An integer linear programming approach for genome scaffolding
Nicolas Briot, Annie Chateau, Remi Coletta, Simon de Givry, Philippe Leleux, Thomas Schiex

To cite this version:
Nicolas Briot, Annie Chateau, Remi Coletta, Simon de Givry, Philippe Leleux, et al.. An integer linear programming approach for genome scaffolding. WCB: Workshop on Constraint-Based Methods for Bioinformatics, Sep 2014, Lyon, France. 16 p. hal-01198359

HAL Id: hal-01198359
https://hal.archives-ouvertes.fr/hal-01198359
Submitted on 11 Sep 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
An Integer Linear Programming Approach for Genome Scaffolding

Nicolas Briot\textsuperscript{1,2}, Annie Chateau\textsuperscript{1,2,3}, Rémi Coletta\textsuperscript{1,2}, Simon de Givry\textsuperscript{4}, Philippe Leleux\textsuperscript{4}, and Thomas Schiex\textsuperscript{4}

\textsuperscript{1} LIRMM, UMR 5506, 161 rue Ada 34095 Montpellier Cedex 5 - France
\textsuperscript{2} Université Montpellier 2, Place Bataillon, Montpellier - France
\textsuperscript{3} Institut de Biologie Computationnelle, 95 rue de la Galéra, Montpellier - France
\textsuperscript{4} MIAT, UR-875, INRA, 31320 Castanet-Tolosan - France
{briot,chateau,coletta}@lirmm.fr
prenom.nom@toulouse.inra.fr

Abstract. This paper presents a simple and fast approach for genome scaffolding, combining constraint modeling and simple graph manipulation. We model the scaffolding problem as an optimization problem on a graph built from a paired-end reads alignment on contigs, then describe an heuristic to solve this problem with the iterative combination of local constraints solving and cycle breaking phases. We tested our approach on a benchmark of various genomes, and compared it with several usual scaffolders. The proposed method is quick, flexible, and provides results comparable to other scaffolders in terms of quality. In addition, contrarily to state of the art approaches that require dedicated servers, it can be run on a basic notebook computer even for large genomes.

1 Introduction

This paper presents an application of the constraint programming paradigm to a crucial problem in bioinformatics, namely the genome scaffolding problem. The genome scaffolding problem consists in determining a suitable order of the incompletely assembled genome parts, called contigs, provided by common assembly techniques to infer a genomic sequence from Next Generation Sequencing (NGS) data.

The initial formulation as a path-merging problem was proved NP-complete [1]. A similar result was recently proven when the problem is simplified and modeled as an optimization problem in a particular graph called the scaffold graph [2]. Although this problem has already been studied through a variety of computational techniques (see 3), the constraint programming approach together with this model is fully original. The main advantage of this approach is its flexibility, enabling the modeling of various aspects of the problem, by progressive extension towards several directions. Indeed, we may have to adapt to various forms of genomic sequences or data sources, and deal with complex issues like repeated contigs. In this paper, we set the first steps of an efficient tool that aims at performing scaffolding even on large genomes in a reasonable time on
a usual computer. This approach is presented here as a proof of concept of the application of constraint programming to the scaffolding problem.

First, we describe the biological context and mathematical formalization of the problem in Section 2. We present the state of the art in Section 3. In Section 4, we detail the model and algorithms. In Section 5, we describe the chosen experimental settings, as well as the datasets. Finally, we examine the results in Section 6.

2 Genome scaffolding

When the genome of a new species is sequenced, the provided data consist in a huge amount (up to several billion for the longest genomes) of short sequences in which there is a variable fraction of so-called paired-end reads. Sequencing technologies [3] may vary and produce several kind of paired-end reads, but the principle remains the same. A pair of reads is defined by the sequenced extremities of a given genomic fragment (see Figure 1). Pairs can be oriented outwards or inwards, depending on the technology, but the orientation and the rough fragment length are consistent within a given dataset. In the remaining of this paper, without loss of generality, we assume that all paired-end reads are inners (inward orientation).

