LOCAS - a Low Coverage Assembler for Next Generation Sequencing and Resequencing Data

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Dipl.-Bioinf. Juliane Damaris Klein
aus Halle an der Saale

Tübingen
2010

Tag der mündlichen Qualifikation:	16.02.2011
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Daniel H. Huson
2. Berichterstatter:	Prof. Dr. Detlef Weigel

Abstract

Within the last five years, a new generation of sequencing technologies has dramatically reduced cost and at the same time increased throughput of genome sequencing. For most application fields these technologies have proven to be good alternatives to the traditional Sanger sequencing although they generate shorter read sequences. For the study of sequence variations like SNPs, indels and longer variant regions between highly related genomes, resequencing has become increasingly popular. Such analyses help to reveal the impact of sequence variations on responses to the environment and in developing diseases. They are, thus, of great interest to disease control, personal genomics and phylogenetic studies.

Currently, the most popular approach to resequencing large and complex genomes is the mapping-consensus approach. It maps the read sequences to a highly related reference genome and from the alignment calculates a consensus sequence which can be compared to the reference genome. Unfortunately, only SNPs and small indels can be detected with this approach. A more promising approach is homology-guided assembly. Here, the reads are mapped against a reference sequence and the layout of the reads is refined before the calculation of the consensus sequence. This method has the capability to additionally reveal the sequences of longer variant regions such as long insertions.

In this thesis, I present an extension to homology-guided assembly that aims at assembling not only regions that are homologous between the target and reference genome but also longer variant regions. After the reads have been mapped to the reference sequence, the reference sequence is partitioned into regions of a fixed length, called blocks. In a reassembly step, the reads of each pair of consecutive blocks are assembled together. In order to also find long variant regions, reads that cannot be mapped onto the reference genome, so called left-over reads, are recruited and incorporated in the assembly of the current blocks.

The main focus of this work was on the development of assembly algorithms for current resequencing projects. To meet the needs of these projects the developed algorithms were especially designed for short read data at low sequencing depth. Furthermore, this work comprises extensions to these assembly algorithms, which are used in the reassembly step of our homology-guided assembly approach. These algorithms additionally incorporate left-over reads in the assembly and can utilize mapping positions that are available for the reads. The assembly algorithms are implemented in the assembly tool LOCAS (Low Coverage ASsembly) and its extension SUPERLOCAS.

The developed tools were evaluated and compared to state-of-the-art assemblers on short read data within a homology-guided assembly approach. For this purpose,

resequencing scenarios with a low sequencing depth were simulated. In the first study, which simulated assemblies of blocks, LOCAS showed better or comparable results regarding error rate and contig size while producing contigs with the best trade-off between both measures. In the second study, which simulated assemblies of blocks with the incorporation of left-over reads, SUPERLOCAS proved to be the superior tool regarding contig size, error rate and runtime while assembling the same amount of long insertion regions as comparable assemblers. In a third study, which used real world data, LOCAS and SUPERLOCAS performed similar as in the simulated studies. In all studies both tools proved to be very robust to different parameter settings.

In conclusion, my homology-guided assembly approach overcomes the problems of the mapping-consensus approach. In addition to homologous regions, it also assembles longer variant regions. Compared to other assembly methods, LOCAS and SUPERLOCAS are well suited for reassembly and show superior performances in this scenario.

Zusammenfassung

Eine neue Generation von Sequenziertechnologien hat in den letzten fünf Jahren die Kosten für die Genomsequenzierung deutlich verringert und gleichzeitig den Sequenzierdurchsatz erhöht. Die neuen Sequenziertechnologien haben sich in vielen Anwendungsgebieten als vielversprechende Alternative zur traditionellen Sangersequenzierung erwiesen, obwohl die erzeugten Sequenzfragmente, welche als Reads bezeichnet werden, deutlich kürzer sind. Zur Untersuchung von Punktmutationen (SNPs), kleinen Insertionen und Deletionen (Indels) sowie längeren variablen Bereichen von nahverwandten Genomen wird inzwischen immer häufiger das Verfahren der Resequenzierung eingesetzt. Mit diesem Analyseverfahren kann die Bedeutung von Sequenzvariationen bei Krankheiten oder in der Reaktion auf die Umwelt festgestellt werden. Daher ist die Resequenzierung von großem Interesse bei der Kontrolle von Krankheiten, in Bereich Personal-Genomics und in phylogenetischen Studien.

