MASTER'S THESIS

UNIVERSITY OF RENNES 1
BIOINFORMATICS AND GENOMICS MASTER'S DEGREE
(2014 - 2015)

TEST AND BENCHMARKING OF A NEW SCAFFOLDING METHODOLOGY

INSTITUTE FOR RESEARCH IN IT AND RANDOM SYSTEMS, GENSCALE
263 AVENUE GENERAL LECLERC, 35000 RENNES, FRANCE

Author:
ALEXANDRINA BODRUG

Supervisors:
Pr. UNIV. RENNES 1 RUMEN ANDONOV
Dr. CNRS DOMINIQUE LAVENIER

22 JUNE, 2015

Thanks

Abbreviations

Acorus calamus chloroplastic genome	acorus
Agrostis stolonifera chloroplastic genome	agrostis
Atropa belladonna chloroplastic genome	atropa
Cucumis sativus chloroplastic genome	cucumis
Eucalyptus globulus chloroplastic genome	eucalyptus
Euglena gracilis chloroplastic genome	euglena
Genscale Scaffolding Tools	GST
Lecomtella madagascariensis chloroplastic genome	lecomtella
Oenothera elata chloroplastic genome	oenothera
Oryza sativa Japonica chloroplastic genome	rice
Pinus koraiensis chloroplastic genome	pinus
Wolbachia endosymbiont bacterial genome	wolbachia

Git repository

https://github.com/aliecs/Stage2015_gr

Contents

1 Introduction

1.1 Context

The nucleic biological molecules of genomes and transcriptomes can be sequenced by many different techniques. Sequencing is the process which determines the nucleic order within DNA and RNA. The first methods involved the extension of location-specific primers, meaning portions of the genome or transcriptome had to be known in advance. An improved and widely known version of this method was developed by Frederick Sanger in 1970. Sanger's sequencing uses radioactive dideoxynucleotide stopping the DNA replication - for each elongated DNA molecule the last nucleic acid is thus determined. These techniques are costly and slow because they involve handling a DNA molecule for each nucleic acid position and genomes can have as many as 150 billion base pairs. With the increasing demand for genomic and transcriptomic sequencing, New Generation Sequencing (NGS) methods were introduced. NGS is faster and sequences the whole genome at once. The Illumina NGS technology does so by fragmenting the genomes into DNA fragments and determining the nucleic order of their extremities thanks to fluorescently-labeled nucleotides. In order to recover the whole genomic sequence the small sequenced parts, called reads, need to be pieced together. The GENSCALE team at the INSTITUTE FOR RESEARCH IN IT AND RANDOM SYSTEMS uses data obtained though the Illumina NGS technique to develop new read assembling strategies.

1.2 Background

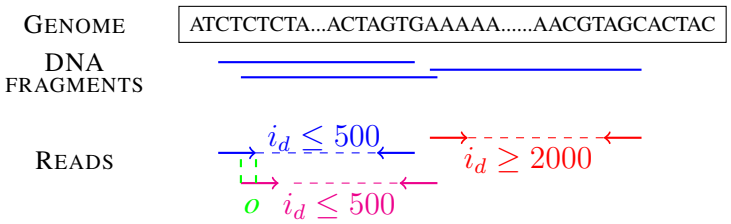
De novo assembly is the process which pieces together overlapping small fragmented DNA sequences produced by NGS methods into larger sequences. The aim is to obtain complete genomes (or chromosomes) containing gaps of known lengths because the less fragmented the genome is, the easier the downstream analysis are[1]. However an incomplete assembly is still sufficient for most of the analysis performed on DNA which explains why databases mainly contain partially assembled genomes. Nonetheless the uninterrupted genome sequence is a precious information and there has been an important effort made to improve the performance of assembly algorithms and the quality of NGS data. The detailed process of assembly is described in section 1.3 page 2; the two main steps are building large uninterrupted sequences (contigs or unitigs) from overlapping reads and scaffolding, the ordering and relative orientation of these large sequences. The 2011 and 2013 Assemblathon projects[2][3] aimed at benchmarking existing assembly tools with high coverage diploid genomes. The studies focused mainly on the contig building step, concluding that although many tools found quality assemblies, the tool and quality criteria should be adjusted to the type of genome and the goal of the assembly project. For example a good N50, an extensively used metric which is the contig length such that using equal or longer contigs produces half the bases of the genome, is not essential in a gene detecting assembly project.
The first stand-alone scaffolder named Bambus[4], originally part of the MetAMOS[5] assembly and analysis pipeline, was published in 2004. Previously the scaffolding step was missing or presented as an option within conting builders, for instance the Velvet[6] assembler *'scaffolding yes or no'* option. In the 2014 comprehensive evaluation of scaffolding tools[7], Hunt *et al* found that no tool identified more than 90% of joins between real-data Velvet assembled contigs, meaning genomes were still fragmented into many scaffolds as joins were missing for a complete and accurate ordering and orientation. The study also used simulated data highlighting the fact that perfect data doesn't always yield perfect results. Despite its simply formulated goal - order and orient contigs - scaffolding is a challenging computational problem. It was first described and modeled in 2002 by Hudson *et al.*[8] which proposed a greedy path-merging strategy, described in section 1.4 page 5 along with other solving methodologies and the distinctive features of the GENSCALE Scaffolding Tools (GST). Concepts surrounding the Illumina NGS technique are explained in section 1.3 page 2

1.3 Assembly terminology

In this report *assembly* will refer to the whole multi-step process starting from once filtered out-of-the-sequencer data and resulting in a highly uninterrupted sequence of a genome or chromosome, in the best case scenarios. As previously mentioned, the two main steps are contig/unitig building and contig/unitg scaffolding. The difference between contig and unitig is fundamental to understanding the GENSCALE scaffolding challenges and is explained in section 1.3.2 page 2. Another key point is the construction of joins between contigs/unitigs - also referred to as links, edges or bonds according to the way they are modeled.

1.3.1 Reads, pairing and overlaps

A read is a short ($< 500pb$) copy of a DNA fragment of known length and nucleic acid sequence. It is produced differently depending on the sequencing technology. Paired reads are copies of the two extremities of the same DNA molecule. The DNA molecule size between two reads of a pair is called an insert. The size of the insert is variable. Reads with small insert sizes ($< 500bp$) are called paired-end reads. Mate-paired reads are reads whose insert size is very big (up to tens of kilobases). The pairing information, the read size and the size of the insert are provided by the sequencer. A collection of reads with their associated insert size is called a a genomic library.



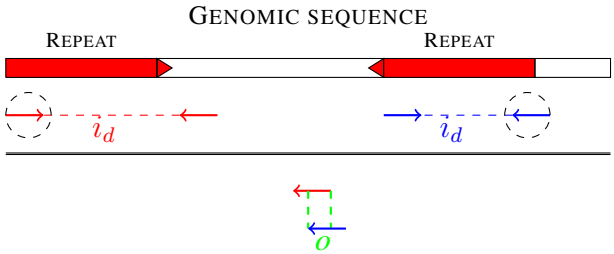
Each end of a DNA molecule is cloned to produce paired reads. Here is represented a mate-paired pair (red) with a big insert size *(i_d)* and two paired-end pairs (blue and magenta) which slightly overlap *(o)*.

Figure 1: Alignment of paired reads on fragmented DNA molecules obtained from the genome

Figure 1 shows three pairs of reads. Within the pairs, reads are facing each other: this configuration is called *Forward-Reverse* read orientation. To be sequenced the genome represented in figure 1 is first fragmented into numerous DNA molecules by sonication or nebulization then amplified by Polymerase Chain Reaction. Each end of the molecule is then fixed on sequencing wells. Overlapping of reads occurs when two reads sequence a portion of the same genomic region, but not only. The overlapping concept implies a common origin but unfortunately overlapping can occur if two reads sequence two different repeated genomic regions. Figure 2 shows how repeated regions create false positive overlaps. Such reads can be detected and filtered out by ignoring high-frequency overlaps (higher than the coverage at which the genome was sequenced). However this can result in false negatives and makes the task of assembling repeated regions very hard.

1.3.2 Unitigs and Contigs

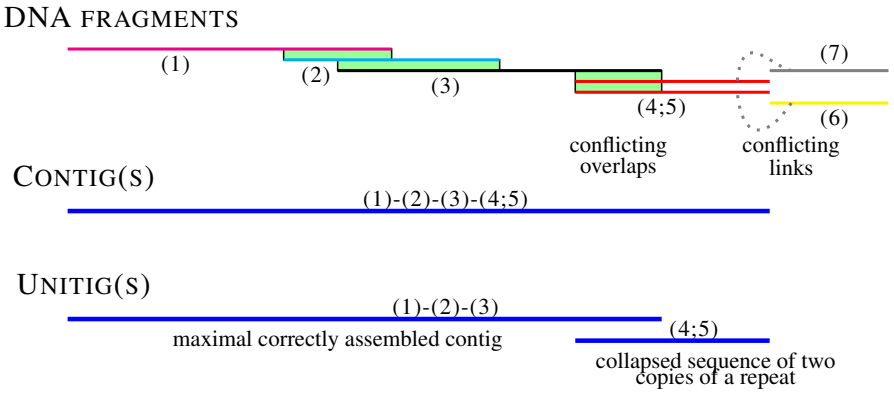
Unitigs are an uniquely assemblable subset of overlapping fragments. At the end of an unitig data shows multiple dubious overlaps as seen in section 1.3.1 page 2 creating joins with multiple other unitigs. Contigs are larger than unitigs, extended through repeat boundaries but are still ungapped sequences. Contigs are interesting to construct because there is a higher chance to detect genes. Taking the example shown in figure 3, a contig will merge the first three DNA fragments and will then be extended though the ambiguous overlaps, merging the red DNA fragments' sequence. Unitigs however will stop at the end of the third DNA



The two circled reads will have a significantly long and accurate overlap to imply a common genomic origin when in fact they come from distant regions.