Notice that, since the genome is oriented by the physical structure of the DNA molecule, reads are also oriented.

Given the reads dataset, the assembly step builds the big puzzle using overlaps between the reads (for a review of usual methods, see [4]). However, due to local lack of coverage (some areas of the genome being harder to sequence than others) and repetitions, the puzzle cannot be completely solved and the assembly produces a set of sequences of various lengths, the contigs (see Figure 2).

Once the contigs are produced, the next step is to order them and determine their orientation, to produce larger parts of the genome called scaffolds (see
We also need to estimate the relative positions of the contigs inside the scaffolds (estimation of gap sizes between successive contigs). Usually, this is determined by the estimated fragment size between both reads in a pair, and the position of the reads on the contigs: the pairing information.

The scaffolding problem is defined as follows: given a set of contigs \( C = \{C_0, \ldots, C_{n-1}\} \), and a set of paired-end reads, we want to infer an order on the contigs, and the orientation of the contigs, that is the most consistent with the pairs. We assume that the reads have already been properly mapped to the contigs and we focus on pairs whose elements are mapped on distinct contigs. Given a fixed orientation of the contigs, there are four ways, called *stories*, in which a pair can map on two distinct contigs, depending on the reads orientations relatively to the contigs orientations (see Figure 4).

The four possible ways of linking two contigs for inward paired-end reads. In story A (resp. B), contigs \( C_1 \) (resp. \( C_2 \)) precedes contig \( C_2 \) (resp. \( C_1 \)) with their original orientations. In story C (resp. D), contig \( C_1 \) (resp. \( C_2 \)), with its alternative orientation, precedes contig \( C_2 \) (resp. \( C_1 \)) with its original orientation.

The scaffolding problem is defined as follows: given a set of contigs \( C = \{C_0, \ldots, C_{n-1}\} \), and a set of paired-end reads, we want to infer an order on the contigs, and the orientation of the contigs, that is the most consistent with the pairs. We assume that the reads have already been properly mapped to the contigs and we focus on pairs whose elements are mapped on distinct contigs. Given a fixed orientation of the contigs, there are four ways, called *stories*, in which a pair can map on two distinct contigs, depending on the reads orientations relatively to the contigs orientations (see Figure 4).

**Fig. 3.** Contigs (segments) are oriented and ordered into scaffolds (boxes) along the genome (above).

**Fig. 4.** The four possible ways of linking two contigs for inward paired-end reads. In story A (resp. B), contigs \( C_1 \) (resp. \( C_2 \)) precedes contig \( C_2 \) (resp. \( C_1 \)) with their original orientations. In story C (resp. D), contig \( C_1 \) (resp. \( C_2 \)), with its alternative orientation, precedes contig \( C_2 \) (resp. \( C_1 \)) with its original orientation.

**Fig. 5.** Three contigs and their linking pairs of reads.
Several pairs could tell the same stories (see Figure 5). In this case, we assume that we can bundle them and attribute a weight to the obtained story equal to the number of stories that have been bundled. The bundling process is similar to the one presented in [1]. The data of the contigs and the weighted stories is called the scaffold graph. We denote by \( m \) the number of distinct stories, meaning the number of weighted edges in the scaffold graph. Since contigs are oriented sequences, they are represented by a pair of extremities linked by an intra-contig edge. The contig numbered \( k \) is represented by two vertices, namely \( 2k \) and \( 2k + 1 \). The example in Figure 5 has six vertices \((k = 3)\) and three edges \((0, 5), (1, 2), (3, 4)\) with weights \( 1, 3, 3 \) respectively \((m = 3)\) and three intra-contig edges \((0, 1), (2, 3), (4, 5)\).

The genomes can be composed of several chromosomes, linear or circular. We want to adopt a general model that allows to handle those different cases. In what follows, we model the problem such that a solution produces a subgraph of the scaffold graph, which has only linear or circular (paths or cycles) connected components, and maximizes the weight of the chosen stories (i.e. weighted edges of the scaffold graph).