Momentan wird bei der Resequenzierung von langen und komplexen Genomen vor allem der Mapping-Consensus Ansatz verwendet. Dabei werden die Reads gegen ein nahverwandtes Referenzgenom aligniert und die Consensus-Sequenz der alignierten Reads berechnet, sodass diese mit der Referenzsequenz verglichen werden kann. Da die Reads meist nur diskontinuierlich aligniert werden können, besteht die Consensus-Sequenz meist aus mehreren Teilsequenzen, welche als Contigs bezeichnet werden. Der Nachteil bei diesem Ansatz ist, dass meist nur SNPs und Indels bestimmt werden können, während lange variable Bereiche unentdeckt bleiben. Ein Ansatz, der hierfür weitaus erfolgversprechender ist, ist das Homology-Guided Assembly. Hier werden die Reads ebenfalls gegen eine Referenzsequenz aligniert. Jedoch wird die Anordnung der Reads anschließend noch einmal verbessert, bevor schließlich die Consensus-Sequenz berechnet wird. Dieser Ansatz hat das Potenzial auch die Sequenz von längeren variable Bereichen, wie langen Insertionen, zu bestimmen.

In meiner Dissertation stelle ich einen erweiterten Ansatz des Homology-Guided Assemblies vor. Durch diesen neuen Ansatz werden nicht nur homologe Bereiche des Referenz- und Zielgenoms assembliert sondern auch lange variable Bereiche. Nachdem die Reads gegen die Referenzsequenz aligniert worden sind, wird die Referenzsequenz in Abschnitte einer festen Länge unterteilt, welche als Blocks bezeichnet werden. Diese Blocks werden anschließend reassembliert, d.h., alle Reads die zu zwei aufeinanderfolgenden Blocks zugeordnet sind werden miteinander assembliert. Dabei werden Reads, die nicht gegen das Referenzgenom aligniert werden konnten (Left-Over Reads), in das Assembly eingebaut, sodass auch lange variable Bereiche assembliert werden können.

Der Hauptaugenmerk meiner Arbeit lag auf der Entwicklung von Assemblierungsalgorithmen, die in Resequenzierungsprojekten, welche die neue Generation der Sequenziertechnologien nutzen, angewendet werden können. Um den Anforderungen dieser Projekte Rechnung zu tragen, wurden die Algorithmen speziell an eine kurze Länge der Reads und an eine niedrige Sequenziertiefe angepasst. Darüber hinaus wurden die Algorithmen so erweitert, dass sie auch zur Reassemblierung genutzt werden können. Durch diese Erweiterung werden auf eine effiziente Weise auch Left-Over Reads mit in das Assembly einbezogen. Weiterhin können eventuell vorhandene Positionen der Reads bezüglich der Referenzsequenz für die Assemblierung genutzt werden. Die Algorithmen wurden in das Assemblierungsprogramm LOCAS bzw. in dessen Erweiterung SUPERLOCAS implementiert.

Die entwickelte Software wurde in einer Vergleichsstudie evaluiert und mit anderen aktuellen Assemblern verglichen. Die Assembler wurden zur Reassemblierung innerhalb des beschriebenen Homology-Guided Assembly Ansatzes verwendet. Zu diesem Zweck wurden kurze Reads mit einer niedrigen Sequenziertiefe innerhalb von Resequenzierungsszenarien simuliert. In der ersten Studie, welche die Reassemblierung von Blocks simulierte, erzielte LOCAS bessere oder vergleichbare Ergebnisse bezüglich der Fehlerrate und der Contig-Länge. Gleichzeitig erreichte es den besten Kompromiss zwischen beiden Maßen. In der zweiten Studie, welche die Reassemblierung von Blocks unter Einbeziehung von Left-Over Reads simulierte, stellte sich SUPERLOCAS als der beste Assembler bezüglich der Contig-Länge, der Fehlerrate und der Laufzeit heraus. In einer dritten Studie, die auf realen Daten basierte, zeigten LOCAS und SUPERLOCAS die gleiche Leistung wie in den Simulationsstudien. In allen Studien waren beide Assembler sehr robust gegenüber unterschiedlichen Parametereinstellungen.

Aus den Ergebnissen dieser Arbeit lässt sich folgern, dass die angesprochenen Probleme des Mapping-Consensus Ansatzes durch den vorgestellten Homology-Guided Assembly Ansatz in weiten Punkten gelöst werden. Zusätzlich zu den homologen Bereichen werden nun auch längere variable Bereiche assembliert. LOCAS und SUPERLOCAS erwiesen sich für die Reassemblierung von Genomen innerhalb des Homology-Guided Assembly-Ansatzes als sehr geeignete Assembler, da sie ausgezeichnete Ergebnisse für dieses Szenario erzielten.

Acknowledgments

First of all, I want to thank my advisors Prof. Dr. Daniel Huson and Prof Dr. Detlef Weigel for raising the interesting questions that formed the basis for this thesis. I very much appreciated their support and their ideas, which I gratefully adopted for this work.

Moreover, I would like to thank Prof. Dr. Daniel Huson, for his steady supervision, for providing direction as well as setting clear limits for my works. I am very thankful for the given freedom to explore and the trust that has been put in me. I am grateful for my participating in scientific conferences, that gave me the opportunity to get insights into current research topics and broadened my scientific and personal horizon.