Figure 2: Overlapping induced by repeated sequences

fragment and assemble the red fragments separately. In a sense, unitigs are either an unambiguous contig or a compression of several copies of a repeat. The advantage of working with unitigs is that there are less chances of erroneous merging of two far away genomic regions. This feature is used in the Genscale scaffolding strategy, further discussed in section 2.1 page 8.



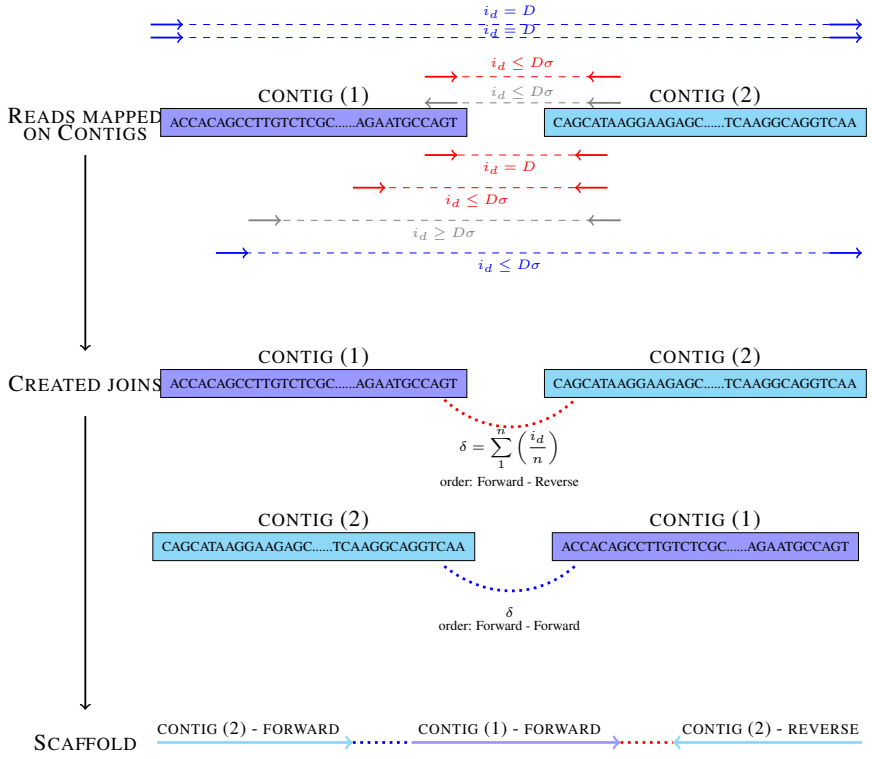
Unitigs end at multiple overlaps indicating a possible repeat. Contigs can be extended through conflicting overlaps. Here, the red DNA fragments are two copies of a repeat. When no more overlaps exist, contigs can be linked (gray dotted line) thanks to information provided by read pairs. This is the scaffolding task. Here alternative paths are possible due to the repeated region.

Figure 3: The difference between unitigs and contigs

1.3.3 Obtaining scaffolds

A scaffold is a linear ordering of contigs (or unitigs). The ordering and relative orientation of contigs is possible thanks to paired reads information. The first step of scaffolding is mapping reads on the previously constructed contigs: the two most used mappers are bwa[9] and bowtie[10;11]. A pair of reads mapping on two different contigs provide a join, which holds the information of distance between the two contigs, and relative orientation (see figure 4). A same contig can have joins with multiple other contigs (see CONFLICTING JOINS figure 3 and CREATED JOINS figure 4). These multiple joins which result in multiple paths when ordering and orienting contigs are solved differently by scaffolders. Most of the time, a choice is made - heuristically - to use one join over another. Another strategy is illustrated in figure 4, where the contig with the two high-confidence joins is duplicated. This is the strategy of the Genscale scaffolders, further described in section 2.1 page 8.
The concept of insert size is essential to understand the challenges of scaffolding when dealing with repeated regions in genomes. Take CONTIG (1) and CONTIG (2) in figure 4: the two contigs are separated

by a gap of undefined size. Say this gap is caused by a repeated region where reads mapped multiple times and were thus discarded, or the region wasn't sequenced by the sequencer and is all together absent. If the region is bigger than the insert size, no read pair will span over it. The join between contig (1) and contig (2) will not exist. This explains why mate-pair information is extremely useful. When multiple mate-pair libraries with different big insert size are available, genomically distanced regions can directly be ordered and orientated. The scaffolds will potentially be longer, as missing or ambiguous data doesn't impede its construction.



i_d : insert size , D : expected insert size , σ : distance standard deviation , n : number of retained correctly mapped paired reads for the join, δ estimated distance.

Paired reads are mapped on previously assembled contigs. Pairs with reads mapping on different contigs provide linking information. The library represented is Forward - Reverse (reads are facing each other, see red pairs). Additional read pairs with a satisfying i_d but with a different orientation can coexist with the Foward - Reverse pairs (here, blue pairs, Forward - Forward). Pairs which map with a big insert size ($\geq D3\sigma$) are usually discarded (here, gray pairs). Red and blue pairs are retained to create two conflicting joins between CONTIG (1) and CONTIG (2). The conflict can be solve with a scaffolder which duplicated CONTIG (2) allowing both joins to coexist, or ignoring one of the joins (not represented).

Figure 4: Creating joins between contigs thanks to read pair information

* * *

The list of contigs or unitigs and the list of joins between them is the minimal data to provide to a scaffolder tool. There are different ways to model the problem computationally. Additional information such as join coverage, contig coverage and join length can be introduced to help solve the scaffolding problem.

1.4 A history of scaffolding strategies and modeling

The scaffolding problem was first introduced in 2002 by Hudson *et al.*[8] following the challenges which arose during the human genome clone-by-clone sequencing by *Lander et al.*[12] and the human whole genome shotgun assembly project by Venter *et al.*[13], both published in 2001. The Hudson *et al.* paper defines the problem as follows:

> "The *Contig Scaffolding Problem* is to order and orientate the given contigs in a manner that is consistent with as many mate-pairs as possible".

The most common way the problem is modeled is as a graph where vertices represent contigs and links represent bundles of pairs of reads joining two contigs (see figure 4, red and blue pairs are bundled into two joins which will be represented links in the graph). Strategies to solve it and find the best path are numerous. It is important to note that Hudson *et al.* regard **the use of mate-pairs as crucial**. There are indeed several scaffolding tools that accept paired-reads libraries with small insert sizes as valid input data - in these cases it is not expected to obtain a quality scaffolding solution. In this section the Hudson *et al.*[8] modeling and scaffolding strategy are detailed, and several scaffolding tools roughly described.

1.4.1 Examples of scaffolding tools

Hudson *et al.* proposed the first heuristic greedy path-merging algorithm for solving the scaffolding problem. Real data can and will be noisy. Inconsistent data is at the basis of the constraints defined for the problem - joins between contigs can be mutually exclusive, erroneous or absent. These features cause the scaffolding problem to be NP-hard. A heuristic approach is a pertinent approach so that scaffolding tools are prepared to quickly take decisions when faced with inconsistent data.

Greedy path merging approach Hudson *et al.*'s scaffolding graph models each contig as two vertexes `Contig - start` and `Contig - end` connected by a `Contig - EDGE` (actually a link, as it is directed `start-> end`). This is a strategy to model the two possible orientations of a contig, read from start to end (*Forward*), or from end to start (*Reverse*). The starting and ending point of a contig are arbitrary as contig building does not provide direction. `Mate - EDGES` are joins between the start (or end) of a contig and the start (or end) of another contig. The weight on these edges is the edge coverage (how many times the created join is supported by data). These edges are undirected. This edge weight conveys the idea of join coverage however the contig coverage is not taken into account in this model. In fact, suspected repeated regions are removed during the graph solving process. The length between two mate pairs (insert size) is not represented in the final graph either. It is only used to allow edge bundling when several mate-pairs support the same join between contigs. The edge bundling is performed by first greedily choosing a median length of a `Mate - EDGE`, then gathering all `Mate - EDGES` that are within $3(standarddeviation)(medianlength)$ to bundle. The weight of `Mate - EDGES` is 1 if it is a single edge and is the number of mate-pairs participating if the `Mate - EDGES` is a bundle. The information of length between contigs is lost. Transitive reduction decreases the number of edges and leads to the final form of the graph that will be solved by the greedy path merging algorithm. The algorithm constructs a path though the scaffolding graph. The `Mate - EDGES` are processed according to their weight (decreasing order) and the found paths are progressively merged until there are no more `Mate - EDGE` left. As `Mate - EDGE` are processed one by one in a determined order, this method is may not be optimal or perfect.

