CONTENTS

[INTRODUCTION 2](#_Toc40068583)

[1. CHROMOSOME SCALE ASSEMBLY 3](#_Toc40068584)

[1.1. 3](#_Toc40068585)

[1.2. 3](#_Toc40068586)

[1.3. Jskfgfgs 3](#_Toc40068587)

[1.3.1. 3](#_Toc40068588)

[1.3.2. 3](#_Toc40068589)

[2. Mosquitos project. 4](#_Toc40068590)

[2.1. Materials and methods 4](#_Toc40068591)

[1.1.1. Main pipeline 4](#_Toc40068592)

[2.1.1. Data description 4](#_Toc40068593)

[2.1.2. Genome size estimation 6](#_Toc40068594)

[2.1.3. Nanopore reads contamination search 8](#_Toc40068595)

[2.1.4. Genome assemblers 9](#_Toc40068596)

[2.1.5. Wtdbg2 assembler 10](#_Toc40068597)

[2.1.6. Miniasm assembler 10](#_Toc40068598)

[2.1.7. Flye assembler 10](#_Toc40068599)

[2.1.8. CANU assembler 11](#_Toc40068600)

[2.1.9. Assemblies assessment 11](#_Toc40068601)

[APPENDIX A 14](#_Toc40068602)

INTRODUCTION

rfgdhds

1. CHROMOSOME-SCALE ASSEMBLY

In this chapter, I describe the main concepts of chromosome-scale assembly and define terminology that is used in my thesis.

1. Mosquitos project.
   1. Materials and methods
      1. Main pipeline
      2. Data description

Three types of data were used:

* Long reads from Oxford Nanopore sequencing
* Short reads from Illumina sequencing
* Hi-C Illumina reads

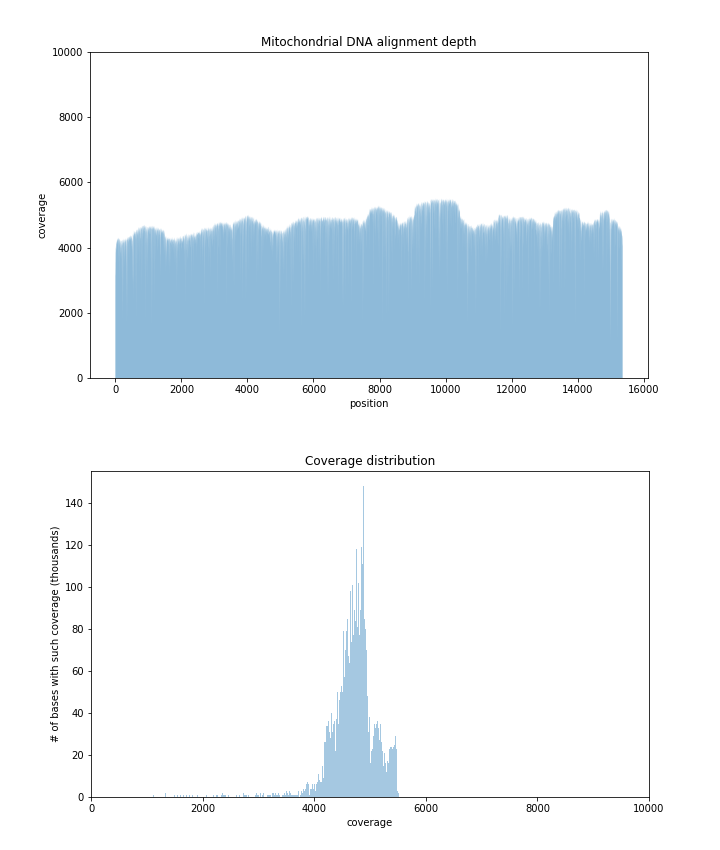
The main source of genomic data for both anopheline assemblies was Oxford Nanopore long reads. Nanopore sequencing data was obtained from our collaborators. For both species, it consisted of nanopore long reads in .fastq file format, raw nanopore signal data in separated files in .fast5 format and sequencing summary information in text format.

Quality control, basic analysis, and visualization of the long nanopore reads were done with Nanostat and Nanoplot from Nanopack software package (de2018nanopack). Main statistics. For *an. coluzzii* genome, these tools reported 3.3M reads of the total length 28Gbp. The read length N50 is 19Kbp, and the read median length and quality are around 4 Kbp and 10.3, respectively. For *an. arabiensis* genome, we have 5.0M reads of the total length 35Gbp. The read length N50 is 21Kbp and the read median length and quality are 2.2Kbp and 10.0, respectively. The detailed statistics reported by these tools can be found in supplementary table #. Reads length and quality distribution plots are shown for *an.coluzzii* in supplementary fig. # for *an.arabiensis* in supplementary fig. #.

From quality control we can see that *an.coluzzii* reads were initially trimmed by 7 quality but *an.arabiensis* reads were not. We decided not to trim *an.arabiensis* reads because assemblers that were used have their own algorithms to work with read quality.

I performed reference alignment to the *an.gembiae* genome to estimate average sequencing coverage of chromosomes. Alignments were done using minimap2 tool(citation). For *an.coluzzii* genome, the total number of aligned and unaligned reads equals 3.3M(99%) and 0.03M(1%), respectively. In the case of *an.arabiensis* genome, the 4.5M(89%) for 0.56M(11%). Minimap2 output .sam files were converted to bam sorted and then for each base in reference genome coverage was computed and stored in table file. The table file then was parsed with my python script and for each chromosome coverage statistics were computed. Also, I visualized chromosome coverage to understand what regions of reference chromosomes are not covered and built a coverage distribution histogram figures ## in supplementary.

The alignment statistics confirmed the 100x coverage for *an.coluzzii* genome and 114x coverage for *an.arabiensis* genome. Mitochondrial DNA was not excluded from sequencing libraries and as we can see on fig. # it has much more read coverage as chromosomal DNA.



* + - 1. Alignment depth and coverage distribution of mitochondrial DNA of *an.arabienses*

Nanopore reads were filtered out from contamination using Kraken2(citation) tool before assembly process. I tried to estimate genome sizes using k-mer analysis of nanopore reads but it cannot be estimated due to large error rate of nanopore sequencing. Contamination filtering and genome size estimation process will be described in following sections.

For polishing assemblies obtained from nanopore reads, we used the Illumina short paired-end data with NCBI SRX accession number SRX3832577 for *an.coluzzii* genome, and NCBI SRX accession numbers SRX084275, SRX084275, SRX084275, SRX111457, SRX111457, SRX111457, SRX111457, SRX200218 for *an.arabiensis* genome. The sequence quality control of the short pair-end reads was performed with FastQC (FastQC, RRID:SCR\_014583) (citation). FastQC showed that *an.coluzzii* reads have high per-base sequence quality (exceeding 32 on Phred scale) and no adapter contamination. It reported 122.3M reads of length in the range 36--200bp and the total length 22.8Gbp. For *an.arabiensis* genome, FastQC reported 260.6M reads of total length 53.2Gbp and average length 90bp. Based on the FastQC analysis, I filtered reads by the quality and minimum read length, and further trimmed TruSeq adapters from reads using fastp v0.20.0 (fastp: Chen et al., 2018). This resulted in filtering 14% of reads and leaving just 224.8M reads.