In Figure 6 we can see a scaffold graph of size 30 (subgraph of the scaffold graph for the bacterial sequence of a Wolbachia endosymbiont), and the solution which is produced by the optimization process. Notice that, although the original sequence consists in one circular chromosome, there are several linear components, and a very small circular one. The linear components are good candidates to be parts of the circular chromosome, but the small cycle should not appear. In fact, it could be an artifact due to repeats in the sequence. Indeed, the corresponding contig appears several times in the complete genome.

### 3 Previous work on scaffolding

Contig scaffolding is an NP-complete problem, whose solutions can be approximated by a greedy approach. The first greedy approach in [1] is dedicated to genomic fragments (BAC\(^5\)) combined with mate-pairs, which differ from the paired-end reads essentially by their larger insert-size\(^6\), and their lower covering depth, so the size of the data and the organization of the graph may differ from actual NGS data coupled with paired-end reads data. In SOPRA [5], the authors introduce a removal procedure for problematic contigs, together with a simulated annealing technique, to perform the scaffolding. Time performances are not really detailed. In OPERA [6], a graph contraction procedure approach is used to limit the size of the input, allowing the use of an exact dynamic programming approach. It is shown that the exact method gives better scaffold quality than existing heuristics. In SSPACE [7], they use a nearly greedy approach, by combining iteratively the longest contigs. In GRASS [8], a genetic algorithm is provided, using mixed integer programming (MIP) as in [9] and an

---

5 Bacterial Artificial Chromosome

6 The insert size is the size, in base pairs, between the two fragments constituting a pair of reads
expectation-maximization process. They obtain results which are intermediary between SSPACE and Opera. In SCARPA [10], the two problems of orienting and ordering the contigs are separated, as in SOPRA. The orientation of the contigs is an FPT (Fixed Parameter Tractable) problem. Then, pre-oriented contigs are ordered using a heuristic, and the removal of articulation vertices is used to limit the size of the connected components. Misassembled contigs are detected and removed. The authors qualify their method as moderately slow, but using not so much memory.

In all these approaches, repeats are not handled, or in the best case, they are detected and removed.

4 Model and algorithms

4.1 Integer Linear Programming Model

We present here a simple integer linear programming (ILP) model of the problem. Since the weights on the inter-contig edges are integers, this model seems quite natural. The set of variables is just the set of stories, denoted by $V = \{x_1, \ldots, x_m\}$. The domain of these variables is $D = \{0, 1\}$, where 0 means that the story is not chosen, and 1 that it is chosen.

The set of constraints $C$ expresses the constraint on the maximal degree of the chosen subgraph, i.e. a set of paths and cycles. For each contig extremity, we impose that at most one outgoing story is chosen. We do not constrain the
solution to be connected, because we want to avoid choosing an artificial path including stories of too low weight.

Finally, we maximize the objective function corresponding to the weight of the chosen subgraph:

\[ \sum_{j=1}^{m} w_j x_j, \]

where \( w_j \) is the weight of the story \( x_j \).

In Figure 7, we present a scaffold graph for \( n = 7 \) contigs (14 vertices). The ILP model leads to the following set of constraints:

\[
\begin{align*}
    x_1 & \leq 1 & x_8 & \leq 1
    
x_2 & \leq 1 & x_8 + x_9 & \leq 1
    
x_3 & \leq 1 & x_4 + x_7 & \leq 1
    
x_3 + x_4 + x_5 & \leq 1 & x_9 + x_{10} & \leq 1
    
x_1 + x_6 & \leq 1 & x_{10} & \leq 1
    
x_6 + x_7 & \leq 1 & x_2 + x_5 & \leq 1
\end{align*}
\]

and the following objective function to maximize:

\[
32x_1 + 2x_2 + 2x_3 + 28x_4 + 52x_5 + 20x_6 + 80x_7 + 6x_8 + 34x_9 + 4x_{10}.
\]

An optimal solution for this problem has an objective value of 198, selecting edges \( x_1, x_5, x_7, x_9 \) (assigned to 1). The resulting scaffolding contains three scaffolds with contig orderings \((C_1), (C_6, C_2, C_0)\), and \((C_5, C_4, C_3)\).