Algorithm 1 Greedy path-merging algorithm as presented in Hudson *et al.* 2002 paper
$H(Path)$ is the sum of mate edge weights which supports $Path$ and $U(Path)$ is the sum of mate edge weights which contradict $Path$

begin
Select all contig edges
for each mate-edge *e* in descending order of weight:
    if e is not selected:
        Let *v*, *w* denote the two nodes connected by *e*
        Let $P1$ be the selected path incident to *v*
        Let $P2$ be the selected path incident to *w*
        if $P1 \neq P2$ and we can merge $P1$ and $P2$ (guided by *e*) to obtain P:
            if $H(P) - (H(P1) + H(P2) \geq U(P) + U(P2))$:
                Replace $P1$ and $P2$ by P
end

SSPACE The SSPACE scaffolding algorithm stores orientation and position of paired contigs by a join during its mapping step. This is list is the starting point of the scaffolding step: SSPACE is one of the few successful scaffolders that does not model the problem as a graph. The scaffolding itself starts with the largest contig and iteratively combines other contigs to it if a minimum number of read pairs exists between them (the default being 5 paired reads joining the two contigs). Alternative connections are dealt with thanks to join coverage as seen in figure **??**. Contig coverage is not taken into account. The scaffolding process stops when the last combined contig has no joins.

Bambus Bambus uses a greedy algorithm to join contigs with the most joins first and ignores subsequent edges that conflict with the formerly used join. This methodology is heavily heuristic as previously accepted joins can be sub-optimal and will cause bias for all the following contig joins.

SOPRA SOPRA was the first tool to try solving the problem exactly. It models the problem as an undirected graph. It removes inconsistent edges and nodes that give rise to spurious links (a choice is presented, ambiguity ensues). This is repeated until no more edges or nodes have to be removes - everything is consistent. Any contig containing repeated regions will thus be discarded.

SCARPA SGA uses a very conservative approach by disallowing any conflicts in the modeled graph, avoiding heuristics at the expense of missing valid joins. Any contig containing repeated regions will thus be discarded.

GST - Genscale Saffolding Tools GST model the problem as a directed graph and try to solve the problem by finding the optimal path. The GST include several models. Their common feature is the fact that they heavily rely on contig coverage and create a node for each contig occurance and orientation. See further explanations about the modeled graph in section 2.1.3 page 10. Conflicting join data will co-exist in the graph and given a choice, the optimal path may be different according to the objective function. Differences between GST models are described in section 2.1.4 page 11.

1.4.2 Limits of the scaffolding tools

Erroneous data (fake links due to poorly assembled contigs, low quality libraries), missing data (low quality libraries, unfit insert size, low genome coverage) and inherent genome characteristics (repeated

regions, heterozygosity) stand in the way of a perfect and easy scaffolding process. Scaffolders employ strategies to solve the ambiguities heuristically (trusting mate-pair join occurrences, ignoring conflicting links with the best path constructed from an arbitrary chosen starting contig, starting from the biggest contig). Those tools which try to solve the problem exactly are too strict regarding multiple possible paths. Last but not least, the contig coverage is not their priority. Repeats are not handled, or at best detected and removed. Considering conflicting joins arise mainly from repeated regions, at least in perfect simulated data without sequencing errors, this kind of information ought to be given some importance. Also, scaffolding tools provide a single solution when multiple versions of the same genome can cohabit within the same organism or cell.

1.5 Goal of the internship project

The goal of this report is to present the performance of the GST with reliable simulated data. This is done to study its decision making in case of conflicting joins that will absolutely arise in real life and are allowed to co-exist in the GST problem modeling. The first step is to evaluate the complexity of the scaffolding modeled graph and compare the solutions obtained by GST with the provided genome reference. The methodology is described in section 2.1 page 8. The second step is to benchmark the GST against published tools. In order to achieve this task, a benchmarking workflow was set up and is described in section 2.3 page 12. Results are presented in section 3 page 14. Global metrics are presented for all instances and a detailed study is made for the chloroplastic genome of *Agrostis stolonifera* and bacterial genome of *Wolbachia endosymbiont*. The section 4 page 21 will provide an insight of the context of the GENSCALE scaffolding project in the light of genome sequencing technique progress. Finally, the conclusion is on page 21.

2 Material and methods

2.1 Genscale scaffolding methodology

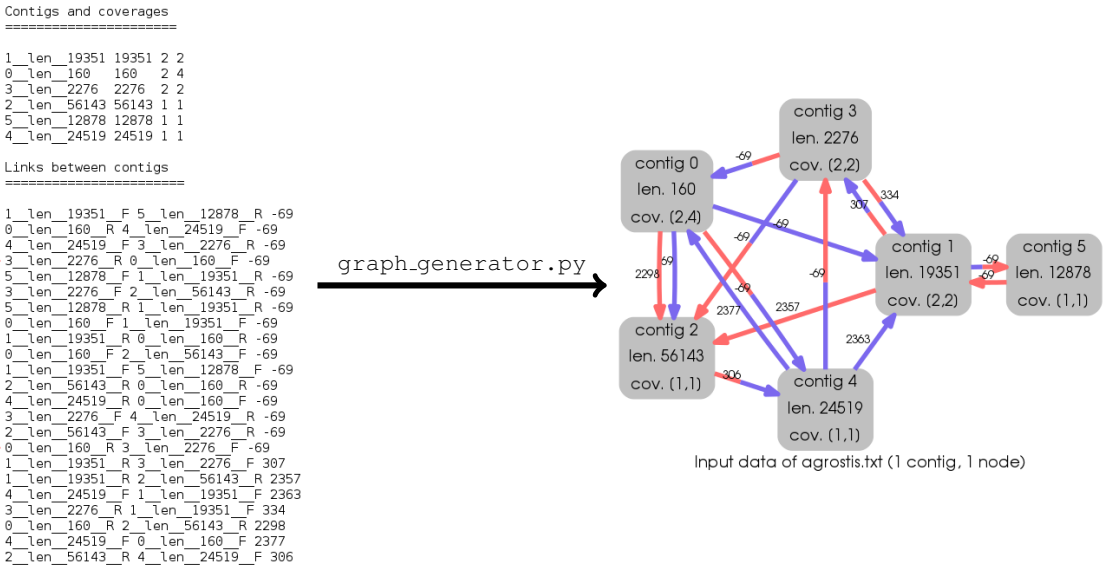
2.1.1 Format of the input data for genscale scaffolders

GST model the scaffolding problem as a graph where vertices are unitigs - *not contigs* - and links (directed edges) are previously obtained bundles of paired reads joining two unitigs. An example of file read by solvers is showed in figure 5 along with its graphical representation generated by the `graph_generator.py` script. The number of links in the *.txt* file is higher than the number of links drawn in the graph because the script merges reverse-equivalent links.

What is a reverse-equivalent link?

Take unitig 0 and unitig 3 of figure 5. In the *.txt* file link list there are two links: `(3R -> 0F -69)` and `(0R -> 3F -69)`, which in fact represents the same link between the two unitigs.

```
                Reverse-equivalent links
(3 Reverse -> 0 Forward) ≡ (0 Reverse -> 3 Forward)
(3 Forward -> 0 Reverse) ≡ (0 Forward -> 3 Reverse)
(3 Forward -> 0 Forward) ≡ (0 Reverse -> 3 Reverse)
(3 Reverse -> 0 Reverse) ≡ (0 Forward -> 3 Forward)
```



The *.txt* file contains a list of unitigs with their associated length and coverage. When the coverage is > 1 the unitig is a repeated sequence. The advantage of using unitigs and not contigs is that the associated coverage is easier to determine (for long unitigs) and more trustworthy. For small unitigs, the coverage is an interval (here, the 160 base unitig 0). The joins between unitigs are orientated and contain the contig orientation (*Forward* or *Reverse*) information. The distance of each link is negative when two unitigs overlap or positive when two unitigs are separated by a gap. Positive distances (gaps) are obtained though mate-pair information. In the graph orientations are represented by color: red for *Reverse* and blue for *Forward*.

Figure 5: Input data of *Agrostis stolonifera* chloroplast genome in *.txt* format and its graph representation

How is the *.txt* file obtained?

The data used in this report is artificial. It is simulated from fasta files of reference genomes using two different script to generate a paired-end read library and a mated-pair read library. The paired-end read library is used to built unitigs with MINIA, a contig building tool developed at GENSCALE which supports an unitig building option. The size of the k-mer (minimal number of overlapping bases between two reads for the overlap to be regarded as valid) is chosen so that MINIA produces the fewest number of unitigs. At this stage published scaffolders such as SSPACE can already be used - they require the unitig file and the

mated-pair library and perform the read mapping themselves to detect joins. The GST require the mapping process to be separately performed and its input file to roughly resemble the one showed in 5. The GST mapping process has three goals:

- mapping mate pairs on large unitigs $\mapsto$ insert size
- mapping paired reads on unitigs $\mapsto$ unitig coverage
- mapping mate pairs on different unitigs $\mapsto$ links between unitigs

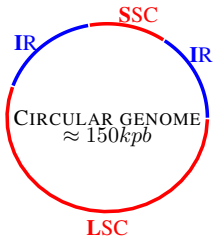
The bigger the unitig, the more robust the mapping information (homogeneously mapped reads) - hence no exact value for small unitigs.

2.1.2 Features of the assembled genomes

As the aim of the Genscale scaffolding project is to produce a complete genome rising to the challenge of repeated sequences, chloroplastic and bacterial genomes are well suited to test the performances of the GSTools. Moreover, chloroplastic and bacterial genomes are small enough to enable a fast and detailed assessment and benchmarking of found solutions.

Chloroplasts

Chloroplasts are small organelles in plant photosynthetic tissues which possess their own DNA. The chloroplast genomes are small ($\approx 150kpb$), circular and have a large inverted repeated sequence of around $25kpb$. Some instances used in this study lack this repetition. This is the case in *Pinus koraiensis* and *Euglena gracilis*. However these two genomes possess significantly more small ($<$ 20bp) repeated sequences.