I thank my collaborators Dr. Stephan Ossowski, Korbinian Schneeberger and Prof. Dr. Detlef Weigel from the Max Planck Institute in Tübingen for helpful discussions. Furthermore, I would like to thank my colleagues at the University of Tübingen. Their comments helped me to improve talks and publications. Working together has been a pleasure.

I am also grateful to Sebastian Briesemeister, Nora Toussaint, Regina Bohnert, Nico Weber and Korbinian Schneeberger for critically reading this thesis manuscript. Their suggestions and corrections helped to improve it substantially. In addition, I would like to thank my office room mates Dr. Regula Rupp, Dr. Johannes Fischer and Nico Weber for their patience, frankness and for providing a friendly atmosphere.

Moreover, I gratefully acknowledge funding from the BMBF GABI-GNADE project.

Finally, I want to thank my sister, who always encouraged me to give my best effort, and my parents, who have created a firm foundation for me to stand on. I would like to express my deep gratitude to my friends for their patience and constant encouragement during my Ph.D. – most of all I want to thank Johanna and Nora. Last, but definitely not least, I would like to express my sincerest thanks to Sebastian for inspiring discussions, his understanding and for his constant faith in me during the last years.

In accordance with the standard scientific protocol, I will use the personal pronoun "we" to indicate the reader and the writer, or my scientific collaborators and myself.

Contents

1. Introduction

Studying sequence variation, like single-nucleotide polymorphisms (SNPs), insertions and deletions, is important in disease genetics and pharmacogenomic studies. The alteration of SNP positions or appearance of insertions and deletions within gene sequences causes alterations in the function of proteins. Thus, these events can result in diseases, changed responses to drugs or environmental toxins and behavioral modifications.

The investigation of SNPs and other sequence variations in plants is essential for dealing with the growing world population in the future. Especially, sequence variations that have an effect on resistance and yield gain of cultured plants are of great interest. Consequently, these variations and their effects are widely studied in model organisms like *Arabidopsis thaliana*. Array-based genotyping, such as methods supported by Illumina and Affymetrix, has been the most prominent approach to investigate sequence variations. Due to advances in Second Generation Sequencing technologies (SGS, also called Next Generation Sequencing), whole genome resequencing studies have come as an alternative approach into focus. Popular examples are the 1000 Genomes Project, which aims at finding genetic variants that have frequencies of at least 1% in the human population, and the 1001 Genomes Project, which aims at discovering such variations in *A. thaliana*.

Since their introduction in 2005, SGS technologies have increased the throughput and cost-efficiency of sequencing by an order of magnitude. For example, at present, the Illumina Genome Analyzer GAIIx, which became commercially available in 2009, can produce up to 10 Gb of sequence in less than three days, while the latest sequencer of Illumina, the HiSeq 2000 system, yields up to 100 Gb in the same amount of time. While the accuracy of new sequencing technologies is similar to that of Sanger sequencing, the achievable read length has decreased from 1 kbp to less than 500 bases for the GS FLX Titanium instrument from Roche/454 Life Sciences or to around 100 bases or less for Illumina's GAIIx or HiSeq and Applied Biosystem's SOLiD instruments.

The SGS technologies are used in resequencing studies that investigate variations between strains of the same organism or closely related species. Usually, a known genome is used as *reference genome* in a *mapping* or *mapping-consensus* approach, in which the reads are aligned to the reference sequence to detect variations between the reference genome and the *target genome*. While this allows for the detection of small variations like SNPs or short insertions and deletions, so called *indels*, regions with high divergence or long insertions will not be represented in the resulting consensus sequence as the respective reads are often not alignable to the reference sequence.

An alternative approach is *de novo* assembly which calculates overlaps between reads to assemble longer sequences, which are called *contigs*. *De novo* assembly does not rely on alignments of reads onto a reference genome and, thus, is also capable of assembling highly polymorphic and longer insertion regions. Unfortunately, the *de novo* assembly of large genomes and genomes with complex regions that were sequenced with SGS technology still faces unsolved issues. Such assemblies result in large numbers of short contigs and demand high amounts of memory.

For short read data, neither the mapping-consensus approach nor *de novo* assembly can be utilized to detect longer variations in complex eukaryotic genomes. Thus, a novel assembly approach is required that detects all variations in a complex and large target genome, including longer insertions and polymorphic regions. Several strategies have been proposed to increase the number of detected variant regions, like reduced representation libraries [KUA+07] and gene-boosted assembly [SSPL08]. In addition, other approaches that detect rearrangements in the target genome from read quantity or mate-pair data have been introduced. However, these approaches do not reveal additional sequence information. Only the *homology-guided assembly* [PPDS04, RKD+09] approach has the potential to assemble eukaryotic genomes with longer variations. It combines the mapping-consensus approach with *de novo* assembly.