Hi-C data for genome scaffolding was obtained from our collaborators. Hi-C libraries preparation protocol for *an.coluzzii* used MboI restriction enzymes, Arima protocol was used for *an.arabiensis*. Hi-C Illumina short paired-end reads quality control was inspected with FastQC. For both mosquito genomes, FastQC showed high per base sequence quality (exceeding 30 on Phred scale), and detected contamination with Illumina TrueSeq adapters in 0.17% reads of *an.coluzzii* and in 1.5% reads of *an.arabiensis*. All such contaminated reads were filtered out in both read sets, resulting in 231.9M and 141.9M reads for *an.coluzzii* and *an.arabiensis*, respectively.

* + 1. Genome size estimation

Due to inability to use nanopore sequencing data for genome size estimation I used Illumina reads which have less error rate.

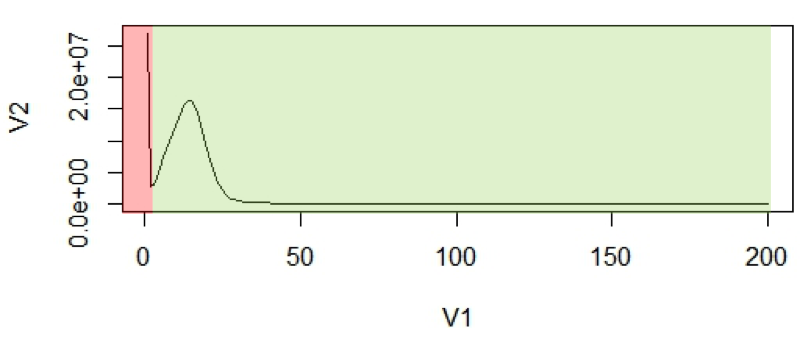
Approximate genome size can be calculated by counting k-mer frequency of the sequencing data.(citation) The k should be sufficiently large that most of the genome can be distinguished. For most eukaryotic genomes at least 17 are usually used.

To understand difficulties of such way of estimation we must look at k-mer distribution of a typical real-world genome. The main issue that is faced in a real-world genome sequencing projects is a non-uniform coverage of genome. This can be accounted to technical and biological variables, for example biased amplification of certain genomic regions during PCR and presence of repetitive sequences in genome.

The size of k-mers should be large enough allowing the k-mer to map uniquely to the genome (a concept used in designing primer/oligo length for PCR).

In the first step, k-mer frequency is calculated to determine the coverage of genome achieved during sequencing. There are software tools like Jellyfish that helps in finding the k-mer frequency in sequencing projects. The k-mer frequency follows a Poisson distribution, it can be treated like pseudo-normal around the mean coverage in histogram of k-mer counts.

Once the k-mer frequencies are calculated a histogram is plotted to visualize the distribution and to calculate mean coverage, see fig. #.



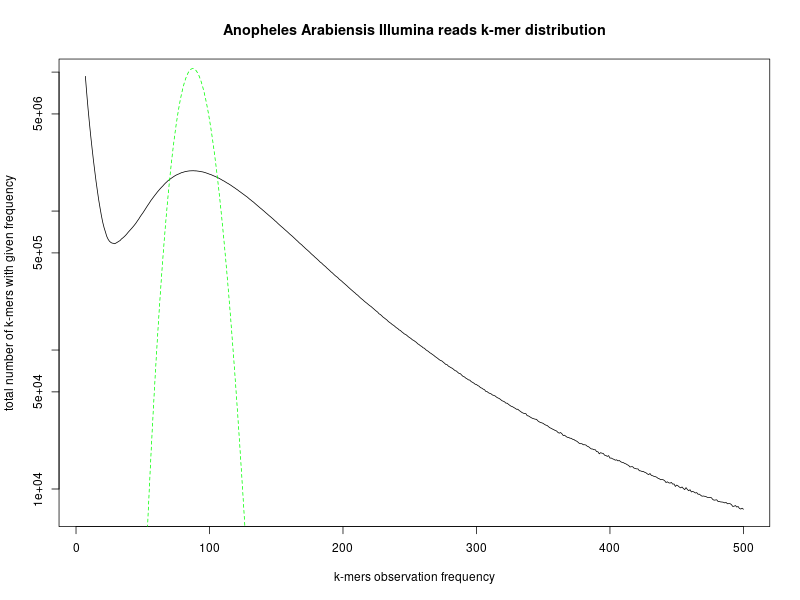
* + - 1. K-mer histogram example. The x-axis (V1), is the frequency or the number of times a given k-mer is observed. The y-axis (V2), is the total number of k-mers with a given frequency.

The first peak in red region is a result of rare and random sequencing errors in reads. These values can be trimmed to remove reads with sequencing errors from estimation process. With the assumption that k-mers are uniquely mapped to genome, they should be present only once in a genome sequence. Thus, their frequency will reflect the coverage of the genome. Mean coverage is used for calculating. The area under the curve will represent the total number of k-mers.

The genome size estimation will be:

To estimate single copy region size we must count not a total number of k-mers but only number of k-mers from this region or in other words we must calculate area under the bell shape only removing all k-mers with higher frequency.

The whole-genome size of *an.coluzzii* and *an.arabiensis* was estimated by the k-mer analysis for k=19 based on Illumina short pair-end reads. I computed the frequency distribution of 19-mers in all high-quality short reads using jellyfish. For *an.coluzzii* genome, the peak of the 19-mer distribution was at a depth of 54, and the whole-genome size was estimated as 301.3Mbp. The length of single genome regions was estimated as 204.1Mbp. For Illumina reads of *an.arabiensis* genome, the peak of the 19-mer distribution was at a depth of 88, and the genome size was estimated as 315.6 Mbp. The size of single genome regions was estimated to be 249.4Mbp. Histograms of k-mere distribution are shown in fig. ##.



* + - 1. Distribution of k-mers (k=19) in *an.arabiensis* genome. Green line is theoretical Poisson distribution. Y axis is log scaled.
    1. Nanopore reads contamination search

Before assembly all nanopore reads were filtered out from contaminants using taxonomic sequence classification system Kraken2 (citation).

There is another approach to filter contaminants as a step of genome validation after assembly (funestus citation). But when I obtained the first draft assembly of *an.coluzzii* genome that was assembled with CANU I have done contamination search with Kraken2 and after that checked contaminated contigs with NCBI Blast over NCBI nucleotide database base(citation). The results showed that there are chimeric contigs that have regions of contaminated DNA inside while other regions belong to mosquito species. To prevent information loss because of filtering contigs with mosquito DNA we decided to search contaminants on the reads stage. Nanopore reads are quite long for this process and results of the search are satisfying enough.

Kraken2 is a taxonomic sequence classifier that assigns taxonomic labels to DNA sequences. Kraken2 examines the k-mers within a query sequence and uses the information within those k-mers to query a database. That database maps k-mers to the lowest common ancestor (LCA) of all genomes known to contain a given k-mer.

I performed a contamination analysis with Kraken2 using a custom database, which includes all bacterial, archaeal, protozoan, viral RefSeq genomes, human genome, artificial contaminants and manually added mosquito genomes from Vectorbase. For An. Coluzzii genome, Kraken2 identified the origin of most reads (98.40%) as mosquito. For An. arabiensis genome, the mosquito origin was detected for 89.55% of reads, but for 4.8% of reads the origin remained unknown. We considered the remaining reads (which were primarily attributed to the bacterial origin) as contaminated and filtered them out. At the same time, we retained the reads of unknown origin for a downstream analysis because they may represent novel mosquito sequences.