![Fig. 7. A scaffold graph for \( k = 7 \) contigs.](image)

This model can be implemented using several types of solvers:
– Pseudo-boolean SAT solvers, e.g., SAT4J [11];
– SAT solvers with cardinality constraint encoding [12], possibly improved by [13];
– Max-SAT solvers using unary soft clauses encoding the objective function and hard clauses encoding cardinality (connectivity) constraints as in SAT;
– Integer Linear Programming (ILP) solvers such as the state-of-the-art branch-and-bound solver IBM ILOG cplex.

4.2 Cycles removing

In our model, since we only use local constraints on the degree of each vertex, we allow solution with cycles. Except for a Hamiltonian cycle, which may correspond to a circular genome, we don’t want those cycles. We also make the assumption that these cycles are generated by repeats in the genome. If two contigs $a$ and $b$ form a tandem repeat in the genome, meaning that they appear several times in the genome consecutively in that order, for instance $ababab$, then the stories “$a$ is followed by $b$” and “$b$ is followed by $a$” have biased weights compared to stories which are not duplicated. Thus, the cycle $aba$ may be chosen instead of the sequence $ab$ included in a longer sequence. Our cycle removing step is then also a potential repeats removing step. More work is needed to fully investigate the relation between the cycles that appear in our optimization process and repeats. In this paper, we focus on solving time, in order to be able to tackle large instances.

Several encoding have been proposed to enforce the no-subcycle constraint in CSP, but such a constraint is less easy to express in an ILP context. Thus we decided to address it by a SAT Modulo Theory inspired scheme: we solve the ILP problem with a solver and integrate this resolution in an incremental process, where cycles that may have been produced by the solver are forbidden in the next iteration.

5 Experiments

We present in this section the chosen dataset to explore the efficiency and accuracy of our method. We describe the experimental results in a more complete way than just comparing NG50 scores and computation times. We recall that the NG50 measures the scaffold length such that using longer or equal length scaffolds produces 50% of the bases of the reference genome. The results show that the NG50 is not enough to assess the quality of a scaffolder, and we give other metrics such as corrected NG50 (NGA50) and misassemblies statistics. In what follows, we refer to our method as scaftools.

5.1 Dataset

We performed the experiments on the following dataset, composed of four bacterial genomes and two eukaryotic genomes:
– *Staphylococcus Aureus* (alias *staphylo*), NC_010079 (Staphylococcus aureus subsp. aureus USA300_TCH1516, complete genome)
– *Escherichia Coli* (alias *ecoli*), U00096.2 (Escherichia coli K-12 MG1655, complete genome)
– *Yersinia Pestis* CO92 strain (alias *ypco92*), NC_003143.1 (Yersinia pestis CO92 chromosome)
– *Wolbachia* endosymbiont (alias *wolbachia*), NC_010981.1 (Wolbachia endosymbiont of Culex quinquefasciatus Pel chromosome, complete genome)
– *Homo Sapiens* chromosome Y (alias *chrY*), human_g1k_v37 GRCh37.
– *Arabidopsis Thaliana* (alias *arabido*), accession TAIR10.

The read datasets have several origins:

– Real sequencing dataset for:
  • *Staphylococcus Aureus*, short jump library\(^7\) (from the GAGE dataset [4])
  • *Escherichia Coli*, Illumina reads library SRR001665
  • *Arabidopsis Thaliana*, Illumina reads library SRR616966
– Simulated datasets for:
  • *Wolbachia*, simulated with *toyseq*, a specific tool developed for the Variathon experiment\(^8\) [14]. This dataset presents a minor variant frequency of 20%.
  • *Yersinia Pestis* and the human chromosome Y, simulated with *wgsim*, a classical read simulator\(^9\) [15].