Comparable to the mapping-consensus approach, homology-guided (or comparative) assembly makes use of an available reference genome. The homology-guided assembly strategy presented in this thesis starts with aligning short reads to a reference genome followed by the local assemblies of reads that have been aligned within the same regions. These regions are called *blocks*. In the assembly of the blocks, also non-alignable reads, so called *left-over* reads, can be incorporated to reveal additional sequence information of the target genome, which often originates from insertions or highly polymorphic regions. This process of assembly and incorporation is called *reassembly*.

Currently, available assembly tools, such as VELVET [ZB08, ZMMB09], EULER-SR [CP08], ABySS [SWJ+09] and SOAPdenovo [LLZ+09], do not provide time-efficient methods for problems that arise in reassembly such as the incorporation of left-over reads in a consecutive execution of multiple local assemblies. The set of left-over reads can be huge compared to the set of reads belonging to a block. Furthermore, it consists not only of reads from highly polymorphic regions but also of erroneous reads, which, in our experience, can comprise about 5% to 20% of all reads from a sample. Since existing assemblers do not distinguish between aligned reads and left-over reads, they would have to assemble each block using all left-over reads, leading to an unacceptable increase in runtime.

The reassembly problem becomes even more difficult in the context of genome resequencing projects that are performed at low sequencing depth. The choice to sequence with a low depth results from a simple cost-benefit analysis: Even with a sequencing depth of 7x, most of the reference genome is covered by aligned reads, enabling the detection of most SNPs and small indels. Current state-of-

the-art assemblers for short reads calculate exact sub-sequence matches (k-mers) of the input reads, represent this information in a *de Bruijn graph* and extract finally contigs from the graph. Thus, these assemblers do not calculate overlap alignments between reads, i.e., alignments that involve the ends of both read sequences, and, consequently, cannot detect overlaps of reads with a substantial number of mismatches. However, for low sequencing depths it is necessary to include as many overlaps as possible in order to assemble low-coverage regions. Thus, assembly tools based on the de Bruijn graph paradigm are not well suited for low sequencing depth assembly or reassembly as they typically require depths of 20x to 30x [ZB08].

In the context of this thesis, we have developed algorithms to address the problems in reassembly of short read data at a low sequencing depth. These algorithms have been implemented in a new assembly tool, LOCAS (LOw Coverage Assembly Software). LOCAS is designed for assembling short to medium sized reads in a *de novo* fashion using an overlap-layout-consensus approach. In this approach, overlap alignments of reads are calculated and represented in a so called *overlap graph* from which the final contigs are extracted. It explicitly handles data of low sequencing depth by allowing mismatches in the overlap calculation of reads. An extension of LOCAS, called SUPERLOCAS, efficiently incorporates left-over reads in the assembly process. It calculates a pre-assembly for the left-over reads once and assembles each block separately by incorporating the part of the pre-assembly that overlaps with the reads of the respective block. In addition, it can take advantage of alignment positions of reads within the reference sequence. SUPERLOCAS' design is perfectly suited to the requirements of a homology-guided assembly approach for large genomes.

We show that LOCAS produces assemblies that are often better than those obtained by existing short read assemblers at a sequencing depth of 7.5x as measured by the $N50$ size and error rate. In addition, for the task of incorporating left-over reads, SUPERLOCAS shows to be superior to common assemblers regarding $N50$ size, error rate and runtime.

Following this introduction, in Chapter 2, sequencing technologies and computational approaches in genome resequencing are introduced. In Chapter 3, one of these approaches, homology-guided assembly, is refined and adapted to the needs of resequencing projects with short reads at a low sequencing depth. This approach employs a reassembly step. In Chapter 4, we present algorithms to assemble short read data at a low sequencing depth.We extend these algorithms in Chapter 5 to fulfill the additional requirements that arise with our specialized homology-guided assembly approach. The presented assembly algorithms have been implemented in the tools LOCAS and SUPERLOCAS. The evaluations of both tools and a comparison to other assemblers are provided in Chapter 6. Observed problems, provided solutions and the experimental evaluations are discussed in Chapter 7, before an overall conclusion of this work is given in Chapter 8.

LOCAS and SUPERLOCAS are open source projects that are distributed under the terms of the GNU General Public License. Both software tools can be down-

loaded from `http://www-ab.informatik.uni-tuebingen.de/software/locas`.

2. Introduction to Genome Resequencing

This chapter gives an introduction into sequencing technologies. We will present the traditional sequencing technology by Sanger as well as technologies of the second generation. Furthermore, we will discuss three strategies for genome resequencing and discuss their applicability to Second Generation Sequencing (SGS) data. In addition to the widely used mapping-consensus and *de novo* assembly approaches, we will introduce the more sophisticated homology-guided assembly approach.