Technically contamination search was divided into three stages. First two are described on tool github page. First stage is Kraken2 database building. There was an issue - to add some genome from .fasta file into database special taxonomy sign must be added into the name of each contig. That was done using python script. The second stage is assigning a taxonomy and the third is reads filtering according to Kraken2 output. Output of Kraken2 is table file with read names and taxonomy ids. I write another python script to parse output and filter contaminant reads.

* + 1. Genome assemblers

Genome assembly from nanopore sequencing data is an actively developing area. Recently, several tools for assembling nanopore reads were released. We decided to perform comprehensive comparison of these software on quality of assembly. To assess assembler performance, we choose several assemblers that were available at the time on nanopore reads for An. coluzzii genome, including wtdbg2 v1.1 (WTDBG, RRID:SCR\_017225) (citation), FLYE v2.4.1 (Flye, RRID:SCR\_017016) (citation), Miniasm v?? (Miniasm, RRID:SCR\_015880) (citation), and Canu v1.8 (Canu, RRID:SCR\_015880) (citation). For wtdbg2, optional polishing step using minimap2 with the same nanopore data was performed per the developers recommendation. After Canu assembling we have two assembly variants: contigs and unitigs assemblies.

* + 1. Wtdbg2 assembler

Wtdbg2 is a de novo sequence assembler for long noisy reads produced by PacBio or Oxford Nanopore Technologies (ONT). It assembles raw reads without error correction and then builds the consensus from intermediate assembly output.

During assembly, wtdbg2 chops reads into 1024bp segments, merges similar segments into a vertex and connects vertices based on the segment adjacency on reads. The resulting graph is called fuzzy Bruijn graph (FBG). It is akin to De Bruijn graph but permits mismatches/gaps and keeps read paths when collapsing k-mers. The use of FBG distinguishes wtdbg2 from the majority of long-read assemblers.

Wtdbg2 has two key components: an assembler wtdbg2 and a consenser wtpoa-cns. Executable wtdbg2 assembles raw reads and generates the contig layout and edge sequences in a file "prefix.ctg.lay.gz". Executable wtpoa-cns takes this file as input and produces the final consensus in .fasta.

I used additional polishing step which is based on consensus with the same nanopore data minimap2 alignment. This step is recommended by developers if you don’t use other sources of sequencing data for further polishing.

* + 1. Miniasm assembler

Miniasm is a very fast OLC-based (overlap layout consensus) de novo assembler for noisy long reads. It takes all-vs-all read self-mappings (typically by minimap2) as input and outputs an assembly graph in the GFA format. Different from mainstream assemblers, miniasm does not have a consensus step. It simply concatenates pieces of read sequences to generate the final unitig sequences. Thus the per-base error rate is similar to the raw input reads.

So far miniasm is in early development stage.

* + 1. Flye assembler

Flye is a de novo assembler for single molecule sequencing reads, such as those produced by PacBio and Oxford Nanopore Technologies. It is designed for a wide range of datasets, from small bacterial projects to large mammalian-scale assemblies. It was written by Mikhail Kolmogorov and can be installed from source or from bioconda repository. I can note good support for this tool. During Flye usage there was issue with memory consumption that was promptly fixed by Mikhail.

Flye is using repeat graph as a core data structure. In difference to de Bruijn graphs (which require exact k-mer matches), repeat graphs are built using approximate sequence matches, and can tolerate higher noise of SMS reads.

The edges of repeat graph represent genomic sequence, and nodes define the junctions. Each edges is classified into unique or repetitive. The genome traverses the graph (in an unknown way), so as each unique edge appears exactly once in this traversal. Repeat graphs reveal the repeat structure of the genome, which helps to reconstruct an optimal assembly.

* + 1. CANU assembler

CANU is a new single-molecule sequence assembler that improves upon and supersedes the now unsupported Celera Assembler.

CANU is a fork of the Celera Assembler, designed for high-noise single-molecule sequencing (such as the PacBio RS II/Sequel or Oxford Nanopore MinION/GridION).

CANU is a hierarchical assembly pipeline which runs in four steps:

Detect overlaps in high-noise sequences using MHAP

Generate corrected sequence consensus

Trim corrected sequences

Assemble trimmed corrected sequences

It can be easily installed from sources but only in single node mode. To run CANU in grid mode it must be configured with SLURM system requirements. That was done by cluster administrator and CANU was installed as SLURM module. Also, it requires specific JVM version that may be an issue.

* + 1. Assemblies assessment

Assembly assessment and ranking by best quality of assembly was done with two state-of-art tools QUAST-LG (citation) and BUSCO genes (citation). For QUAST-LG, we used *an.gambiae* as a reference genome (AgamP4)

QUAST-LG is an extension of QUAST intended for evaluating large-scale genome assemblies (up to mammalian-size). It is included in the QUAST package starting from version 5.0.0. QUAST was written in Center of Algorithmic Biology Saint-Petersburg.

QUAST default pipeline utilizes Minimap2. Also, it uses bedtools for calculating raw and physical read coverage, which is shown in Icarus contig alignment viewer. QUAST-LG introduced modules requiring KMC and Red. That all means that QUAST has a lot of dependences and the best choice is to install it into independent python environment like full package.

Main metrics that QUAST -lg outputs for assembly:

Metrics based only on contigs:

* Number of large contigs (i.e., longer than 500 bp) and total length of them;
* Length of the largest contig;
* N50 (length of a contig, such that all the contigs of at least the same length together cover at least 50% of the assembly) and other similar metrics;
* Numbers of misassembles of different kinds (inversions, relocations, translocations;
* Number and total length of unaligned contigs.
* Numbers of mismatches and indels, over the assembly and per 100 kb.
* Genome fraction %, assembled part of the reference.
* Duplication ratio, the total number of aligned bases in the assembly divided by the total number of those in the reference. If the assembly contains many contigs that cover the same regions, its duplication ratio will significantly exceed. This occurs due to multiple reasons, including overestimating repeat multiplicities and overlaps between contigs.
* NGA50, a reference-aware version of N50 metric. It is calculated using aligned blocks instead of contigs. Such blocks are obtained after removing unaligned regions, and then splitting contigs at misassembly breakpoints. Thus, NGA50 is the length of a block, such that all the blocks of at least the same length together cover at least 50% of the reference.

BUSCO completeness assessments employ sets of Benchmarking Universal Single-Copy Orthologs from OrthoDB (www.orthodb.org) to provide quantitative measures of the completeness of genome assemblies, annotated gene sets, and transcriptomes in terms of expected gene content. Genes that make up the BUSCO sets for each major lineage are selected from orthologous groups with genes present as single-copy orthologs in at least 90% of the species. While allowing for rare gene duplications or losses, this establishes an evolutionarily-informed expectation that these genes should be found as single-copy orthologs in any newly-sequenced genome. The evolutionary expectation means that if the BUSCOs cannot be identified in a genome assembly or annotated gene set, it is possible that the sequencing and/or assembly and/or annotation approaches have failed to capture the complete expected gene content.