For the read assembly step required to produce the contigs, it has been performed using two different tools (both being de Bruijn graph based):

– *Staphylococcus Aureus*, *Escherichia Coli*, and *Arabidopsis Thaliana* were assembled with *velvet* [16]
– *Wolbachia*, *Yersinia Pestis* and the human chromosome Y were assembled with *minia* [17]

The mapping of the reads on the contigs was performed using *bowtie* [18] for *Escherichia Coli* and *Arabidopsis Thaliana*, and *bwa* [19] for the other datasets. Notice that, according to [20], these mapping tools are among the best ones from the point of view of building scaffolds.

Table 1 provides additional information on the genomes, reads and contigs. Table 2 presents information about the scaffold graphs used as input for *scafftools*. In series of four figures for the number of stories and total stories weight, the first one represents the figure without any filtering, and the other ones when the stories of weight less than 3, 6 and 10 are respectively filtered. Despite the fact that this filtering may seem drastic concerning the number of stories, it does not change the total weight so much, indicating that the data contain a lot of low supported stories.

\(^7\) [http://gage.ccb.umd.edu/](http://gage.ccb.umd.edu/)
\(^8\) [http://bioinf.dimi.uniud.it/variathon](http://bioinf.dimi.uniud.it/variathon)
\(^9\) [https://github.com/lh3/wgsim](https://github.com/lh3/wgsim)
<table>
<thead>
<tr>
<th>Genome</th>
<th>Ref. genome size</th>
<th>Read length</th>
<th>Number of pairs</th>
<th>Insert size</th>
<th>Number of contigs</th>
</tr>
</thead>
<tbody>
<tr>
<td>staphylo</td>
<td>2903081 bp</td>
<td>102</td>
<td>833596</td>
<td>3500</td>
<td>301</td>
</tr>
<tr>
<td>ecoli</td>
<td>4639675 bp</td>
<td>36</td>
<td>20816448</td>
<td>200</td>
<td>866</td>
</tr>
<tr>
<td>yeeco92</td>
<td>4653728 bp</td>
<td>70</td>
<td>2000000</td>
<td>500</td>
<td>4983</td>
</tr>
<tr>
<td>wolbachia</td>
<td>1482455 bp</td>
<td>100</td>
<td>703838</td>
<td>300</td>
<td>280</td>
</tr>
<tr>
<td>chrY</td>
<td>59373566 bp</td>
<td>71</td>
<td>16600000</td>
<td>500</td>
<td>24560</td>
</tr>
<tr>
<td>arabido</td>
<td>119667750 bp</td>
<td>100</td>
<td>49703592</td>
<td>500</td>
<td>172616</td>
</tr>
</tbody>
</table>

Table 1. Statistics on the datasets.

<table>
<thead>
<tr>
<th>Genome</th>
<th>Number of vertices</th>
<th>Number of stories</th>
<th>Total stories weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>staphylo</td>
<td>602</td>
<td>4464/1442/716/489</td>
<td>72729/68875/66213/64602</td>
</tr>
<tr>
<td>ecoli</td>
<td>1732</td>
<td>7276/3177/2239/1829</td>
<td>303161/298351/294808/291911</td>
</tr>
<tr>
<td>yeeco92</td>
<td>2656</td>
<td>2137/1521/1323/1197</td>
<td>69340/68592/67823/66918</td>
</tr>
<tr>
<td>wolbachia</td>
<td>560</td>
<td>756/243/179/119</td>
<td>12234/5013/4412/3851</td>
</tr>
<tr>
<td>chrY</td>
<td>49120</td>
<td>52169/31831/26940/23357</td>
<td>1268481/1242982/1222351/1198088</td>
</tr>
<tr>
<td>arabido</td>
<td>345232</td>
<td>146368/80146/57717/42478</td>
<td>1901142/1819557/1733228/1622083</td>
</tr>
</tbody>
</table>

Table 2. The scaffold graphs.