2.1. Sequencing Technologies

To understand the inheritance of traits in organisms their genomes have to be studied. The genomes of many organisms are organized in chromosomes. A chromosome is a single strand of coiled DNA (deoxyribonucleic acid) with DNA-bound proteins, which stabilize its structure. DNA was discovered by Friedrich Miescher in Tübingen, Germany, in 1869. Its three dimensional structure was characterized by James D. Watson and Francis Crick in 1953. Chromosomal DNA consists of two chains of nucleotide molecules, i.e., adenine, guanine, cytosine, and thymine. Specific regions in DNA molecules, which are called genes, encode for RNA (ribonucleic acid). In a process that is called transcription, genes are read by the RNA polymerase to produce RNA molecules. If the gene codes for a protein, the product of transcription is a messenger RNA (mRNA), which is later translated into a protein. Otherwise, the transcription product is a non-coding RNA molecule which can have regulatory and other functions. With the regulation of their genes, organisms control their cellular or overall function. Thus, we can learn about genome function, developing diseases and evolution of species by analyzing and comparing their DNA sequences.

The first sequencing technology was invented by Frederick Sanger and others in the 1970's [SC75]. The process of DNA sequencing determines the sequence of nucleotides in a DNA molecule. Until the development of SGS technologies, sequencing of DNA had been a very expensive and complex process [Met09]. In 2005, the first new sequencing technology, Roche's FLX Genome Sequencer, was released [MEA+05]. In the following years, further technologies such as Illumina's Genome Analyzer [Ben06, BBS+08] and Applied Biosystem's sequencer SOLiD [SPR+05] became commercially available. The first system towards a

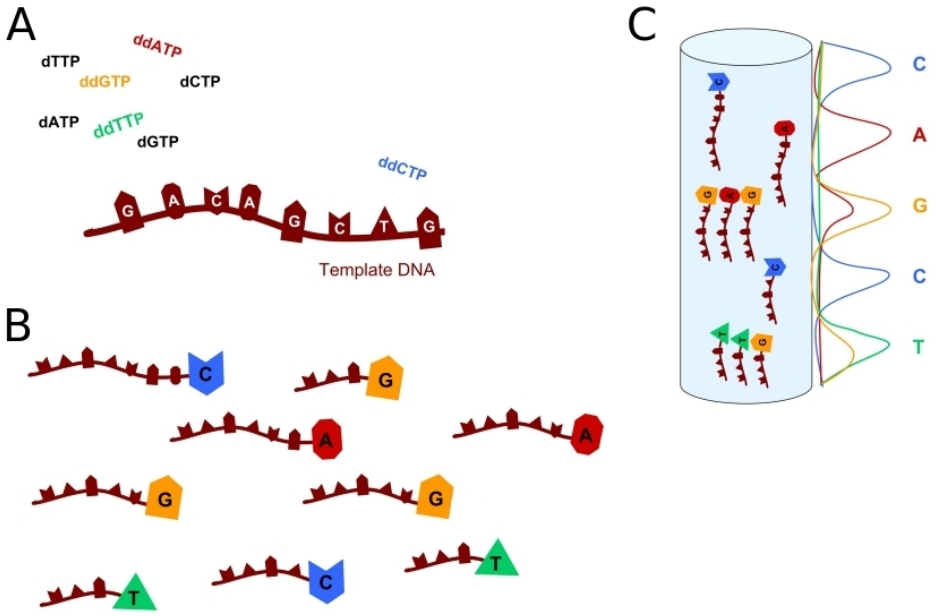

Figure 2.1.: Sanger Sequencing. (A) The initial reaction mix contains the single-stranded template DNA, dNTPs and in lower concentration ddNTPs with distinct fluorescent markers. DNA polymerase is added to start the PCR (not shown). After a ddNTP has been incorporated by chance, the reaction terminates. (B) The resulting sequence fragments have different length with distinct labels indicating their 3'-base. (C) Finally, the sequence fragments are separated via electrophoresis in mass-produced, gel-filled capillary tubes and their sequences are automatically determined as output. These sequences are given as intensity curves of the base-specific colors. The curve of each base displays the intensity of the base-signal for each position in the fragment.

third generation of sequencing technologies was the Helicos' Genetic Analysis System [HBB+08].

2.1.1. Sanger Sequencing

In the 1970's, two sequencing methods were developed independently: The Maxam-Gilbert method [MG80] and Sanger sequencing, which is also known as *chain termination* method. It was the more efficient and less toxic Sanger sequencing that became popular. Until today, the method has improved steadily by taking advantage of other inventions like the *polymerase chain reaction* (PCR) and *fluorescent labeling*. For determining the sequence of DNA fragments that exceed the length of a single sequencing run, a *shot gun approach* is used as follows (see Figure 2.1 A):

1. DNA is fragmented to form a library of single-stranded DNA fragments.
2. The fragments are subcloned into bacterial vectors. The vectors are inserted in bacterial cells for amplification. The amplified DNA fragments are used

as templates in the following sequencing process.

3. The fragments, deoxynucleotides (dNTPs) and primers that are complementary to the linked adapters are given into an assay.