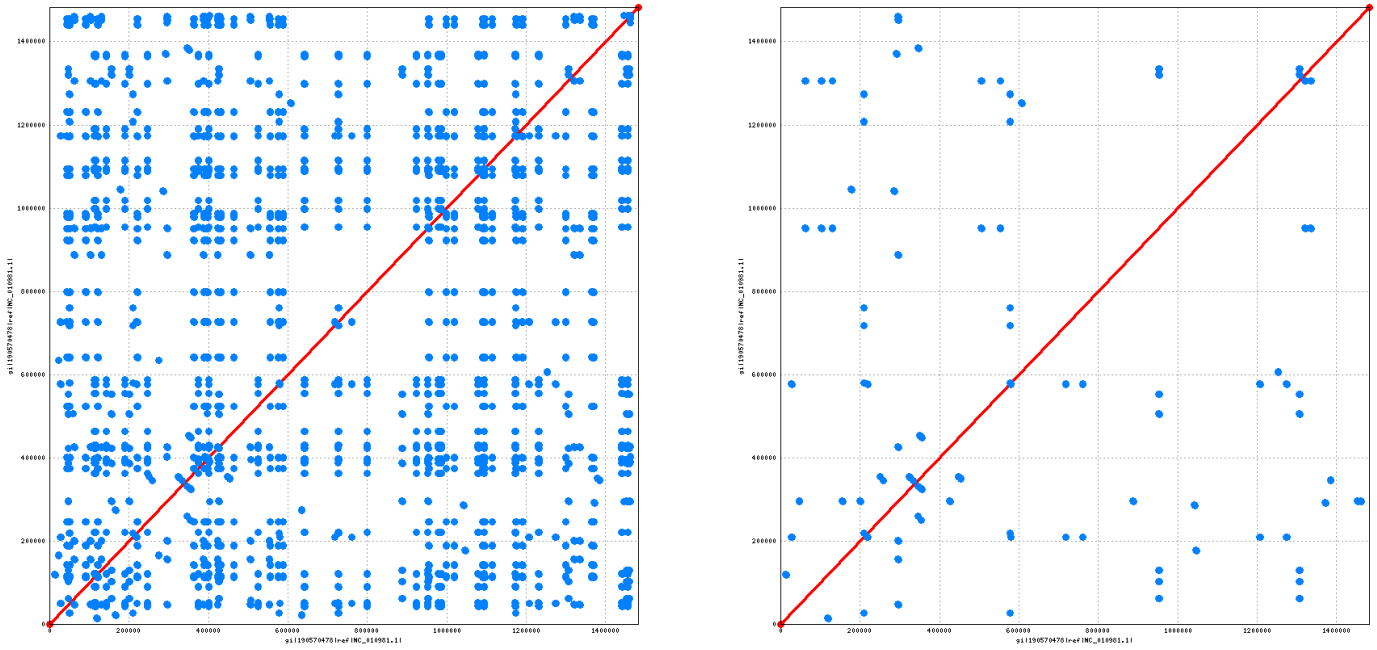


Inverted Repeat (IR $\approx 23kpb$) ; Long Single Copy (LSC); Small Simple Copy (SSC $\approx 85kpb$)

Figure 6: Chloroplast genome structure

Bacteria

Bacterial genomes are bigger ($>$ 1Mpb) and contain many small repeated sequences between 500pb and 1000pb.

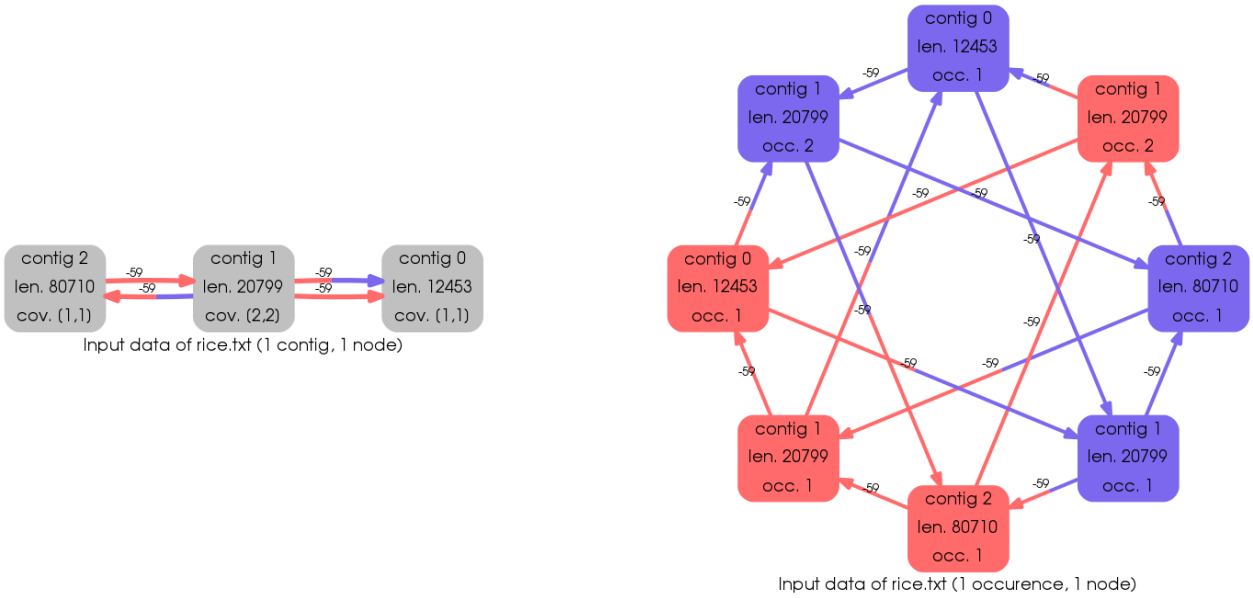


The dot-plots were generated by the nucmer script of the MUMmer sequence fast-alignment tool. (x) and (y) axis are both the*Wolbachia endosymbiont* reference genome downloaded on NCBI (*NC_010981.1*). The blue dots and lines show repetitions in the genome as the DNA sequence matches in several places. The closer the dot is to the red matching axis, the closer the repeats are. We can say, for instance, that at the 1.3Mbp there is a couple of tandem repeats. On the left repeats monger than 500bp are represented (`nucmer -l 500` option). On the left repeats longer than 1200bp are represented.

Figure 7: Wolbachia endosymbiont genome dotplotted against itself

2.1.3 Genscale scaffolding problem modeling

As seen in figure 5, the input data can be visualized by the `graph_generator.py` script as a graph where nodes uniquely represent unitigs regardless of their coverage or orientation. However, this visualization is only useful for a first human assessment of the data. Genscale scaffolding tools model takes each unitig and multiplies it by the number of occurrences. The `unitig 3` with coverage 2 is transformed into `unitig 3 occurance 1` and `unitig 3 occurance 2`, two distinct nodes in the graph. The total number of contig occurrences is then duplicated to model the *Forward* and *Reverse* orientation. `unitig 3 occurance 1` is transformed into `unitig 3 occurance 1` *`Forward`* and `contig 3 occurance 1` *`Reverse`*. The number of links increases consequently. When transforming joins into model graph links, no duplicate links are allowed (merged) and for each link its reverse equivalent is created. Link coverage is not taken into account. Weight link is the estimated distance between two nodes, which is the space left between the two joined tips of two unitig. A distance is negative when two unitig overlap and positive when there is a gap. Gaps are obtained thanks to mate paired information: in this case the distance is the estimated length of the pair's insert size minus the remained sequence length of the unitig following the reads' mapping. A simple example of the difference between the raw input graph and the processed model input graph that the GST solve is presented in figure 8.



Sum of unitig occurrences is 4 so the final number of nodes in the model graph is $4 * 2 = 8$. There are no duplicate links in this example. Flipping the model graph (change the link orientation and node color) produces all the reverse complement links.

Figure 8: Input graphs of rice, as observed in the txt file (left) and as modeled by the Genscale scaffolding tools (right)

2.1.4 Differences between Genscale scaffolding tools

Although all GST try to solve the problem exactly, different models were developed having different objective functions and input information taken into account.

Weighted path model Weighted math model (wpm) does not take link weight into account and only aims at solving the order and relative orientation of unitig. The distance between unitig is not known. This model produces several solutions, among them several can be valid or sub-optimal ones. The objective function of this model maximizes the number of links (bundled read pairs) whose orientation corroborates the suggested orientation of unitigs they are linking. All nodes must be visited once because this model doesn't support intervals as unitig coverage. Visually, it means that each node has $inDegree = outDegree = 1$ and $color(inArrow) = color(outArrow) = color(Node)$.

Distance based model Distance based model (dist) gives weight to links which is the sequence size between two extremities of two unitigs: negative weight in case of an overlapping and positive weight in case of a mated-pair estimated insert size. It produces a single solution. On an axis from 0 to ∞ it tries to find the best position for a unitigs using linear programming so that the number of errors each read pair is minimized. The less contradicting data used, the better. The solution provided can contain two consecutive unitigs for which no join exists in the input data if positioning them so helps reducting the number of errors.

Flow model Flow model is currently in the developing process and it accepts slightly different input data information as the previous ones. Instead of fixed values of unitig coverage and fixes lengths for mate pair joins generated links, it takes intervals. Indeed, for real data it is difficult to provide exact coverage for small unitigs. The insert size of mate pairs which is at the origin of the distance weight of links is always a confidence interval. For previous models the mean of the upper and lower limits was used.