The assessment tool implements a computational pipeline to identify and classify BUSCO group matches from genome assemblies, annotated gene sets, or transcriptomes, using HMMER hidden Markov models and de novo gene prediction with Augustus. Running the assessment tool requires working installations of Python, HMMER, Blast+, and Augustus (genome assessment only). Genome assembly assessment first identifies candidate regions to be assessed with tBLASTn searches using BUSCO consensus sequences. Gene structures are then predicted using Augustus with BUSCO block profiles. These predicted genes, or all genes from an annotated gene set or transcriptome, are then assessed using HMMER and lineage specific BUSCO profiles to classify matches. The recovered matches are classified as ‘complete’ if their lengths are within the expectation of the BUSCO profile match lengths. If these are found more than once they are classified as ‘duplicated’. The matches that are only partially recovered are classified as ‘fragmented’, and BUSCO groups for which there are no matches that pass the tests of orthology are classified as ‘missing’.

At practice BUSCO is python script that consequently run two cycles of tBLASTn -> Augustus -> HMMER pipeline. In this case it has many dependences and it is hard to install it manually from sources. Installing whole as Conda environment from bioconda channel is very useful. But there are another issues: you must use correct orthologs database and configure Augustus relatively, for this project diptera database was used and Augustus configured to using *aedes aegipty* model. Another issue is that tBLASTn sometimes cant work in multithread mode and it must be run on single core.

QUAST-LG and BUSCO results for all draft assemblies are presented in supplementary tables # #. CANU has the best result, the second place is for Flye, third is wtdbg2 and the last is miniasm.

* + 1. Genome polishing

The assembly of long reads from Oxford Nanopore Technologies typically requires resource-intensive polishing to obtain high-quality assemblies. There are two commonly recommended polishing strategies for fixing frequent insertion and deletion errors in assemblies obtained from nanopore reads (citation). The first pipeline is to run Racon (Racon, RRID:SCR\_017642) (citation) several times using raw nanopore reads and then run Medaka (Medaka, RRID:SCR\_005857) (citation). The second strategy is to run Nanopolish (Nanopolish, RRID:SCR\_016157) (citation) using signal-level data measured by the nanopore sequencer. In both strategies, for obtaining better quality assemblies, it is recommended to run Pilon (Pilon, RRID:SCR\_014731) (citation) several times using short high-quality reads. We tried different strategies on An. coluzzii contig assembly obtained by Canu. After each run of a polishing program, we queried the resulting genome for a set of diptera and metazoa conserved single-copy genes (see supplementary table #). It should be noted that BUSCO single-copy genes usually covers a short portion of a genome and it remains unclear how these polishing tools perform on regions that contain repeats or represent non-coding sequences.

I ran Nanopolish on Canu contig assembly of An. coluzzii genome. Nanopolish corrected 283,935 substitutions, 1.6M insertions, and 51,104 deletions. My collaborator ran 4 rounds of Racon and then Medaka on the *an.arabiensis* assembly. If not stated otherwise, we report BUSCO score for the diptera gene set. The BUSCO score jumped from 77.6% to 93.6% of complete genes after Nanopolish and to 95.1% of complete genes after Racon+Medaka. Despite better BUSCO scores of Racon+Medaka polishing pipeline, we decided to proceed with assembly polished by Nanopolish because there exists an opinion that Nanopolish corrects errors in low-complexity regions better than Racon+Medaka. We also tried to run four rounds of Racon using nanopore raw reads after Nanopolish but that dropped the percentage of complete genes from 93.6% to 88.6%.

Using Canu contig assembly polished by Nanopolish, I ran Pilon several times by utilizing Illumina reads. After the first run of the Pilon, I had 97.9% of complete genes for diptera gene set. After three runs of Pilon, I reached 98.5% of complete genes. We did not run Pilon for the fourth time because the changes were insignificant during the third run. I also tried to run the second time Nanopolish after the first round of Pilon but this dropped the BUSCO's score to 95.9%.

I ran Nanopolish and three rounds of Pilon on Canu contig assembly of An. arabiensis genome. After Nanopolish, BUSCO score became equal 94.5% (for Canu contig assembly, it was 83%). Nanopolish corrected 143458 substitutions, 1.1M insertions, and 40694 deletions. After three rounds of Pilon, I obtained the assembly with 98.6% complete genes. For the next scaffolding step, I also polished An. coluzzii and An. arabiensis unitig assembly obtained by Canu using Nanopolish and three rounds of Pilon.

The final BUSCO scores for An. coluzzii and An. arabiensis assemblies are similar with BUSCO score for An. gambiae PEST (AgamP4) genome (i.e., 98.5%, 98.6%, and 98.3% of complete genes, respectively).

* + 1. Scaffolding

After draft assembly and polishing we have .fasta files two for each species. One file for Canu contigs assembly and other for Canu unitigs. Our task was using information from Hi-C experiment reconstruct chromosome scaffolds.

I experimented with different tools to perform scaffolding. This process is not fully automatized now and last steps must be done by hand. I used three tools HiCExplorer, SALSA2 and JuiceBox. Only last one have option for by hand editing of results.

Using of HiCExplorer pipeline did not come to any meaningful results. I describe here only SALSA2 and JuiceBox.

* + 1. SALSA2

SALSA2 is a tool for scaffolding long read assemblies with Hi-C data.

To start the scaffolding, first step is to map reads to the assembly. BWA mem tool was used for reads mapping. The read mapping generates a .bam file. SALSA requires .bed file as the input. Bam file was done using the bamToBed command from the Bedtools package. Also, SALSA requires bed file to be sorted by the read name, rather than the alignment coordinates. This was done with sorting options in samtools sort command.

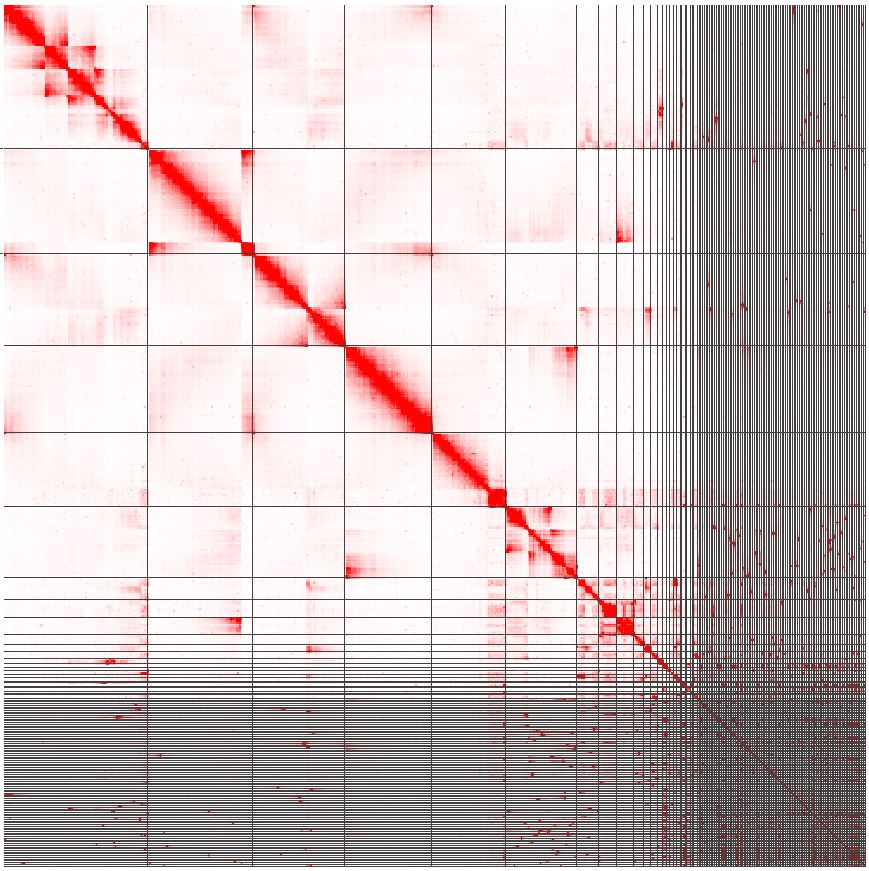
SALSA requires contig lengths as an input. File with contig lengths was created using samtools faidx command on contig sequence file.