5.2 Evaluation

We choose to evaluate our method on several criteria. Since one of our goals is to offer a very efficient and quick tool, a major criterion is the computation time. scaftools was run on a usual laptop (Intel i3-2348M with 2 cores at 2.3 GHz and 5.6 GB of RAM), and memory consumption was never a problem. The other scaffolding tools were run on servers of the Genotoul cluster (48 cores AMD Operon 6176 at 2.3 GHz with 128 GB of RAM).

The other criteria concern the quality of the produced scaffolds and are offered by the QUAST tool\(^\text{10}\) [21]. QUAST is originally targeted at assessing assembly quality, but can also be applied for scaffold evaluation. It performs local alignment and rearrangement analysis on the produced scaffolds.

The filling of the gaps for the scaftools method was arbitrary set to 50bp, since it does not yet estimate the size of gaps. Preliminary experiments have shown that this length has not a strong influence on the quality of the results.

The scaffolding of the genomes were performed with respectively:

- SSPACE [7]
- SOPRA [5]
- OPERA [6]
- scaftools
- scaftools + filtering out the stories of weight < 3
- scaftools + filtering out the stories of weight < 6
- scaftools + filtering out the stories of weight < 10

\(^{10}\) bioinf.spbau.ru/quast
6 Results

In this section, we present the results obtained on the different datasets.

6.1 Computation time

The computation times presented on Table 4 show that scaftools is faster than the other methods. The rare exceptions are due to the output writing time, for large genomes, as shown on Table 4, which presents the detailed computation time for each step of the scaftools process.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Opera</th>
<th>SSPACE</th>
<th>SOPRA</th>
<th>Scaftools</th>
<th>Scaftools 3</th>
<th>Scaftools 6</th>
<th>Scaftools 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Aureus</td>
<td>33s</td>
<td>16s</td>
<td>1min40s</td>
<td>≃1s</td>
<td>&lt;1s</td>
<td>&lt;1s</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>E. Coli</td>
<td>3h28min16s</td>
<td>16min4s</td>
<td>39min48s</td>
<td>13s</td>
<td>&lt;1s</td>
<td>7s</td>
<td>8s</td>
</tr>
<tr>
<td>Y. Pestis</td>
<td>39s</td>
<td>44s</td>
<td>6min26s</td>
<td>6s</td>
<td>6s</td>
<td>4s</td>
<td>3s</td>
</tr>
<tr>
<td>Wolbachia</td>
<td>13s</td>
<td>7s</td>
<td>3s</td>
<td>&lt;1s</td>
<td>&lt;1s</td>
<td>&lt;1s</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>Chr. Y</td>
<td>4min27</td>
<td>3min45s</td>
<td>10min54s</td>
<td>7min36s</td>
<td>NA</td>
<td>6min52s</td>
<td>6min4s</td>
</tr>
<tr>
<td>A. Thaliana</td>
<td>22h11min13s</td>
<td>3h25min42s</td>
<td>31h3min24s</td>
<td>8h13min43s</td>
<td>5h24min25s</td>
<td>6h3min9s</td>
<td>4h22min57s</td>
</tr>
</tbody>
</table>

Table 3. Computation time comparison.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Phase</th>
<th>Detailed computation time</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cplex1</td>
<td>CR 1</td>
<td>cplex2</td>
</tr>
<tr>
<td>S. Aureus</td>
<td>0.12s</td>
<td>0.0037s</td>
<td>0.14s</td>
</tr>
<tr>
<td>E. Coli</td>
<td>0.48s</td>
<td>0.0610s</td>
<td>0.08s</td>
</tr>
<tr>
<td>Y. Pestis</td>
<td>0.10s</td>
<td>0.17s</td>
<td>0.11s</td>
</tr>
<tr>
<td>Wolbachia</td>
<td>0.06s</td>
<td>0.0042s</td>
<td>0.08s</td>
</tr>
<tr>
<td>Chr. Y</td>
<td>2.69s</td>
<td>42.06s</td>
<td>4.94s</td>
</tr>
<tr>
<td>A. Thaliana</td>
<td>3.85s</td>
<td>019.44s</td>
<td>5.00s</td>
</tr>
</tbody>
</table>

Table 4. Detailed computation times for scaftools. CR means “Cycle Removal”. The output time does not include the generation of the fasta file.

