabishirsuta	153689	12	24	24	5	1.163s
8	Atropa	156687	46	90	34	9	1.272s
9	Capsella Bursa Pastoris	154490	12	24	24	5	1.193s
10	Chaetosphaeridium Globosum	131183	8	16	16	3	1.257s
11	Chara Vulgaris	184933	24	56	24	7	1.297s
12	Chlorella Vulgaris	150613	52	50	50	24	1.274s
13	Chlorokybus Atmophyticus	152229	10	18	18	4	1.262s
14	Citrus Sinensis	160129	12	24	32	8	1.274s
15	Cyanidioschyzon Merolae	149067	72	82	46	22	1.209s
16	Cyanidium Caldarium	164921	38	36	32	15	1.232s

Table 1: The benchmark containing 33 chloroplast genomes whose names are given in the first column. The second column contains their lengths. We observed this value equals the value given by the first term of the objective function (12). The next three columns describe the input graph for the mate-pairs case. The third and fourth columns give the number of vertices and edges of the graph, respectively, while $|L|$ indicates the number of the links. The last two columns are relevant to the solution found by our tool. NSL stands here for number of satisfied links, while `Running time` (in seconds) indicates the execution time.

No.	Genomes	Size	$\|V\|$	$\|O\|$	$\|L\|$	NSL	`Running time`
17	Daucus Carota	155911	8	16	16	3	1.165s
18	Draba Nemorosa	153289	12	24	24	5	1.220s
19	Eimeria Tenella	160604	10	18	18	4	1.178s
20	Epifagus Virginiana	70028	12	24	24	5	1.177s
21	Euglena Gracilis	143171	146	554	30	11	17.932s
22	Gossypium Barbadense	160317	12	24	24	5	1.175s
23	Gossypium Hirsutum	160301	14	28	24	5	1.199s
24	Gracilaria Tenuistipitata	183883	54	54	44	21	1.210s
25	Guillardia Theta	121524	44	88	24	5	1.242s
26	Helianthus Annuus	151104	10	18	18	4	1.270s
27	Huperzia Lucidula	154259	20	48	20	5	1.294s
28	Lactuca Sativa	152765	8	16	16	3	1.223s
29	Lepidium Virginicum	154743	24	48	48	11	1.167s
30	Liriodendron Tulipifera	159886	8	16	16	3	1.203s
31	Lobularia Maritima	152659	16	32	32	7	1.216s
32	Lotus Corniculatus	150519	20	80	32	7	1.329s
33	Pinus	116864	58	128	12	6	1.156s

Table 2: The last 17 chloroplasts of our benchmark. Euglena Gracilis (No. 21) is the most difficult instance of this set–its input data contains one big unitig, and a huge number of small and highly repetitive unitigs.

	GAT				BESST				SPAdes				SSPACE			
1	2	100	0	0	3	83,2	0	0	3	83	0	0	2	100	0	0
2	2	100	0	0	3	100	0	0	3	100	0	0	3	100	0	0
3	2	100	0	0	3	100	2	0	4	99,5	0	0	6	84,6	0	0
4	3	100	0	0	7	100	2	0	6	99,3	0	0	7	100	0	0
5	2	100	0	0	4	100	2	0	3	100	0	0	7	90,4	0	1
6	2	100	0	0	4	100	2	0	3	100	0	0	5	100	0	0
7	2	100	0	0	3	100	0	0	3	100	0	0	3	82,9	0	0
8	5	100	0	0	9	100	4	0	3	100	0	0	18	100	0	0
9	2	100	0	0	4	100	0	0	3	100	0	0	4	100	0	0
10	2	100	0	0	3	100	0	0	3	100	0	0	2	90,6	0	0
11	3	100	0	0	4	100	2	0	5	100	0	0	6	94,2	0	0
12	1	100	0	0	8	100	6	0	1	100	1	0	34	100	0	0
13	2	100	0	0	3	100	0	0	3	100	0	0	1	100	0	1
14	2	100	0	0	2	100	2	0	3	100	0	0	3	100	0	0
15	2	100	0	0	2	100	2	0	2	100	0	0	5	100	0	0
16	1	100	0	0	7	100	8	0	1	99,7	0	0	40	100	0	0
17	2	100	0	0	3	100	0	0	3	100	0	0	3	100	0	0
18	2	100	0	0	3	100	0	0	3	100	0	0	3	100	0	0
19	2	100	0	0	3	100	0	0	3	100	0	0	2	83,6	0	1
20	2	100	0	0	4	100	0	0	3	100	0	0	8	100	0	0
21	16	99,6	0	0	-	-	-	-	9	92,8	0	0	30	100	0	1
22	2	100	0	0	3	100	0	0	3	100	0	0	6	84,4	0	0
23	1	100	0	0	10	100	7	0	1	100	1	0	51	100	0	0
24	1	100	0	0	13	100	0	0	3	100	0	0	14	100	0	0
25	2	100	0	0	4	100	0	0	3	100	0	0	5	100	0	0
26	2	100	0	0	6	100	0	0	4	100	0	0	6	90,5	0	0
27	2	100	0	0	8	100	6	0	4	82,4	0	0	24	100	0	0
28	2	100	0	0	2	100	0	0	3	100	0	0	6	90,4	0	0
29	2	100	0	0	4	100	0	0	3	100	0	0	5	83,5	0	0
30	2	100	0	0	2	100	2	0	3	100	0	0	4	100	0	1
31	2	100	0	0	2	100	2	0	3	100	0	0	2	82,9	0	0
32	2	100	0	0	8	100	0	0	5	100	0	0	9	86,3	0	0
33	1	100	0	0	3	100	4	0	2	99,8	0	0	6	100	1	0