2.2 Input data features and complexity inspection

Transforming the input data file into the graph solved by the GST has been carried out by the Python interface to the Graphviz graph layout and visualization package, pygraphviz. Graphviz provides quick methods to access the features of the graph. Features and complexity of the input graph was evaluated with the underlying wish to understand why certain instances were wrongly scaffolded. Transformation and inspection of the input data was implemented in the *graph_inspector.py* script.

2.3 Benchmarking

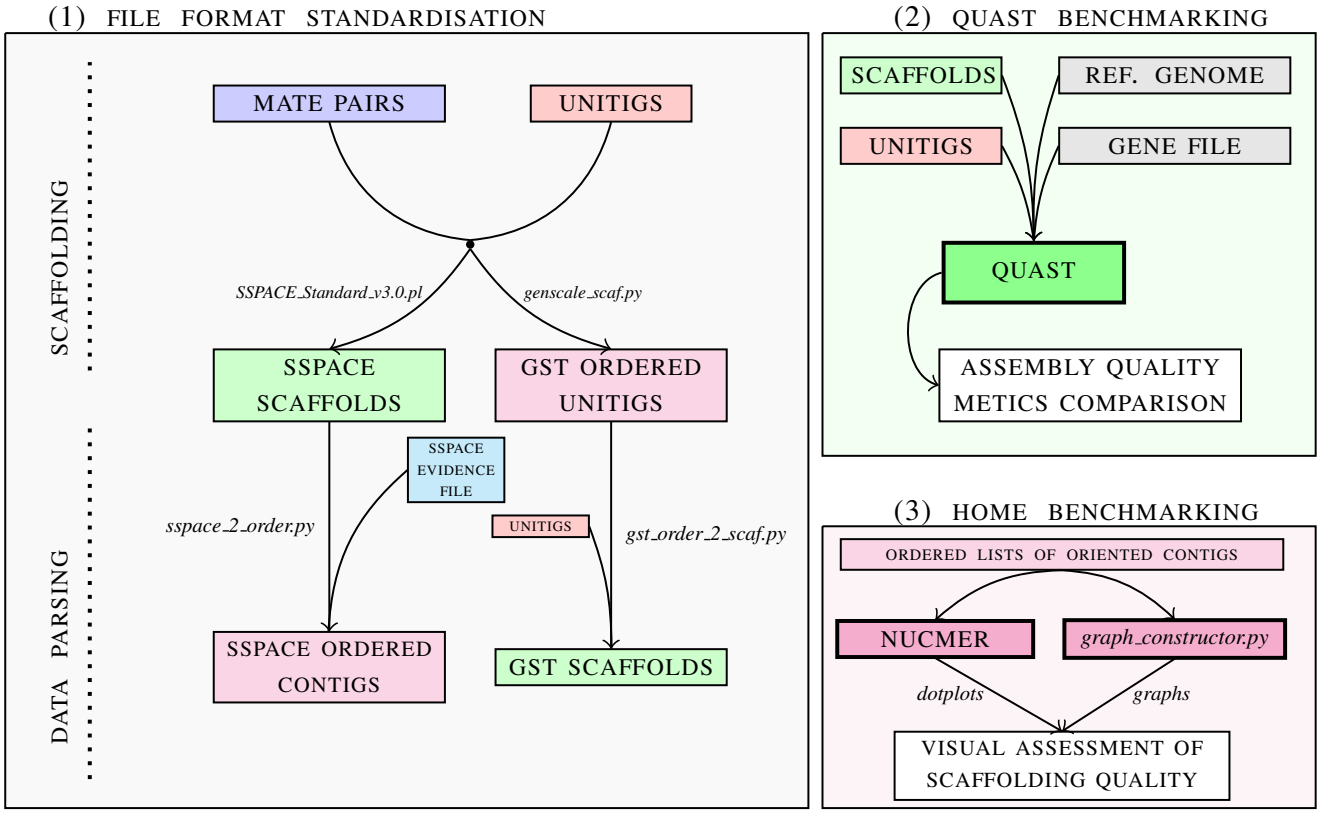
Comparing the performance of the GST can be done on several levels. As the data is artificially generated from an already known genome, the best solution is already known. The GST solutions can be compared between themselves, with the expected solution and with solutions provided by other published tools. The solution formats are not the same and automated methods of comparing were set up to extract and compare scaffoldings.

2.3.1 Comparison with the reference solution

The expected solution provided by the mapping of unitigs of the reference genome is the one GST aim to obtain in the form of an uninterrupted and accurately gapped scaffold. However in some cases there is not a single solution possible. In real life several versions of the genome can co-exist, so the GST should aim at providing all valid solutions. In our artificial datasets, only one version of the genome is used, however in the case of the chloroplastic genome the several valid solution configuration arises. The Small Single Copy and the Long Single Copy have both extremities next to the same extremities of the Inverted Repeat (see figure 6). In absence of strong mate-pair information overthrowing the overlapping information of these extremities, the SSC and the LSC can be oriented in forward or reverse, creating a total of 4 valid combination. Of course, the genomic structure can be much more complex than that presented in figure 6, providing linking information that would corroborate an orientation over another. The fact is, a valid solution is not an exact copy of the expected solution. Because it is not known in advance which contigs form the IR or the genomic chloroplast and because in *de novo* assembly no assumption of the genomic structure shall be made at all, these alternative solutions are very hard to detect automatically.
The detection of exact copies of the expected solution has been implemented in the *graph_comparator.py* script. It is a simple list comparison after the retrieval of GST solution in the form of an ordered list of oriented unitigs from the GST output files. Distances between unitigs has been ignored for this comparison. Sub-optimal solutions are also detected. This detection effort has been made especially for the distance based model which provides a single solution which sometimes possesses only one or two errors of orientation. However misplacement of a large set of contigs can only be detected visually with the graph provided by the *graph_generator.py* script. This kind of error did not occur many times in the instances tested as misplacement of one or several contigs often leads to more errors.

2.3.2 Comparison with published tools: benchmarking strategy

Draw the benchmarking in form of a workflow with tikz, explain how quast works and the parameters used, explain nucmer and parameters used, talk about data format converting scripts (list of ordered unitigs $< - >$ fasta sequence), graph_generator



(1) SSPACE provides a *.fasta* file with built scaffolds. This file does not track the unitig information: it is not known which unitig participates in which scaffold. SSPACE also provides a file called *scaffolds.evidence* which is used by the sspace_2_order.py script to retrieve the unitig composition of SSPACE scaffolds and built an ordered list of oriented contigs for each scaffold. The GST does not compute .fasta sequences of unitigs. The scaffolding final result is an ordered list of unitigs. gst_order_2_scaf.py builts sequence scaffolds from the ordered list by appending the unitig sequences. If distances between unitigs are not available, the k-mer size with which the unitigs were built is used.
(2) The reference genome and the gene file are downloaded from ncbi. They provide additional metrics for assembly comparison (number of found genes, percentage of genome assembled, N50 . . .). Assembly quality metrics are computed by QUAST for all the genomes, scaffolding solutions or unitigs provided in .fasta format.
(3) Further inspection of scaffolding solutions can be done thanks to nucmer alignment tool from the MUMmer software. nucmer dotplots scaffolding solutions against the reference genome so challenging regions can be detected. The graph_constructor.py script visualizes the unitigs which were not or wrongly assembled.

Figure 9: Benchmarking workflow

3 Results

3.1 Comparison between the data sets

Results presented in table 1 are those of the complete model-like graphs analyzed by the *graph_inspector.py*.

organism	G type	G size	#nodes	#edges	node degree	G density	diameter	#periphery nodes	radius	#central nodes
agrostis	chpl.	size	22	98	min 4 , max 13 , avg 8	0.212	6	1	4	6
acineto	bacter	size	924	7984	min 0 , max 114 , avg 17	0.009	gnt:ipl	gnt:ipl	gnt:ipl	nt:ipl
acorus	chpl.	size	30	204	min 4 , max 26 , avg 13	0.234	5	13	4	17
atropa	chpl.	size	52	268	min 5 , max 16 , avg 10	0.101	8	12	6	6
cucumis	chpl.	size	194	1446	min 4 , max 50 , avg 14	0.038	11	30	8	6
eucalyptus	chpl.	size	8	16	min 4 , max 4 , avg 4	0.285	4	8	4	8
euglena	chpl.	size	296	11894	min 12 , max 176 , avg 80	0.136	gnt:ipl	gnt:ipl	gnt:ipl	gnt:ipl
lecomtella	chpl.	size	22	90	min 4 , max 13 , avg 8	0.194	6	3	4	5
oenothera	chpl.	size	172	3086	min 4 , max 70 , avg 35	0.104	14	1	9	47
pinus	chpl.	size	122	628	min 4 , max 29 , avg 10	0.042	13	7	7	1
rice	chpl.	size	8	16	min 4 , max 4 , avg 4	0.285	4	8	4	8
sacchar.	chr3	size	370	4416	min 4 , max 113 , avg 23	0.032	13	1	6	37
wolbachia	bacter	size	-	-	min -, max -, avg -					

Table 1: Graph complexity

DIFFERENCES WITH THE EXPECTED SOLUTION

3.2 Comparison of GST solutions with the expected solution

organism	expected solution found	time	partial solution found	time	note
agrostis	wpm	0.5s	-	-	1 valid solution among 4, sub-optimal found with dist and flow
acineto	-	-	wpm	0.6s	ordering big unitigs correctly
acorus	wpm	0.5s	-	-	1 valid solution among 2, also found with dist and flow
atropa	wpm	1s	-	-	1 valid solution among 2, also found with flow
cucumis	wpm	35s	-	-	several valid solution among 18
eucalyptus	wpm	0.2s	-	-	1 valid solution among 2, also found with dist and flow
euglena	-	-	-	-	instance never fully or partially solved
lecomtella	wpm	-0.6s	-	-	1 valid solution among 4, sub-optimal found with dist
oenothera	wpm	163s	-	-	1 valid solution among 4
pinus	-	-	flow	0.2	instance never solved with wpm or dist, good frame found with flow
rice	flow	0.1s	-	-	also found with wpm and dist
sacchar.	-	-	-	-	instance never fully or partially solved
wolbachia	-	-	flow	0.4s	built frame with large unitigs resulting in 11 disconnected scaffolds

Table 2: Genscale scaffolders solutions

3.3 Comparison between best GST solution and SSPACE

3.3.1 QUAST and comparison function

Find similar table for rice, pinus and wolbachia in Annexes.