Table 6 shows that the cycle breaking process presents few iterations, even for large genomes. Thus, the iterative strategy is not expensive and we do not have to fear long collapsing time. Concerning the RAM requirements, memory usage was never a problem during the experiments on a usual laptop. We recall the memory requirements of the other scaffolders on Table 7.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Graph generation time (s)</th>
<th>SAM file size (Mo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Aureus</td>
<td>5.052</td>
<td>272.1</td>
</tr>
<tr>
<td>E. Coli</td>
<td>140.580</td>
<td>4734</td>
</tr>
<tr>
<td>Y. Pestis</td>
<td>13.140</td>
<td>632.4</td>
</tr>
<tr>
<td>Wolbachia</td>
<td>4.720</td>
<td>261.8</td>
</tr>
<tr>
<td>Chr. Y</td>
<td>97.984</td>
<td>4360</td>
</tr>
<tr>
<td>A. Thaliana</td>
<td>304.980</td>
<td>15463.1</td>
</tr>
</tbody>
</table>

**Table 5.** Time to generate the scaffold graph from the SAM file.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of iterations</th>
<th>Number of cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Aureus</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>E. Coli</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Y. Pestis</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Wolbachia</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Chr. Y</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>A. Thaliana</td>
<td>4</td>
<td>60 – 7 – 2</td>
</tr>
</tbody>
</table>

**Table 6.** Number of iterations and cycles

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Opera</th>
<th>SSPACE</th>
<th>Sopra</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Aureus</td>
<td>336M</td>
<td>496M</td>
<td>1.150G</td>
</tr>
<tr>
<td>E. Coli</td>
<td>10.745G</td>
<td>1.453G</td>
<td>16.878G</td>
</tr>
<tr>
<td>Y. Pestis</td>
<td>345M</td>
<td>462M</td>
<td>685M</td>
</tr>
<tr>
<td>Wolbachia</td>
<td>251M</td>
<td>448M</td>
<td>230M</td>
</tr>
<tr>
<td>Chr. Y</td>
<td>262M</td>
<td>512M</td>
<td>1.718G</td>
</tr>
<tr>
<td>A. Thaliana</td>
<td>15.167G</td>
<td>9.355G</td>
<td>25.169G</td>
</tr>
</tbody>
</table>

**Table 7.** Memory usage for Opera, SSPACE, and Sopra.
6.2 Quality of the scaffolding

QUAST provides a variety of measures to estimate the quality of a set of sequences (usually contigs, here scaffolds) relatively to a reference genome. Table 8 presents some of these measures and Figure 8 shows a visualization of the Opera and Scaftools scaffolds for the *staphylo* dataset. The NGA50 is the NG50 where the lengths of aligned blocks are counted instead of scaffold lengths, i.e., if a scaffold has a misassembly with respect to the reference, the scaffold is broken into smaller pieces.Opera and Scaftools show larger NG50 statistics because they are more aggregative, but it does not mean that this aggregation is always better.

Results on other genomes are similar, detailed results can be found at http://www.lirmm.fr/~chateau/scaftools/v_0_1/results/.