Table 3: Detailed results concerning any of the 33 mate pair instances. Four columns are associated to any of the four tools subject of comparison (GAT, BESST, SPAdes, SSPACE). The first coluln corresponds to the **#contigs** metric, the second one–to the **Genome fraction (%)**, the third one–to the **#local misassemblies**, while the fourth one–to the **#misassemblies** metric.

	GAT				BESST				SPAdes				SSPACE			
1	2	100	0	0	3	83,2	0	0	10	84,2	0	0	3	83,1	0	0
2	2	100	0	0	4	100	0	0	3	100	0	0	4	100	0	1
3	5	98,5	0	0	5	98,4	0	0	8	98,5	0	0	6	84,4	0	0
4	8	100	0	0	11	100	0	0	16	89	0	0	11	99,5	0	0
5	4	100	0	0	6	100	0	0	4	100	0	0	6	93,9	0	0
6	4	100	0	0	7	100	0	0	3	100	0	0	5	100	0	0
7	2	100	0	0	4	100	0	0	3	100	0	0	5	100	0	0
8	14	99,9	0	0	14	100	0	0	3	100	0	0	19	100	0	0
9	2	100	0	0	4	100	0	0	3	100	0	0	3	83	0	0
10	2	100	0	0	3	100	0	0	5	100	0	0	3	100	0	0
11	8	100	0	0	8	100	0	0	23	100	0	0	9	94,1	0	1
12	1	100	0	0	25	100	0	0	19	100	0	0	57	100	0	0
13	3	100	0	0	4	100	0	0	3	100	0	0	3	95,1	0	0
14	2	100	0	0	4	100	0	0	3	100	0	0	2	83,3	0	0
15	3	99,4	0	0	29	100	2	0	6	100	0	0	80	100	0	0
16	1	100	0	0	18	100	0	0	1	100	0	0	55	100	0	0
17	2	100	0	0	3	100	0	0	3	100	0	0	3	100	0	0
18	2	100	0	0	4	100	0	0	3	100	0	0	3	82,9	0	1
19	2	100	0	0	4	100	0	0	4	100	0	0	4	83,5	0	0
20	2	100	0	0	4	100	0	0	3	100	0	0	2	68	0	0
21	29	100	0	0	32	100	0	0	13	98,3	0	0	40	95,8	0	0
22	2	100	0	0	4	100	0	0	3	100	0	0	4	100	0	0
23	2	100	0	0	4	100	0	0	4	100	0	0	5	100	0	0
24	2	100	0	0	24	100	0	0	1	100	0	0	72	100	0	0
25	15	100	0	0	14	100	0	0	8	100	0	0	16	98,2	0	0
26	2	100	0	0	4	100	0	0	3	100	0	0	4	83,8	0	1
27	2	100	0	0	7	100	0	0	18	100	0	0	10	94,9	0	0
28	2	100	0	0	3	100	0	0	3	83,7	0	0	3	100	0	0
29	2	100	0	0	7	100	0	0	3	100	0	0	4	100	0	1
30	2	100	0	0	3	100	0	0	3	100	0	0	7	100	0	0
31	2	100	0	0	5	100	0	0	3	100	0	0	6	83	0	0
32	2	100	0	0	6	100	0	0	3	100	0	0	4	86,3	0	0
33	12	100	0	0	12	100	0	0	8	100	0	0	17	100	0	0

Table 4: Detailed results concerning any of the 33 paired-end instances. Four columns are associated to any of the four tools subject of comparison (GAT, BESST, SPAdes, SSPACE). The first column corresponds to the **#contigs** metric, the second one–to the **Genome fraction (%)**, the third one–to the **#local misassemblies**, while the fourth one–to the **#misassemblies** metric.