4. Dideoxynucleotides (ddNTP), which are similar to dNTPs except for lacking the 3'-hydroxyl group, that are fluorescently labeled with base-specific colors are added in low concentration.

5. DNA polymerase is added to initiate the PCR starting at the DNA primers.

6. The PCR terminates after a ddNTP is incorporated by the DNA polymerase.

7. The resulting sequence fragments have different length and are labeled by different colors attached to their 3'-base.

8. The sequence of a template fragment is obtained by separating all sequenced fragments according to their length with a Capillary Array Electrophoresis (CAE) and identifying their labeled 3'-bases.

As a result, the nucleotide sequence of each fragment in the library is determined. These sequences are called *reads*. The sequence of each read is given by the bases that correspond to the highest peak in the intensity curve at the respective positions. In addition, quality values are calculated from the curve that reflect the error probability of each base in a read.

In order to gain information about the relative order of the generated reads, the method can be extended to produce *mate-pairs*, i.e., pairs of reads for which the orientation and approximate distance to each other are known.

2.1.2. Second Generation Sequencing Technologies

With the rise of a new generation of sequencing technologies, sequencing has become cheaper and less time consuming [Met09]. Due to the dramatically increased throughput of the sequencing technologies, more projects that study the genome variation within and across species or the transcriptome of a species can be realized.

The read sequences generated by the new technologies are shorter and tend to be more erroneous than Sanger reads. The higher error rate can be compensated to a certain extent by the higher amount of data that is available due to a higher sequencing depth. The shorter read length, however, yields several new issues and problems for the processing and analysis of sequencing data.

Like Sanger sequencing, most of the SGS technologies follow the *sequencing-by-synthesis* approach, i.e., they synthesize a strand of a DNA molecule according to a template strand and record the order of the incorporated nucleotides. Here, the recording becomes feasible using a light signal that is emitted upon incorporation and that is recorded by a camera. The whole cycle can easily be run in parallel,

synthesizing different fragments at the same time. The major difference to Sanger sequencing is the use of reversible termination for DNA synthesis.

In the next sections, we will present two widely used SGS technologies in detail, pyrosequencing by Roche/454 Life Sciences [LLT+03, MEA+05] and sequencing using cyclic reversible termination by Illumina [Ben06, BBS+08]. Finally, we will briefly describe other recent sequencing technologies.

Pyrosequencing (Roche/454 Life Sciences)

The FLX Genome Sequencer by Roche/454 Life Sciences is based on the pyrosequencing technology, which detects the release of pyrophosphates when a nucleotide is incorporated by the DNA polymerase. Each time a pyrophosphate is released, it is converted into visible light using a series of enzymatic reactions. The sequence of emitted light that codes for the nucleotide sequence of the synthesized template is recorded.

At first, the DNA of the target organism is randomly broken into fragments to create templates for the amplification. The main workflow continues as follows (see Figure 2.2):

1. Single-stranded fragments are ligated to universal adapters at both ends.
2. Each fragment is captured onto one bead (favoring one fragment per bead) in a water-oil-emulsion.
3. Amplification of each fragment by emulsion PCR such that populations of identical template fragments are bound to each bead.
4. Beads are given individually into arrays of wells (PicoTiterPlate).
5. Sequencing-by-synthesis of the template fragments using single-nucleotide addition:
   a) One type of dNTPs is added per step.
   b) From the synthesized dNTP molecule a pyrophosphate is cleaved that converts provided APS (adenosine-5'-phosphosulfate) to ATP.
   c) With the help of Luciferase, ATP is transformed into visible light that is emitted proportional to the number of identical nucleotides that are added simultaneously.
   d) For the next nucleotide, the remaining dNTPs and ATPs are degraded.

The templates are amplified since imaging systems have problems with the detection of single fluorescent signals. The light signal detected during the synthesis is a consensus signal that is emitted simultaneously by all identical templates. Due to incomplete extension or addition of multiple nucleotides to the identical templates, the fluorescent signal is dephased occasionally. The order of light peaks and their intensities determine the original sequence of the synthesized fragment.