As could be expected, the blunt Scaftools method without filtering leads to more errors than the classical methods, but this is improved by filtering the low supported stories. Notice that the results are then comparable to those provided by other scaffolders.
Fig. 8. Visualization of the S. Aureus assembly with Opera (left) and Scaftools 10 (right) respectively on y-axis versus the reference genome on x-axis. Each horizontal dotted line separates two different scaffolds.
<table>
<thead>
<tr>
<th>Statistics</th>
<th>Contigs</th>
<th>Opera</th>
<th>SSPACE</th>
<th>SOPRA</th>
<th>Scaftools</th>
<th>Scaftools 3</th>
<th>Scaftools 6</th>
<th>Scaftools 10</th>
</tr>
</thead>
<tbody>
<tr>
<td># scaffolds</td>
<td>301</td>
<td>194</td>
<td>174</td>
<td>300</td>
<td>150</td>
<td>164</td>
<td>134</td>
<td>97</td>
</tr>
<tr>
<td>Assembly length (bp)</td>
<td>2860307</td>
<td>2899757</td>
<td>2853630</td>
<td>2860279</td>
<td>2867857</td>
<td>2865655</td>
<td>2862009</td>
<td>2856206</td>
</tr>
<tr>
<td>NG50</td>
<td>48440</td>
<td>364718</td>
<td>48440</td>
<td>48440</td>
<td>895496</td>
<td>895353</td>
<td>895118</td>
<td>895118</td>
</tr>
<tr>
<td>NGA50</td>
<td>48149</td>
<td>197571</td>
<td>48149</td>
<td>48149</td>
<td>171910</td>
<td>171910</td>
<td>171910</td>
<td>171910</td>
</tr>
<tr>
<td># misassemblies:</td>
<td>9</td>
<td>16</td>
<td>9</td>
<td>9</td>
<td>56</td>
<td>42</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td># local misassemblies</td>
<td>8</td>
<td>81</td>
<td>8</td>
<td>8</td>
<td>74</td>
<td>73</td>
<td>72</td>
<td>69</td>
</tr>
<tr>
<td># predicted genes (unique)</td>
<td>2720</td>
<td>2693</td>
<td>2721</td>
<td>2720</td>
<td>2721</td>
<td>2719</td>
<td>2710</td>
<td>2706</td>
</tr>
</tbody>
</table>

Table 8. QUAST scaffolding analysis for the *staphylo* dataset. We display here 7 different statistics for the assemblies. **#scaffolds:** the number of scaffolds in the assembly. **Assembly length:** total length of the assembled scaffolds in base pairs. **NG50:** scaffold length such that using longer or equal length scaffolds produces half (50%) of the bases of the reference genome. **NGA50:** NG50 where the lengths of aligned blocks are counted instead of scaffold lengths, i.e. if a scaffold has some misassemblies with respect to the reference, the scaffold is broken into smaller pieces. **# misassemblies:** number of positions in the assembled scaffolds where (1) two contigs should be adjacent but are separated by over 1kbp or overlap on more than 1 kbp: relocation or (2) two contigs appear on different strands: inversion or (3) they appear on different chromosomes: translocation. Notice that, for all scaffolders and genomes included, most misassemblies were relocations. **# local misassemblies:** number of relocations where two contigs should be adjacent but are separated by over 1kbp or overlap on more than 1 kbp. **# predicted genes (unique):** number of unique genes in the assembly identified by GeneMark.hmm.
7 Conclusion and future works

We developed and analyzed an efficient method for scaffolding, which defines a proof of concept with encouraging results when compared to other methods. There are still a variety of possible improvements that need to be explored in future work. For instance, we aim at integrating the distances between contigs in the scaffolding process. This information can be extracted from the mapping results, based on the mean insert size. This may lead to an improvement of the quality of the stories, since we will be able to put aside pairs of reads showing inconsistent distance information. We also could improve the model by accounting for possible multiplicities of the contigs (repeats), either by inferring those multiplicities using the mapping profile, or by deeply investigating the connexion between artefactual small cycles and repeated contigs. There are multiple ways to integrate this information in our model and it is still unclear which approach is the most favorable. Finally, the computation time can still be further improved, by exploiting, for example, incremental resolution of the ILP problems instead of iterated solving\textsuperscript{11}.

Acknowledgements

This work was partially funded by the Institut de Biologie Computationnelle. Thanks to Laurent Lemarchand, for his ‘last minute’ interesting discussion on CPLEX. We also thank to Toulouse Christophe Klopp for introducing the scaffolding problem to us and Matthias Zytnicki for his help on collecting the datasets.

References


\textsuperscript{11} using, for instance, the mulprod\_main module for CPLEX.