Reads Coverage	K-mer Size	#Unitigs MINIA	#Contigs GAT	genome fraction MINIA	genome fraction GAT	Misassemblies GAT
10X	31	160	NAN	98	NAN	NAN
30X	47	86	37	100	100	0
50X	121	36	10	100	100	0
70X	121	18	4	100	100	0
90X	121	23	4	100	100	0
100X	121	23	4	100	100	0

Table 5: Acorus Calamus genome: Illustration of the robustness of our approach as a function of the reads coverage (sequencing depth) given in the first column. The second column contains the value of the k-mers used by MINIA for unitigs generation. The number of the unitigs is given in the third column, while the fourth column contains the number of contigs assembled by GAT. The fifth(resp. sixth) column reports the genome fraction covered by the unitigs(resp. contigs) generated by MINIA(resp. GAT). The last column gives the number of misassembled contigs generated by GAT. When the read coverage is insufficient (less than 30X), GAT does not find any solution (indicated by `NAN` in the table). Otherwise, no misassemblies in the produced contigs are observed.

return a result. For slightly bigger value of the read coverage (30X), GAT succeeds in finding an assembly by choosing a k-mer of size 47. However, the solution is relatively fragmented–37 contigs. The results improve more and more above 50X sequencing depth, and attend the best assembly of 4 contigs above 70X.

5 Conclusion

It is commonly accepted that the last two stages in genome assembly are scaffolding and gap filling. Since both problems have been proven to be NP-hard, people usually propose heuristics as their solution. While we completely support this attitude in the case of huge genome instances, since mostly exact approaches do not scale well, we believe that the bioinformatics community should not neglect the design of exact techniques for tackling computationally hard problems. In case of manageable data size instances, the exact approaches normally produce better quality results than the competing heuristics and thus provide a precious information for the associated genomes. The advantages of exactly solving the gap-filling problem by dynamic programming on a bacterial data sets of moderate size has been recently shown in [24].

Here we continue in the same direction of research and propose an exact

approach for solving the scaffolding and gap filling phases in the particular case of circular genomes. We merge both theses tasks and model them as a graph optimization problem asking for the longest path in a graph with additional distance constraints encoding the insert-size information. It works both in case of mate-pairs and paired-ends distances. Furthermore, we propose a Mixed Integer Linear Programming formulation for the above graph problem and use the AMPL language to code it and to solve it by an optimization solver.

On a benchmark of 33 chloroplast genomes our tool significantly outperforms three recent heuristics with respect to the quality of the results. Moreover, our running times are acceptable–the longest instance took less than 18 seconds (see tables 1 and 2). These results fully justify the efforts for designing exact approaches for genome assembly. Extending the method to much bigger genomes is very challenging and a topic of ongoing research. We are currently implementing advanced combinatorial optimization techniques to increase the scalability of the approach without sacrificing the accuracy of the results.

Acknowledgments

We would like to thank Guillaume Rizk for adapting the Minia assembler [4] for the purpose of our algorithm and for his help in data generation. Many thanks to Rayan Chikhi for valuable discussions.

This work has been supported in part by Inria international program Hipcogen and by the Laboratory Directed Research and Development program of Los Alamos National Laboratory under project number 20160317ER.

Data availability

The input data that served to generate the graphs associated to any of the 33 instances referenced here (both for the mate-pair and the paired-end case) is available at `https://data-access.cesgo.org/index.php/s/e7JV2rXOsmPCbb9`.

References

[1] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, Alexey V Pyshkin, Alexander V Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A Alekseyev, and Pavel A Pevzner. Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology : a journal of computational molecular cell biology*, 19(5):455—477, May 2012.

[2] Marten Boetzer, Christiaan V. Henkel, Hans J. Jansen, Derek Butler, and Walter Pirovano. Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics (Oxford, England)*, 27(4):578–579, February 2011.

[3] Rayan Chikhi and Paul Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 2014.

[4] Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de bruijn graph representation based on a bloom filter. *Algorithms for Molecular Biology*, 8(1):22, 2013.