Hi-C experiments can use different restriction enzymes. SALSA2 uses the restriction sites frequency in contigs to normalize the Hi-C interaction frequency. Restriction site for the enzyme which was used for Hi-C experiment need to be specified while running SALSA2.

I had contig sequences in polished genome assembly and the alignment bam file but also wanted to use Hi-C data to correct input assembly errors. Method that allows to correct some of the errors in the assembly with Hi-C data was implemented in SALSA2.

SALSA2 generates a bunch of files in the output folder. SALSA is an iterative algorithm, so it generates files for each iteration. The files I was interested in contain sequences of scaffolds generated by the algorithm. Another file which was of interest is .agp file, which is the agp style output for the scaffolds describing the assignment, orientation and ordering of contigs along the scaffolds.

After this process I created a Hi-C map file(.hic) using Juicer tool. This map was visualized using JuiceBox. Visualization of the SALSA output is presented on figure #.



* + - 1. SALSA2 generated Hi-C map for *an.coluzzii* canu contigs assembly. Map is visualized using JuiceBox. Each dot represents Hi-C signal for 500kbp region. Scaffold borders are represented by black lines

Special .hic file format is a container for Hi-C maps with different levels of resolution. Hi-C map is a matrix each element of which is a count of aligned Hi-C reads at positions on genome that corresponds to number of row and column. In that version of visualization using JBAT each point corresponds to number of reads that was aligned to particular coordinates in genome. Count of reads is presented by color of a dot. More red means more reads were aligned. White dot means no aligned reads at these coordinates.

Resolution of Hi-C map determines how many base pairs of genome are presented as one dot. In .hic file more than one map can be stored with different levels of resolution.

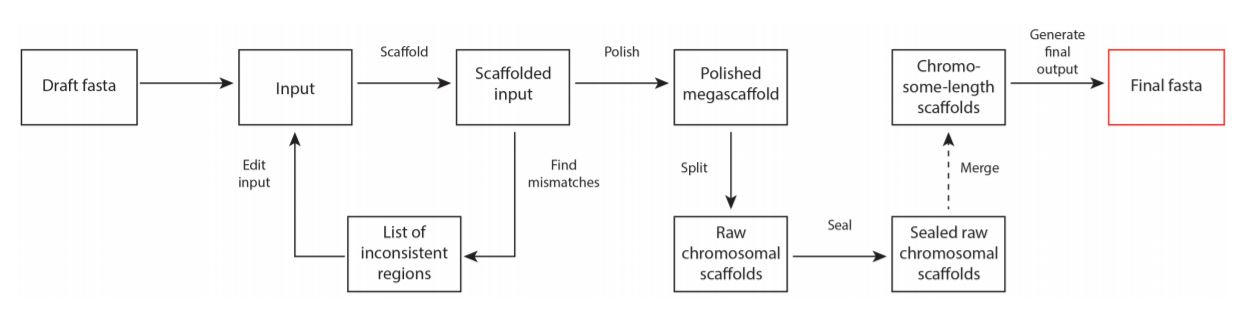
For visualizing .hic files can be used different online and offline tools. JuiceBox from Aiden Lab was chosen because .hic file was initially designed and standardized in Aiden Lab.

At figure # we can see trial of SALSA2 tool to automatically rearranged contigs from polished genome and construct scaffolds. Each scaffold region is presented by gridlines. We can see that diagonal is not in correct state. We expect that diagonal must be more consistent without breakage. This expectation follows the fact that there are more contacts between chromatin regions that are closer with each other than between far regions.

As we can see automatic output of SALSA was not a complete chromosome level genome. It must be corrected by hand. But there was no clear ways to do by hand correction because of issues with initial division of assembly into “chromosomal” regions that cant be treated with JuiceBox without changing this separation in .hic file and creating special .assembly file needed to JuiceBox for by-hand scaffolding. Thus, we tried automatic scaffolding pipeline from Aiden Lab.

* + 1. 3D-DNA and JuiceBox

3D-DNA is a custom computational pipeline to correct misassembles, anchor, order and orient fragments of DNA based on Hi-C data. Information about usage of 3D-DNA pipeline was obtained from “Genome Assembly Cookbook” written by authors of this pipeline. An overview of the workflow is schematically given in figure #.



* + - 1. 3D-DNA pipeline

Next is the short description of automated scaffolding pipeline of 3D-DNA from cookbook.

The pipeline starts with setting aside very small scaffolds (threshold side defined by the --input option). The remaining scaffolds are ordered and oriented, and the output is used to detect and correct misjoins in the input scaffolds. The corrected scaffolds again become subjects to ordering and orienting: the procedure (from scratch). This can be repeated several times (controlled by the --rounds option). Once the iterative scaffolding and misjoin detection are finished, the results are polished by running a coarse-grained misassembly detection and rescaffolding the resulting large pieces. The resulting megascaffold is then split into chromosomes, sealed to examine and restore false-positive edits introduced during misjoin detection.

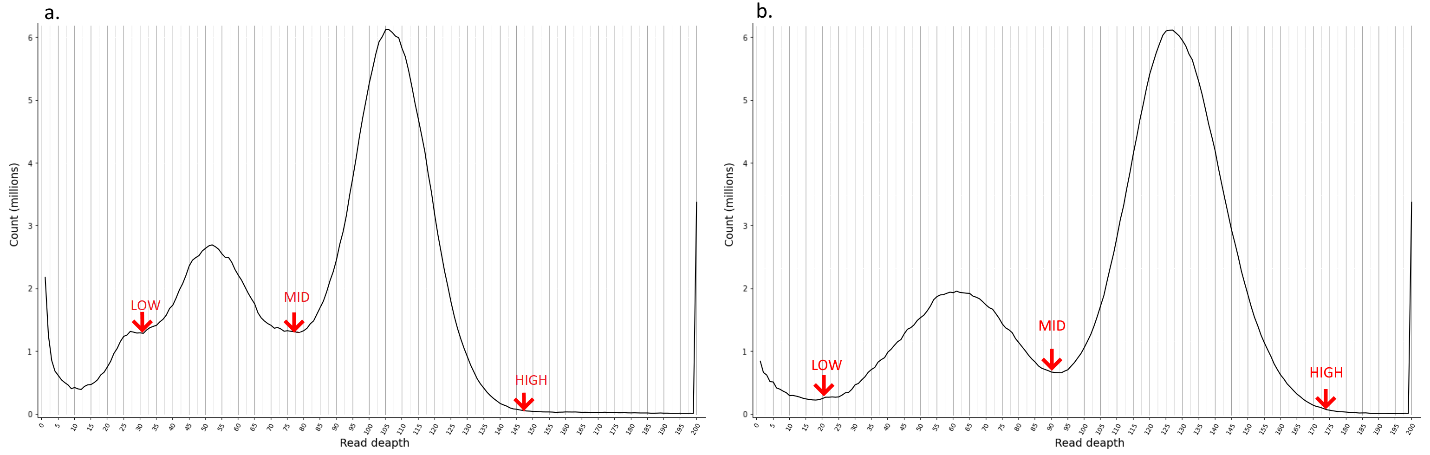
3D-DNA automated scaffolding for each assembly was performed by my supervisor

* + 1. Purge haplotigs

Each contig was marked as primary contig, haplotig, or assembly artefact using Purge Haplotigs.