Assembly	unitigs	wpm_sol1.fsa	wpm_sol2.fsa	wpm_sol3.fsa	wpm_sol4.fsa	dist_sol	sspace_sol	ref_genome
# contigs (≥ 0 bp)	6	**1**	**1**	**1**	**1**	2	3	**1**
# contigs (≥ 1000 bp)	5	**1**	**1**	**1**	**1**	**1**	2	**1**
Total length (≥ 0 bp)	115327	136584	136584	136584	136584	**137641**	115344	136584
Total length (≥ 1000 bp)	115167	136584	136584	136584	136584	**137145**	115184	136584
# contigs	5	**1**	**1**	**1**	**1**	**1**	2	**1**
Largest contig	56143	136584	136584	136584	136584	**137145**	102256	136584
Total length	115167	136584	136584	136584	136584	**137145**	115184	136584
Reference length	136584	136584	136584	136584	136584	136584	136584	136584
GC (%)	37.39	38.45	38.45	38.45	38.45	38.45	37.40	38.45
Reference GC (%)	38.45	38.45	38.45	38.45	38.45	38.45	38.45	38.45
N50	24519	136584	136584	136584	136584	**137145**	102256	136584
NG50	24519	136584	136584	136584	136584	**137145**	102256	136584
N75	19351	136584	136584	136584	136584	**137145**	102256	136584
NG75	12878	136584	136584	136584	136584	**137145**	12928	136584
L50	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**
LG50	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**
L75	3	**1**	**1**	**1**	**1**	**1**	**1**	**1**
LG75	4	**1**	**1**	**1**	**1**	**1**	2	**1**
# misassemblies	**0**	**0**	2	1	1	2	1	**0**
# misassembled contigs	**0**	**0**	1	1	1	1	1	**0**
Misassembled contigs length	**0**	**0**	136584	136584	136584	137145	102256	**0**
# local misassemblies	**0**	**0**	**0**	**0**	**0**	1	**0**	**0**
# unaligned contigs	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part
Unaligned length	0	0	0	0	0	0	0	0
Genome fraction (%)	84.219	**100.000**	**100.000**	98.657	98.657	91.522	84.590	**100.000**
Duplication ratio	1.001	1.000	1.155	1.014	1.014	1.097	**0.997**	1.000
# N's per 100 kbp	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	218.75	12.15	**0.00**
# mismatches per 100 kbp	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
# indels per 100 kbp	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	5.60	0.87	**0.00**
# genes	110 + 4 part	132 + 1 part	**133 + 0 part**	131 + 0 part	131 + 0 part	122 + 1 part	114 + 2 part	**133 + 0 part**
Largest alignment	56143	**136584**	81612	82913	134750	102364	81140	**136584**
NA50	24519	**136584**	81612	82913	134750	102364	81140	**136584**
NGA50	24519	**136584**	81612	82913	134750	102364	81140	**136584**
NA75	19351	**136584**	53138	53671	134750	22817	21116	**136584**
NGA75	12878	**136584**	53138	53671	134750	22817	12928	**136584**
LA50	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**
LGA50	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**
LA75	3	**1**	2	2	**1**	2	2	**1**
LGA75	4	**1**	2	2	**1**	2	3	**1**

Table 3: All statistics are based on contigs of size ≥ 50 bp, unless otherwise noted (e.g., "# contigs (≥ 0 bp)" and "Total length (≥ 0 bp)" include all contigs).

Assembly	unitigs	flow_step1_sol	wpm_sol1.fsa	sspace_sol	ref_genome
# contigs (≥ 0 bp)	33	2	**1**	21	**1**
# contigs (≥ 1000 bp)	10	**1**	**1**	**1**	**1**
Total length (≥ 0 bp)	118629	116214	116866	**119654**	116866
Total length (≥ 1000 bp)	115186	115838	116866	**117042**	116866
# contigs	11	**1**	**1**	**1**	**1**
Largest contig	20834	115838	116866	**117042**	116866
Total length	115888	115838	116866	**117042**	116866
Reference length	116866	116866	116866	116866	116866
GC (%)	38.77	38.86	38.80	38.79	38.80
Reference GC (%)	38.80	38.80	38.80	38.80	38.80
N50	14862	115838	116866	**117042**	116866
NG50	14862	115838	116866	**117042**	116866
N75	10041	115838	116866	**117042**	116866
NG75	10041	115838	116866	**117042**	116866
L50	4	**1**	**1**	**1**	**1**
LG50	4	**1**	**1**	**1**	**1**
L75	6	**1**	**1**	**1**	**1**
LG75	6	**1**	**1**	**1**	**1**
# misassemblies	**0**	1	11	2	**0**
# misassembled contigs	**0**	1	1	1	**0**
Misassembled contigs length	**0**	115838	116866	117042	**0**
# local misassemblies	**0**	4	4	5	**0**
# unaligned contigs	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part
Unaligned length	0	0	0	0	0
Genome fraction (%)	99.079	92.153	99.471	99.254	**100.000**
Duplication ratio	1.001	1.076	1.005	1.010	**1.000**
# N's per 100 kbp	**0.00**	6997.70	**0.00**	757.85	**0.00**
# mismatches per 100 kbp	**0.00**	**0.00**	5.16	0.86	**0.00**
# indels per 100 kbp	2.59	1.86	8.60	2.59	**0.00**
# genes	261 + 6 part	249 + 4 part	262 + 7 part	262 + 6 part	**270 + 0 part**
Largest alignment	20834	96535	28559	86569	**116866**
NA50	14862	96535	20762	86569	**116866**
NGA50	14862	96535	20762	86569	**116866**
NA75	10041	96535	19567	29474	**116866**
NGA75	10041	96535	19567	29474	**116866**
LA50	4	**1**	3	**1**	**1**
LGA50	4	**1**	3	**1**	**1**
LA75	6	**1**	4	2	**1**
LGA75	6	**1**	4	2	**1**

Table 4: All statistics are based on contigs of size ≥ 50 bp, unless otherwise noted (e.g., "# contigs (≥ 0 bp)" and "Total length (≥ 0 bp)" include all contigs).

3.3.2 Visualization with mummer and graph_generator.py

3.3.2.1 graph_generator.py visualization

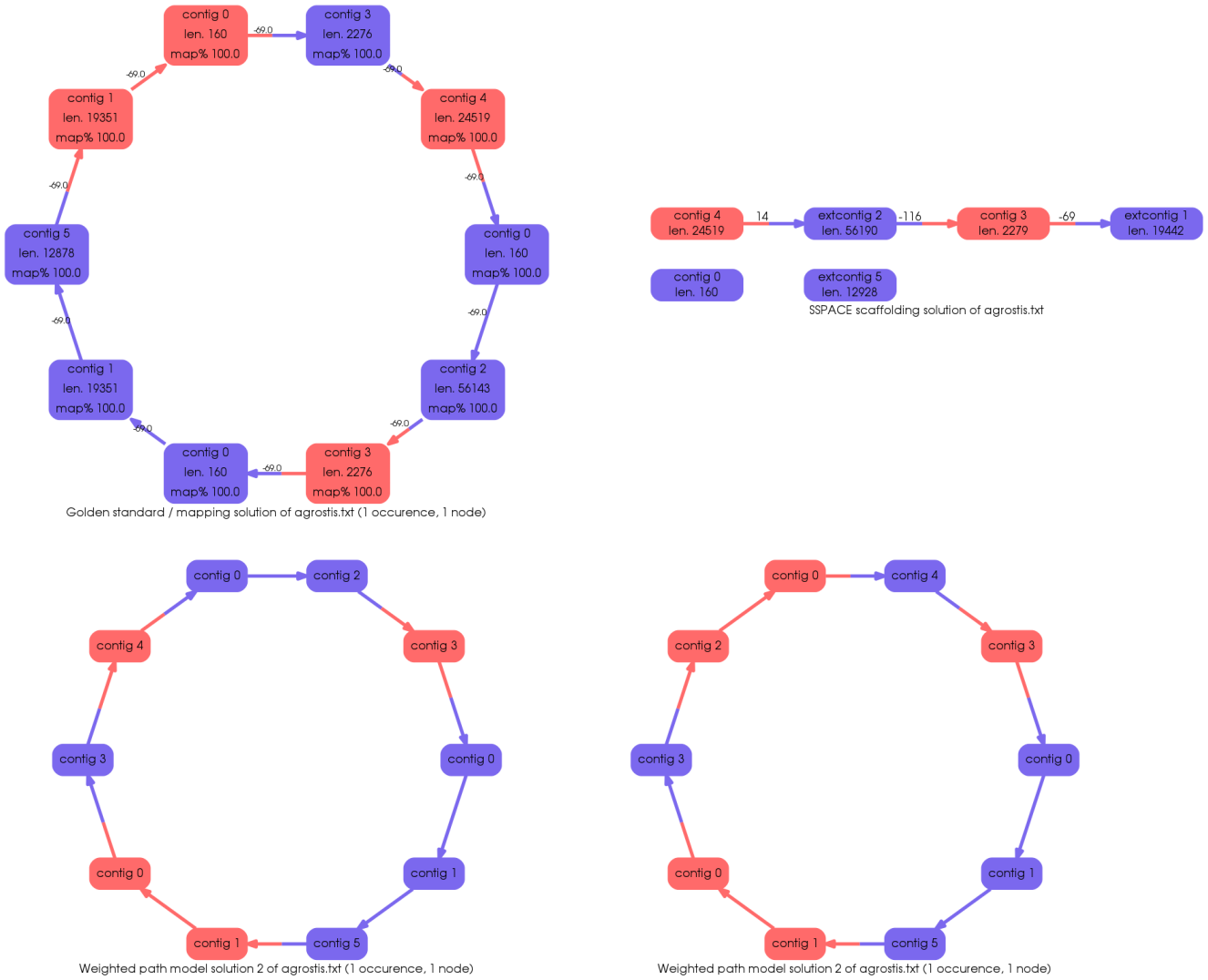


Figure 10: Expected solution and scaffolding solutions of weighted-path model and SSPACE scaffolders