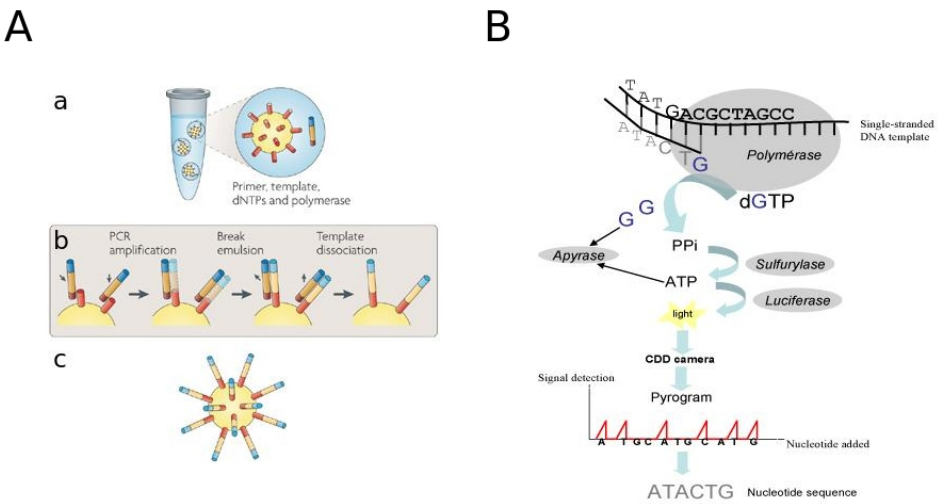

Figure 2.2.: (A) **Pyrosequencing - DNA Amplification Step.** The DNA fragments are captured by beads and these bead-DNA complexes are encapsulated into single aqueous droplets. Within these droplets, the template fragments are amplified by emulsion PCR. Several thousand copies of the same template sequence are bound to each bead. (B) **Pyrosequencing - Sequencing Step.** A DNA fragment is synthesized from the single stranded DNA template by the DNA polymerase. During that process, dNTPs are added one at a time (here dGNTPs is added). With the incorporation of a nucleotide a pyrophosphate (PPi) is released. The provided sulfurylase quantitatively converts PPi to ATP. A light signal is emitted that is produced with the catalysator luciferase in presence of ATP. The signal is detected by a charge coupled device camera and represented as a peak in a pyrogram. The nucleotide degrading enzyme apyrase continuously degrades unincorporated dNTPs and ATP. In the next step, the process starts from the beginning by adding the next dNTP. Finally, the whole nucleotide sequence of the synthesized DNA strand is inferred from the signal peaks of the produced pyrogram. Image (A) is from Metzker [Met09] and image (B) is from Armougom and Raoult [AR09]

Homopolymer DNA segments, i.e., regions with multiple consecutive copies of a single base, will result in higher intensities depending on their copy number. Since this is difficult to measure exactly for longer homopolymer regions, the resolution of homopolymer DNA segments is often erroneous. The most common error type are insertions, the second most are deletions. The mean read length is 330 bp while 1.29 Gb can be produced per day with a single machine.

Sequencing Using Cyclic Reversible Termination (Illumina)

Like the technology of 454/Roche, Illumina employs the sequencing-by-synthesis approach for its sequencing system Genome Analyzer. Instead of adding just one type of nucleotide, all types are provided to the sequencing reaction at the same time. A reversible terminator at each nucleotide prevents the incorporation of several nucleotides per step. The nucleotides are labeled with different fluoroscentic signals such that their specific light spectrum can be recorded after incorporation. The method works as follows (see Figure 2.3):

1. Adapter linkers are ligated onto both ends of the DNA fragment.
2. Fragments are tethered in great distance to a glass plate by the flexible linker at their 5'-end and will eventually hybridize with primers linked to the plate that are complementary to the 3'-adapter, forming a bridge on the glass plate.
3. Amplification via bridge PCR produces clusters of identical fragments on the plate.
4. Sequencing-by-synthesis with reversible termination:
   a) dNTPs are flowed over the plate simultaneously and only one single nucleotide is synthesized since the 3'-end of the dNTPs are blocked.
   b) Not incorporated nucleotides are washed away.
   c) Incorporated bases are detected via their fluorescent labels.
   d) Fluorescent dye and 3'-block is removed and the process is repeated.

Substitutions are the most frequent error type of the Genome Analyzer. Also, an under-representation of AT-rich and GC-rich regions has been reported. The mean read length is about 75 bp or 100 bp. 4.5 Gb of reads can be produced per day with a single machine. Currently, the Genome Analyzer dominates the market of SGS due to its good trade-off between costs, reads length and accuracy.

Remarks and Outlook

All SGS technologies can also provide mate-pair information, i.e., two reads whose approximate distance and orientation is known. The distance between them is