Purge Haplotigs is a computational pipeline to deal with diploid assemblies e.g. FALCON. But in case of presence in haploid assembly large number of haplotigs it can help to mark them using read coverage. The pipeline includes three steps.

Purge Haplotigs uses reads alignment the curated assembly. It was done with minimap2. First step is a creating read-depth histogram. Histogram for the both species are shown in figure #.



* + - 1. Purge Haplotigs read-depth histograms. a. *an.coluzzii* b. *an.arabiensis* draft assemblies

X-axis is read-deapth or coverage Y-axis is nucleotide count. We can see that distribution is bimodal. This is because real genome is diploid and there is haplotigs in assembly (not only from diploidy but from different species in sequencing library too). The first read-depth peak results from the duplicated regions that corresponds to 'haploid' level of coverage (0.5x). Contigs with this coverage are our suspects to be haplotigs. The second read-depth peak results from regions that are haplotype-fused that corresponds to 'diploid' level of coverage (1x). Contigs with this level we will mark as primary. Contigs with inadequate mix of coverage levels we will mark as assembly artefacts. At this step we must choose low, mid, and high cutoffs for coverage. Cutoffs for *an.coluzzii* are: # # #, for *an.arabiensis*: # # #.

The second step is producing a contig coverage stats .csv file with suspect contigs flagged for further analysis or removal.

The third step is the iterative purging pipeline. The script will automatically run a windowed coverage analysis and assess which contigs to reassign and which to keep.

At the end of the Purge Haplotigs process we had three .fasta files: primary contigs, haplotigs, and assembly artefacts. I used this information in by-hand scaffolding process.

* + 1. Validation genes from reference assembly

I mapped genes from An Gambiae using blast. Tracks were created for genes from each PEST chromosome separately. The results are in the first attached picture.

In PEST features gff3 there are 13 057 genes

13 038 were mapped with different alignment length and quality scores.

9 503 (72.8%) were mapped with e-value = 0 and alignment length equal to gene length or less no more than for 10%

statistics for each chromosome (e-val=0, alen >= 0.9len):

X chromosome: 579 / 1063 54.5%

2R arm: 2806 / 3668 76.5%

2L arm: 2105 / 2935 71.7%

3R arm: 1959 / 2686 72.9%

3L arm: 1644 / 2211 74.4%

Y\_unplaced: 2 / 2 100%

UNKN: 395 / 479 82.5%

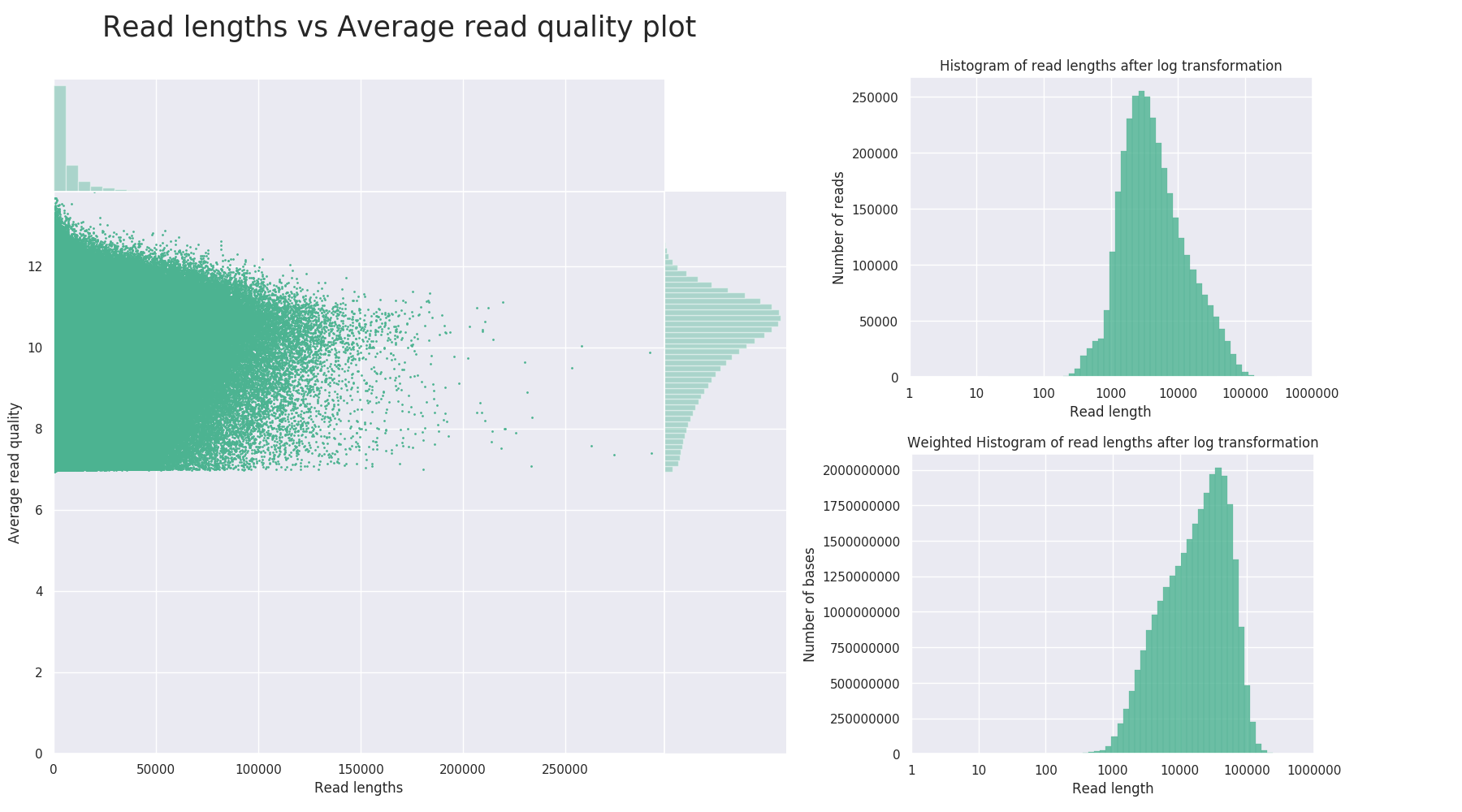
Mt: 13 / 13 100%

APPENDIX A

Supplementary figures and tables for Mosquitos project.