3.3.2.2 Dotplots with mummer

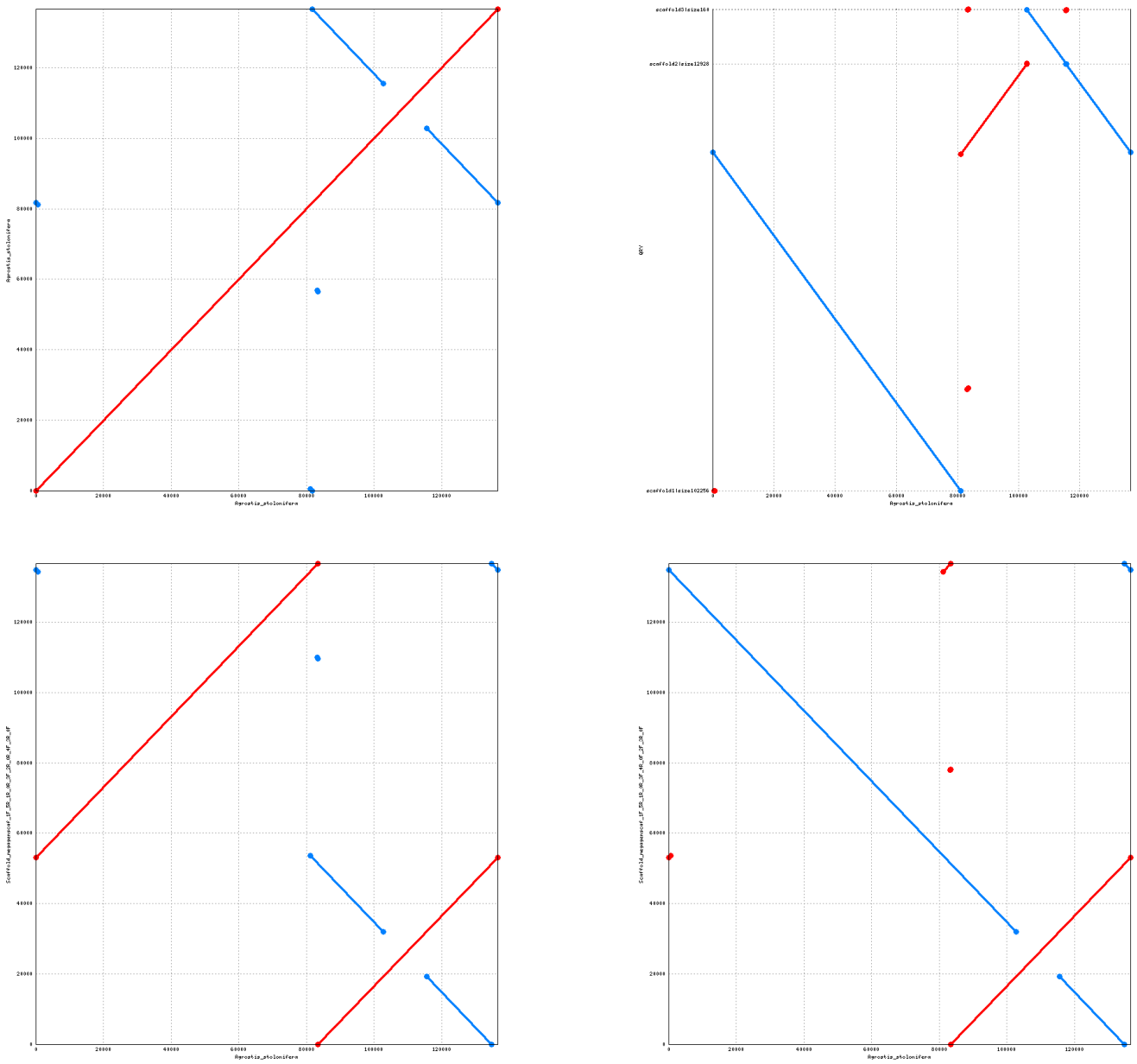


Figure 11: Control dotplot and dotplots of weighted path model and SSPACE scaffolding solution with the reference genome

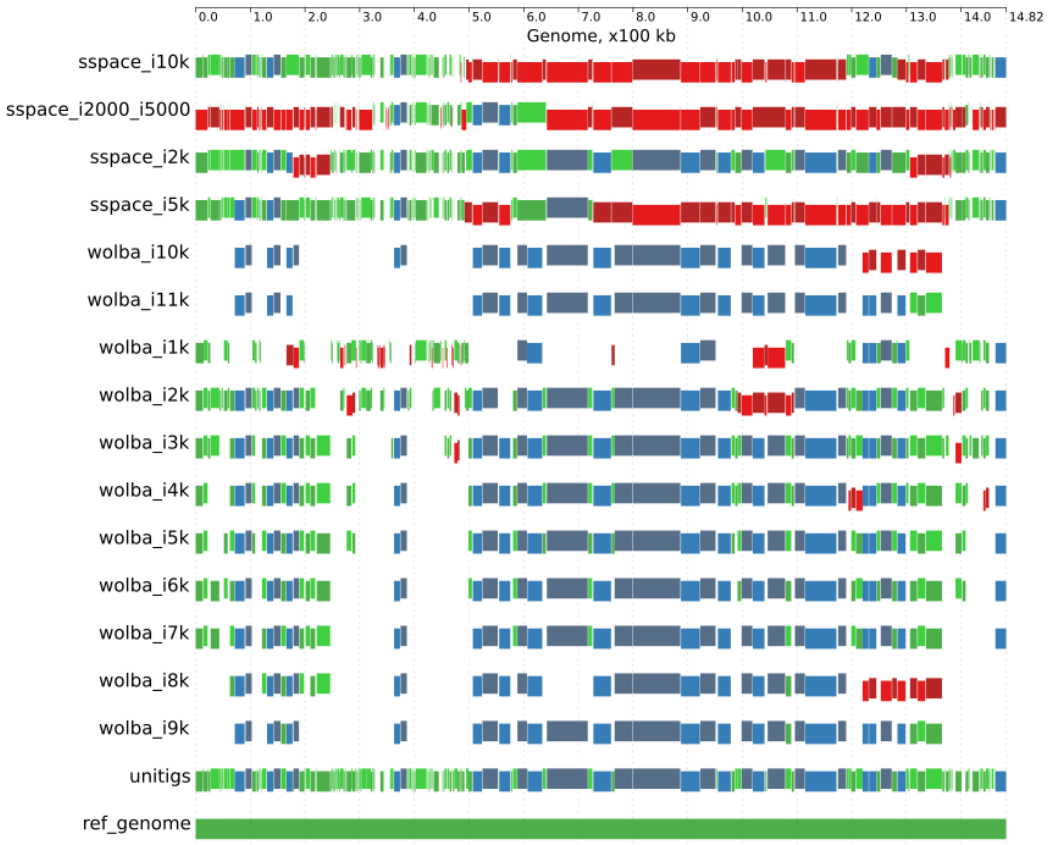
3.4 Partially solved instances

The bacterial genomes are partially assembled by the flow model, which processes big mate-pair connected unitigs first. The wolbachia example highlights the importance of correctly choosing the mate pairs' insert size. The results presented in table 5 show a drastic improvement of the total assembled length when changing the mate paired insert size from 1000bp to 2000bp. This length gradually decreases as the insert size increases (same with the genome size metric). The largest scaffold is also assembled with the 2000bp library (over 2000bp whereas all other libraries yield scaffolds $<$ 500kb). One proposed explanation is the repeats' sequence size of *Wolbachia Endosymbiont*. As previously seen in section 7 page 10 section 2.1.2 on page 9 Features of the assembled genomes, the bacterial genome has repeats with sizes mainly between 500bp and 1kbp. These repeats are partially solved during the unitig building step. The issue comes from genomically close repeats bigger than 1.2kbp. There are 25 repeats of this type, among them 3 bigger than 5kbp. The location of these repeats coincide with regions of major scaffolding problems, where unitigs are not scaffolded or so badly scaffolded that they do not map on the reference genome. To see those location see figure 12 and table 6. The insert size must be big enough to overlap all repeats however increasing the insert size too much impacts on the scaffolders' ability to precisely join regions containing smaller overlaps. Using multiple libraries would be a solution however this strategy didn't result in encouraging scaffoldings with SSPACE (lib i2000 and i5000 used simultaneously).