[5] Sebastien François, Rumen Andonov, Dominique Lavenier, and Hristo Djidjev. Global optimization approach for circular and chloroplast genome assemblies. In *10th International Conference on Bioinformatics and Computational Biology (BICoB 2018)*, pages 17 – 24, 2018. Mars, 2018, Las Vegas, USA.

[6] Sebastien François, Rumen Andonov, Dominique Lavenier, and Hristo Djidjev. Global optimization for scaffolding and completing genome assemblies. *Electronic Notes in Discrete Mathematics*, 64:185–194, 2018.

[7] Song Gao, Denis Bertrand, Burton K. H. Chia, and Niranjan Nagarajan. Opera-lg: efficient and exact scaffolding of large, repeat-rich eukaryotic genomes with performance guarantees. *Genome Biology*, 17(1):102, May 2016.

[8] Alexey A. Gritsenko, Jurgen F. Nijkamp, Marcel J.T. Reinders, and Dick de Ridder. GRASS: a generic algorithm for scaffolding next-generation sequencing assemblies. *Bioinformatics*, 28(11):1429–1437, 2012.

[9] Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. Quast: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.

[10] Inc. Gurobi Optimization. Gurobi optimizer reference manual. `http://www.gurobi.com`, 2010.

[11] Weichun Huang, Leping Li, Jason R Myers, and Gabor T Marth. Art: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2011.

[12] Daniel H. Huson, Knut Reinert, and Eugene W. Myers. The greedy path-merging algorithm for contig scaffolding. *J. ACM*, 49(5):603–615, 2002.

[13] Shaw Joey, B. Lickey Edgar, E. Schilling Edward, and L. Small Randall. Comparison of whole chloroplast genome sequences to choose noncoding

regions for phylogenetic studies in angiosperms: the tortoise and the hare iii. *American Journal of Botany*, 94,(3):275–288, March 2007.

[14] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.

[15] Antoine Limasset, Bastien Cazaux, Eric Rivals, and Pierre Peterlongo. Read mapping on de bruijn graphs. *BMC Bioinformatics*, 17(1):237, Jun 2016.

[16] Igor Mandric, Sergey Knyazev, and Alex Zelikovsky. Repeat-aware evaluation of scaffolding tools. *Bioinformatics*, 34(15):2530–2537, 03 2018.

[17] Igor Mandric and Alex Zelikovsky. ScaffMatch: scaffolding algorithm based on maximum weight matching. *Bioinformatics*, 31(16):2632–2638, 04 2015.

[18] Paul Medvedev, Son Pham, Mark Chaisson, Glenn Tesler, and Pavel Pevzner. Paired de Bruijn graphs: A novel approach for incorporating mate pair information into genome assemblers. *Journal of Computational Biology*, 18(11):1625–1634, 11 2011.

[19] Briot Nicolas, Chateau Annie, Rémi Coletta, Simon de Givry, Philippe Leleux, and Schiex Thomas. An integer linear programming approach for genome scaffolding. In *Workshop on Constraint based Methods for Bioinformatics*, 2015.

[20] Pavel A Pevzner, Haixu Tang, and Glenn Tesler. De novo repeat classification and fragment assembly. *Genome research*, 14(9):1786–1796, 09 2004.

[21] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *PNAS*, 98(17):9748–9753, 2001.

[22] Kristoffer Sahlin, Francesco Vezzi, Björn Nystedt, Joakim Lundeberg, and Lars Arvestad. BESST - efficient scaffolding of large fragmented assemblies. *BMC Bioinformatics*, 15:281, 2014.

[23] Leena Salmela, Veli Mäkinen, Niko Välimäki, Johannes Ylinen, and Esko Ukkonen. Fast scaffolding with small independent mixed integer programs. *Bioinformatics*, 27(23):3259–3265, 2011.

[24] Leena Salmela, Kristoffer Sahlin, Veli Mäkinen, and Alexandru I Tomescu. Gap filling as exact path length problem. *Journal of computational biology : a journal of computational molecular cell biology*, 23(5):347–61, 2016.

[25] James L. Weber and Eugene W. Myers. Human whole-genome shotgun sequencing. *Genome Research*, 7(5):401–409, 1997.

[26] Mathias Weller, Annie Chateau, Clément Dallard, and Rodolphe Giroudeau. Scaffolding problems revisited: Complexity, approximation and fixed parameter tractable algorithms, and some special cases. *Algorithmica*, 80(6):1771–1803, Jun 2018.

[27] Mathias Weller, Annie Chateau, and Rodolphe Giroudeau. Exact approaches for scaffolding. *BMC bioinformatics*, 16(Suppl 14):S2, 2015.