|  |  |  |
| --- | --- | --- |
|  | An.Coluzzii | An.Arabiensis |
| **General summary:** | | |
| Mean read length: | 8509.6 | 6788.7 |
| Mean read quality: | 10.1 | 9.3 |
| Median read length: | 3833 | 2 256 |
| Median read quality: | 10.3 | 10 |
| Number of reads: | 3 299 012 | 5 094 106 |
| Read length N50: | 19 315 | 21 969 |
| Total bases: | 28 073 279 865 | 34 582 156 501 |
| **Number, percentage and megabases of reads above quality cutoffs** | | |
| >Q5: | 3299012 (100.0%) 28073.3Mb | 4718400 (92.6%) 33503.9M |
| >Q7: | 3297112 (99.9%) 28059.2Mb | 4457124 (87.5%) 32336.6Mb |
| >Q10: | 1994146 (60.4%) 17513.7Mb | 2520015 (49.5%) 20173.2Mb |
| >Q12: | 34417 (1.0%) 202.7Mb | 62557 (1.2%) 232.6Mb |
| >Q15: | 0 (0.0%) 0.0Mb | 0 (0.0%) 0.0Mb |
| **Top 5 highest mean basecall quality scores and their read lengths** | | |
| 1: | 13.8 (19847) | 14.5 (33760) |
| 2: | 13.7 (924) | 14.5 (651) |
| 3: | 13.7 (576) | 14.4 (484) |
| 4: | 13.7 (1109) | 14.3 (19429) |
| 5: | 13.6 (1416) | 14.3 (177) |
| **Top 5 longest reads and their mean basecall quality score** | | |
| 1: | 298483 (9.9) | 276317 (9.0) |
| 2: | 292246 (7.4) | 276021 (10.7) |
| 3: | 291487 (9.9) | 265300 (10.6) |
| 4: | 273986 (7.4) | 252875 (8.5) |
| 5: | 262854 (7.6) | 245928 (9.3) |

* + - * 1. Statistics for nanopore long reads



* + - 1. An.Coluzzii nanopore reads quality and lengths distribution plots



* + - 1. An.Arabiensis nanopore reads quality and lengths distribution plots

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Anopheles Coluzzii** | | | | |  | Anopheles Arabiensis | | |
| **wtdbg2** | **miniasm** | **Flye** | **CANU - unitigs** | **CANU - contigs** | **wtdbg2** | **Flye** | **CANU - contigs** | **CANU-unitigs** |
| **Genome statistics** | **Genome statistics (A.Gambiae PEST assembly as reference)** | | | | | | | | |
| Genome fraction (%) | 54,32 | 0,16 | 52,58 | 59,18 | 58,48 | 52,869 | 43,074 | 56,24 | 59,89 |
| Duplication ratio | 1,07 | 1,02 | 1,14 | 1,36 | 1,25 | 1,13 | 1,139 | 1,23 | 1,31 |
| Largest alignment | 1 440 034 | 3 064 | 1 154 080 | 1 284 741 | 1 469 500 | 2 082 067 | 2 182 677 | 2 090 800 | 2 126 425 |
| Total aligned length | 145 808 258 | 420 914 | 150 581 571 | 202 100 538 | 184 201 098 | 150 768 812 | 123 807 719 | 173 732 578 | 196 854 393 |
| NG50 | 3 518 408 | 3 351 593 | 3 617 662 | 4 485 652 | 13 842 187 | 7 492 164 | 10 796 042 | 23 700 761 | 23 710 802 |
| NG75 | 712 008 | 956 538 | 565 263 | 550 578 | 6 304 690 | 737 639 | 1 341 337 | 6 985 567 | 1 144 137 |
| NA50 | 2 952 | NA | 2 166 | 4 228 | 3 874 | 629 | NA | 10 884 | 27 527 |
| NGA50 | 2 196 | NA | 2 658 | 44 375 | 17 350 | 1 625 | NA | 15 731 | 54 779 |
| LG50 | 17 | 20 | 21 | 16 | 7 | 12 | 7 | 5 | 5 |
| LG75 | 54 | 57 | 78 | 71 | 14 | 38 | 25 | 10 | 25 |
| LA50 | 3 508 | NA | 5 884 | 4 036 | 4 247 | 20 751 | NA | 1 135 | 994 |
| LGA50 | 4 675 | NA | 4 717 | 837 | 1 155 | 8 457 | NA | 980 | 677 |
| **Misassemblies** | **Misassemblies** | | | | | | | | |
| # misassemblies | 4 316 | 0 | 6 196 | 11 834 | 10 729 | 4 061 | 3 958 | 12 308 | 12 860 |
| # relocations | 1 706 | 0 | 2 467 | 4 461 | 3 880 | 1 320 | 1 287 | 4 396 | 4 276 |
| # translocations | 2 554 | 0 | 3 634 | 7 222 | 6 723 | 2 706 | 2 631 | 7 826 | 8 477 |
| # inversions | 56 | 0 | 95 | 151 | 126 | 35 | 40 | 86 | 107 |
| # misassembled contigs | 300 | 0 | 360 | 518 | 210 | 334 | 151 | 134 | 348 |
| Misassembled contigs length | 197 415 431 | 0 | 213 113 463 | 258 775 961 | 240 726 872 | 166 552 571 | 168 135 090 | 238 615 865 | 247 176 514 |
| # local misassemblies | 18 019 | 0 | 20 122 | 26 014 | 22 903 | 22 376 | 23 629 | 30 814 | 31 980 |
| # scaffold gap ext. mis. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| # scaffold gap loc. mis. | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| # possible TEs | 2 228 | 0 | 2 320 | 3 704 | 3 478 | 1 878 | 1 422 | 2 824 | 3 138 |
| # unaligned mis. contigs | 510 | 86 | 321 | 358 | 191 | 831 | 266 | 45 | 93 |
| **Unaligned** | **Unaligned** | | | | | | | | |
| # fully unaligned contigs | 310 | 462 | 174 | 24 | 12 | 428 | 716 | 11 | 21 |
| Fully unaligned length | 6 082 863 | 88 230 278 | 3 238 628 | 1 858 773 | 985 300 | 15 125 468 | 26 219 848 | 1 255 286 | 2 375 670 |
| # partially unaligned contigs | 1 051 | 172 | 834 | 1 024 | 450 | 1 456 | 529 | 200 | 497 |
| Partially unaligned length | 115 076 975 | 230 334 583 | 124 479 154 | 139 181 121 | 128 372 967 | 132 515 795 | 139 638 537 | 101 637 092 | 98 726 070 |
| **Mismatches** | **Mismatches** | | | | | | | | |
| # mismatches | 3 560 597 | 5 311 | 3 801 380 | 4 738 319 | 4 239 427 | 5 677 775 | 4 537 311 | 6 933 636 | 7 814 852 |
| # indels | 1 160 243 | 6 779 | 1 724 034 | 1 269 221 | 1 167 976 | 1 040 792 | 1 294 719 | 994 669 | 784 501 |
| Indels length | 2 744 847 | 10 804 | 3 289 354 | 3 511 488 | 3 215 434 | 2 579 692 | 2 568 697 | 2 851 751 | 2 863 834 |
| # mismatches per 100 kbp | 2 596,36 | 1 289,52 | 2 863,66 | 3 171,32 | 2 871,56 | 4 254 | 4 173 | 4 884 | 5 169 |
| # indels per 100 kbp | 846,04 | 1 645,96 | 1 298,75 | 849,48 | 791,13 | 780 | 1 191 | 701 | 519 |
| # indels (<= 5 bp) | 1 081 711 | 6 686 | 1 654 237 | 1 163 569 | 1 070 969 | 959 948 | 1 235 354 | 897 417 | 672 054 |
| # indels (> 5 bp) | 78 532 | 93 | 69 797 | 105 652 | 97 007 | 80 844 | 59 365 | 97 252 | 112 447 |
| # N's | 0 | 0 | 500 | 0 | 0 | 0 | 500 | 0 | 0 |
| # N's per 100 kbp | 0,00 | 0,00 | 0,18 | 0,00 | 0,00 | 0 | 0,17 | 0,00 | 0,00 |
| **Statistics without reference** | **Statistics without reference** | | | | | | | | |
| # contigs | 1 391 | 634 | 1 048 | 1 055 | 465 | 1 920 | 1 280 | 211 | 521 |
| # contigs (>= 0 bp) | 1 392 | 638 | 1 618 | 1 073 | 474 | 1 928 | 2 048 | 220 | 541 |
| # contigs (>= 1000 bp) | 1 392 | 638 | 1 388 | 1 073 | 474 | 1 928 | 1 763 | 220 | 541 |
| # contigs (>= 5000 bp) | 1 348 | 630 | 861 | 1 051 | 463 | 1 883 | 1 060 | 208 | 517 |
| # contigs (>= 10000 bp) | 1 067 | 622 | 731 | 1 045 | 460 | 1 428 | 836 | 207 | 512 |
| # contigs (>= 25000 bp) | 653 | 616 | 624 | 1 029 | 455 | 929 | 644 | 207 | 507 |
| # contigs (>= 50000 bp) | 364 | 606 | 502 | 935 | 432 | 564 | 455 | 205 | 494 |
| Largest contig | 17 446 215 | 18 411 938 | 13 804 856 | 19 763 313 | 33 413 712 | 22 238 065 | 32 507 593 | 44 591 211 | 33 440 724 |
| Total length | 267 203 205 | 318 985 775 | 278 686 226 | 343 961 469 | 314 190 168 | 298 413 078 | 289 704 167 | 277 203 818 | 298 533 470 |
| Total length (>= 0 bp) | 267 206 174 | 318 993 565 | 279 473 599 | 343 996 597 | 314 208 573 | 298 431 996 | 290 752 171 | 277 218 797 | 298 564 895 |
| Total length (>= 1000 bp) | 267 206 174 | 318 993 565 | 279 331 846 | 343 996 597 | 314 208 573 | 298 431 996 | 290 567 540 | 277 218 797 | 298 564 895 |
| Total length (>= 5000 bp) | 267 016 028 | 318 970 674 | 277 964 236 | 343 945 192 | 314 182 073 | 298 254 454 | 288 856 343 | 277 192 427 | 298 517 822 |
| Total length (>= 10000 bp) | 264 994 751 | 318 910 721 | 277 080 562 | 343 900 991 | 314 159 967 | 294 987 018 | 287 284 715 | 277 183 822 | 298 485 334 |
| Total length (>= 25000 bp) | 257 974 837 | 318 820 092 | 275 269 049 | 343 630 110 | 314 083 595 | 286 915 173 | 284 122 552 | 277 183 822 | 298 387 283 |
| Total length (>= 50000 bp) | 247 916 099 | 318 460 456 | 270 692 452 | 340 005 869 | 313 137 556 | 273 706 097 | 277 219 122 | 277 094 612 | 297 857 656 |
| N50 | 4 111 930 | 2 679 573 | 3 617 326 | 1 211 583 | 13 401 784 | 5 903 928 | 10 796 042 | 23 700 761 | 15 085 722 |
| N75 | 897 490 | 448 329 | 507 563 | 222 639 | 2 423 489 | 268 064 | 603 035 | 6 985 567 | 475 129 |
| L50 | 16 | 28 | 22 | 30 | 8 | 14 | 7 | 5 | 6 |
| L75 | 48 | 111 | 86 | 229 | 21 | 85 | 39 | 10 | 54 |
| GC (%) | 44 | 44 | 43 | 44 | 44 | 43 | 43 | 44 | 44 |
| **K-mer-based statistics** | **K-mer-based statistics** | | | | | | | | |
| K-mer-based compl. (%) | 14,53 | 0,24 | 10,43 | 15,65 | 15,95 | 8,35 | 5,70 | 9,18 | 11,57 |
| K-mer-based cor. length (%) | 23,43 | 34,90 | 48,30 | 34,34 | 10,88 | 11,68 | 20,45 | 6,28 | 19,47 |
| K-mer-based mis. length (%) | 64,97 | 22,41 | 43,84 | 52,74 | 78,74 | 58,02 | 59,66 | 84,51 | 68,44 |
| K-mer-based undef. length (%) | 11,60 | 42,68 | 7,87 | 12,92 | 10,38 | 30,30 | 19,89 | 9,21 | 12,10 |
| # k-mer-based misjoins | 266 | 14 | 215 | 512 | 513 | 108 | 54 | 168 | 216 |
| # k-mer-based translocations | 171 | 4 | 113 | 267 | 254 | 82 | 29 | 117 | 174 |
| # k-mer-based 100kbp relocations | 95 | 10 | 102 | 245 | 259 | 26 | 25 | 51 | 42 |