Assembly	unitigs	wolba_i10k	wolba_i11k	wolba_i1k	wolba_i2k	wolba_i3k	wolba_i4k	wolba_i5k	wolba_i6k	wolba_i7k	wolba_i8k	wolba_i9k	ref_genome
# scaffolds ($\geq$ 0 bp)	444	9	7	37	28	19	13	13	13	11	9	9	**1**
# scaffolds ($\geq$ 1000 bp)	138	9	7	37	28	19	13	13	13	11	9	9	**1**
Total length ($\geq$ 0 bp)	1364357	871378	847704	615455	1214243	1197652	1121566	1101921	1084716	1045627	869142	879828	**1482355**
Total length ($\geq$ 1000 bp)	1290990	871378	847704	615455	1214243	1197652	1121566	1101921	1084716	1045627	869142	879828	**1482355**
# scaffolds	444	9	7	37	28	19	13	13	13	11	9	9	**1**
Largest scaffolds	87315	387284	387518	63122	222630	368628	368594	483412	211466	334472	249957	249957	**1482355**
Total length	1364357	871378	847704	615455	1214243	1197652	1121566	1101921	1084716	1045627	869142	879828	**1482355**
Reference length	1482355	1482355	1482355	1482355	1482355	1482355	1482355	1482355	1482355	1482355	1482355	1482355	1482355
GC (%)	34.01	33.92	33.94	34.08	33.89	33.95	33.96	33.96	33.91	33.93	33.95	33.94	34.19
Reference GC (%)	34.19	34.19	34.19	34.19	34.19	34.19	34.19	34.19	34.19	34.19	34.19	34.19	34.19
N50	17458	142935	143187	21300	68630	75831	142496	122998	116953	189499	189823	189972	**1482355**
NG50	14993	68085	68090	-	59359	69791	122971	77779	93415	122899	45001	57433	**1482355**
N75	7665	79427	79468	12604	44181	48864	75838	58047	77742	62360	75545	68114	**1482355**
NG75	5696	-	-	-	14316	22037	15579	-	-	-	-	-	**1482355**
L50	22	2	2	7	6	4	3	2	4	2	2	2	**1**
LG50	25	5	5	-	9	6	4	4	6	4	6	6	1
L75	52	4	4	17	12	8	5	5	7	5	4	5	**1**
LG75	65	-	-	-	19	14	13	-	-	-	-	-	1
# misassemblies	**0**	1	**0**	12	3	1	1	**0**	**0**	**0**	1	**0**	**0**
# misassembled scaffolds	**0**	1	**0**	7	3	1	1	**0**	**0**	**0**	1	**0**	**0**
Misassembled scaffolds length	**0**	142935	**0**	139698	143728	22037	35610	**0**	**0**	**0**	142540	**0**	**0**
# local misassemblies	**0**	23	23	45	66	60	52	47	42	38	30	26	**0**
# unaligned scaffolds	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part	0 + 0 part
Unaligned length	0	0	0	0	0	0	0	0	0	0	0	0	0
Genome fraction (%)	90.341	53.726	51.551	40.034	78.587	76.826	71.970	70.313	68.955	66.047	54.424	54.992	**100.000**
Duplication ratio	1.019	1.094	1.109	1.040	1.042	1.052	1.051	1.057	1.061	1.068	1.077	1.079	**1.000**
# N's per 100 kbp	**0.00**	8603.27	9854.62	1808.26	3969.22	4880.30	4859.72	5402.66	5764.55	6364.03	7177.65	7348.94	**0.00**
# mismatches per 100 kbp	**0.00**	**0.00**	**0.00**	42.97	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
# indels per 100 kbp	**0.00**	**0.00**	**0.00**	4.72	0.09	0.09	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Largest alignment	87315	356813	356813	62424	219709	359164	359164	459850	205359	319611	237693	237693	**1482355**
NA50	17458	89651	115294	16288	59479	71508	134513	111512	111512	116821	99293	119120	**1482355**
NGA50	14993	23733	29168	-	51212	64638	71508	56028	74077	56028	33486	44273	**1482355**
NA75	7665	65191	65191	10269	35490	42401	47594	49146	56028	55817	44360	65191	**1482355**
NGA75	5696	-	-	-	10269	10269	-	-	-	-	-	-	**1482355**
LA50	22	2	2	9	7	4	3	2	4	3	3	3	**1**
LGA50	25	8	7	-	9	6	5	5	7	5	8	7	1
LA75	52	5	5	20	13	9	6	6	8	6	6	5	**1**
LGA75	65	-	-	-	24	18	-	-	-	-	-	-	1

All statistics are based on contigs of size $\geq$ 50 bp, unless otherwise noted (all scaffolds are).

Table 5: Scaffolding solutions of the flow model step 1 with mate-pair libraries of different insert sizes compared to the reference genome and the initial unitig sample.



This plot shows alignment of contigs to the reference genome and positions of misassemblies in these scaffolds. Correctly aligned big ($>$ 10kb) scaffolds are blue if the boundaries agree, and green if the boundaries don't agree. Scaffolds containing an important amount of misassemblies are red. Here, SSPACE solution with the 2kbp insert size library shows the less misassembled regions with the most reference coverage. However the solution is broken up in an enormous amount of scaffolds (336), most of them being just the unsucessfully attached small unitigs. SSPACE also fails to produce any valid scaffold for the genomic regions containing the large $>$5000 repeats (see their exact coordinates in table 6).

Figure 12: Alignment of scaffolding solutions on the reference genome of Wolbachia endosymbiont

[S1]	[E1]	[S2]	[E2]	[LEN 1]	[LEN 2]	[% IDY]
1	1482355	1	1482355	1482355	1482355	100.00
118235	119770	14639	13104	1536	1536	99.67
951662	953006	62728	61384	1345	1345	100.00
13104	14639	119770	118235	1536	1536	99.67
295130	296467	156339	155002	1338	1338	100.00
1044418	1046269	178963	177112	1852	1852	100.00
295131	296471	201307	199967	1341	1341	99.93
346054	354494	259128	250682	8441	8447	99.56
1040287	1042282	286820	284825	1996	1996	100.00
1370666	1372235	293326	291757	1570	1570	99.87
886392	887730	296467	295129	1339	1339	100.00
1459215	1461421	296650	294476	2207	2175	98.50
353564	355681	325239	323122	2118	2118	97.54
346054	351130	332648	327572	5077	5077	99.94
250682	259128	354494	346054	8447	8441	99.56
295130	296467	426552	425215	1338	1338	100.00
353564	355681	449855	447738	2118	2118	97.54
951661	953011	506502	505152	1351	1351	99.93
1305338	1306686	553913	552565	1349	1349	99.93
295129	296467	887730	886392	1339	1339	100.00
505152	506502	953011	951661	1351	1351	99.93
284825	286820	1042282	1040287	1996	1996	100.00
177112	178963	1046269	1044418	1852	1852	100.00
1334495	1335840	1306687	1305342	1346	1346	100.00
1305342	1306686	1320819	1319475	1345	1345	100.00
1305342	1306687	1335840	1334495	1346	1346	100.00
291757	293326	1372235	1370666	1570	1570	99.87
294476	296650	1461421	1459215	2175	2207	98.50

Table 6: Coordinates and length of repeats in the *Wolbachia endosymbiont* genome

4 Discussion

5 Conclusion

6 References

References

[1] Hunt, M., Newbold, C., Berriman, M. & Otto, T. D. A comprehensive evaluation of assembly scaffolding tools **15**, R42.

[2] Earl, D. *et al.* Assemblathon 1: A competitive assessment of de novo short read assembly methods **21**, 2224–2241. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3227110/.

[3] Bradnam, K. R. *et al.* Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species **2**, 10. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3844414/.

[4] Pop, M., Kosack, D. S. & Salzberg, S. L. Hierarchical scaffolding with bambus **14**, 149–159. URL http://genome.cshlp.org/content/14/1/149.

[5] Treangen, T. J. *et al.* MetAMOS: a modular and open source metagenomic assembly and analysis pipeline **14**, R2. URL http://genomebiology.com/2013/14/1/R2/abstract.

[6] Zerbino, D. R. & Birney, E. Velvet: Algorithms for de novo short read assembly using de bruijn graphs **18**, 821–829. URL http://genome.cshlp.org/content/18/5/821.

[7] Hunt, M., Newbold, C., Berriman, M. & Otto, T. D. A comprehensive evaluation of assembly scaffolding tools **15**, R42. URL http://genomebiology.com/2014/15/3/R42/abstract.

[8] Huson, D. H., Reinert, K. & Myers, E. W. The greedy path-merging algorithm for contig scaffolding **49**, 603–615. URL http://doi.acm.org/10.1145/585265.585267.

[9] Li, H. & Durbin, R. Fast and accurate short read alignment with burrows-wheeler transform **25**, 1754–1760.

[10] Langmead, B. & Salzberg, S. L. Fast gapped-read alignment with bowtie 2 **9**, 357–359. URL http://www.nature.com/nmeth/journal/v9/n4/full/nmeth.1923.html.

[11] Langmead, B., Trapnell, C., Pop, M. & Salzberg, S. L. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome **10**, R25. URL http://genomebiology.com/2009/10/3/R25/abstract.

[12] Lander, E. S. *et al.* Initial sequencing and analysis of the human genome **409**, 860–921.

[13] Venter, J. C. *et al.* The sequence of the human genome **291**, 1304–1351.

7 Annexes