* + - * 1. Quast-lg reports for draft assemblies

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | BUSCO diptera | | | | | | BUSCO metazoa | | | | | |
|  | assemblies | complete | single | duplicated | fragmented | missing | number | complete | single | duplicated | fragmented | missing | number |
| A.Coluzzii |  | **A.Coluzzii MOPTI Draft Assemblies** | | | | | | | | | | | |
| CANU unitigs | 79.3% | 73.0% | 6.3% | 12.9% | 7.8% | 2799 | 94.4% | 84.4% | 10.0% | 2.7% | 2.9% | 978 |
| CANU contigs | 77.6% | 72.7% | 4.9% | 13.7% | 8.7% | 93.3% | 85.3% | 8.0% | 3.5% | 3.2% |
| wtdbg2 | 65.9% | 65.6% | 0.3% | 18.7% | 15.4% | 87.4% | 86.9% | 0.5% | 6.9% | 5.7% |
| Flye | 62.7% | 61.7% | 1.0% | 19.0% | 18.3% | 81.2% | 80.0% | 1.2% | 9.7% | 9.1% |
| miniasm | 1.4% | 1.4% | 0.0% | 3.1% | 95.5% | 4.6% | 4.6% | 0.0% | 21.1% | 83.3% |
| A.Arabiensis |  | **A.Arabiensis DONGOLA draft assemblies** | | | | | | | | | | | |
| CANU unitigs | 83.9% | 78.0% | 5.9% | 10.8% | 5.3% | 2799 | 94.4% | 87.5% | 6.9% | 3.3% | 2.3% | 978 |
| CANU contigs | 83.0% | 78.% | 4.1% | 11.4% | 5.6% | 94.1% | 88.8% | 5.3% | 3.5% | 2.4% |
| Flye | 63.0% | 62.6% | 0.4% | 18.5% | 18.5% | 81.9% | 80.7% | 1.2% | 9.3% | 8.8% |
| wtdbg2 | 70.4% | 70.2% | 0.2% | 16.0% | 13.6% | 85.7% | 85.1% | 0.6% | 5.1% | 9.2% |
| miniasm | 1.5% | 1.5% | 0.0% | 4.4% | 94.1% | 8.2% | 8.2% | 0.0% | 18.8% | 73.0% |

* + - * 1. BUSCO scores for draft assemblies