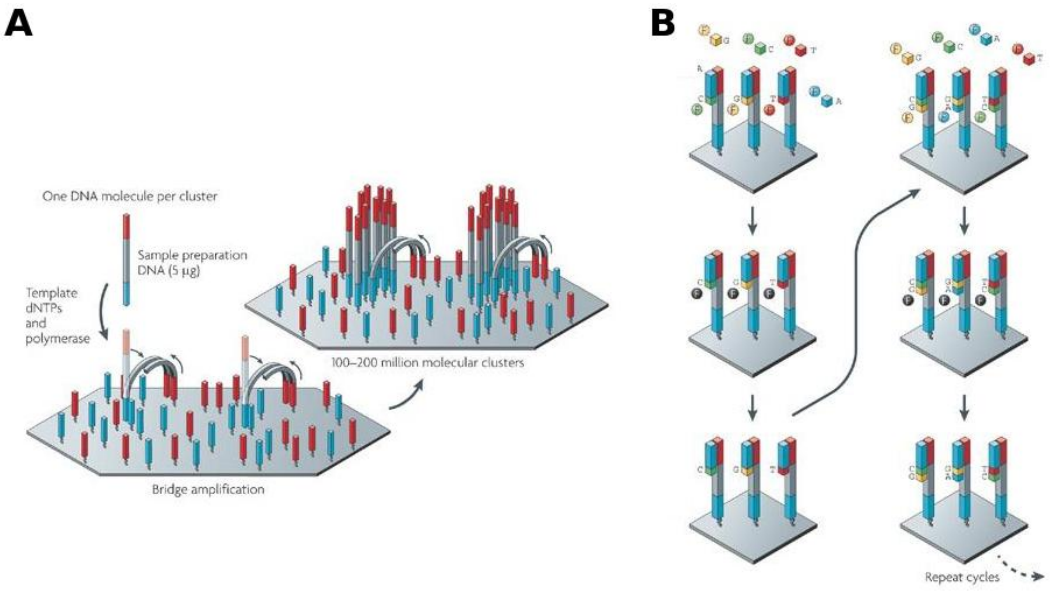

Figure 2.3.: (A) For the amplification with Illumina's Genome Analyzer, forward and reverse primers are covalently attached to a glass plate in the beginning. The single-stranded DNA templates are hybridized to adapters at both ends. The templates are attached to the slide by hybridizing one of their adapters to a complementary primer on the slide. The templates are extended such that they build double-stranded bridges with immediately adjacent primers on the slide. These bridges are denatured resulting again in single-stranded templates tethered to the slide. The process is repeated until a cluster of identical fragments is built for each template. (B) The four-color cyclic reversible termination method is shown, starting with the incorporation of nucleotides in the complementary strand to the template sequences, which are attached to the glass slide. Incorporated nucleotides are labeled with different dyes that are imaged after all unincorporated nucleotides have been washed off. Following imaging, the fluorescent dyes are cleaved and the 3'-hydroxyl group is regenerated such that the synthesized sequence can be elongated. Then, the cycle repeats with the incorporation of the next labeled nucleotides that are blocked at their 3'-end. The image is from Metzker [Met09].

called *insert size*. To distinguish between pairs with a longer (at least 1 kbp) and a shorter (at most 1 kbp) insert size, the terms *mate-pair* or *paired-end* reads were introduced, respectively. We will use the term mate-pair in this work for both types.

In the following, we briefly describe other SGS technologies and recent developments towards a third generation of sequencing technologies. Their sequencing methods differ in the template preparation and the applied sequencing method.

Another SGS method is *sequencing-by-ligation*, in which DNA polymerase is replaced by DNA ligase [SPR+05]. The method is commercialized in a sequencing platform, called SOLiD (support oligonucleotide ligation detection), by Applied Biosystem. The method became available in 2008 but is not used as often as the previously described technologies of Roche/454 Life Sciences and Illumina.

The sequencing methods are steadily improving with increased read lengths and decreasing sequencing costs. Furthermore, more mate-pair libraries with different insert sizes are provided. Assembly approaches benefit from various insert sizes and from longer read lengths, which will be discussed in Section 2.3. One technology that is already commercially distributed is HeliScope of Helicos BioSciences, which uses single-molecular templates instead of amplified templates [HBB+08]. This allows for an unbiased quantification of sequenced RNA molecules. The platform applies a one-color cyclic reversible termination method for sequencing.

Another promising method is *real-time sequencing* developed by Pacific Biosciences [EFG+09]. DNA synthesis is performed in real-time and labeled nucleotides are measured at the moment of incorporation in the synthesized DNA strand. With a fixation of the DNA polymerase, the light signals are emitted within a reduced observation volume. Thus, the signals can be recorded in real-time and reversible terminators are no longer required.

The most recent technology is ion torrent. The technology uses the fact that during nucleotide incorporation a hydrogen ion is released. The template DNA fragments are hold in microwells of an array such that the process can be performed in parallel. The ion-sensitive layer and sensor, which are located beneath the array, detect the release of a hydrogen ion upon nucleotide incorporation. However, the methods still requires PCR amplification and terminates the sequencing process after a nucleotide is incorporated [STK10].

In addition, *nanopore sequencing*, which has been under development since 1995, is currently improved by Oxford Nanopore. As the technology of Pacific Biosciences, this technology uses also single-molecule DNA templates. In this sequencing method, nanopores are utilized to detect and analyze nucleotides of the templates. In one approach, individual nucleotides on the DNA template are identified as it passes through a protein nanopore. There are still some problems that have to be solved before this technology, which would be likely to be the cheapest of the third generation sequencing technologies, will become commercially available [Rus09].

2.2. Mapping-Consensus Approach

One of the challenges introduced by new sequencing technologies is the efficient alignment of large amounts of short reads onto a reference genome. This process is also called *mapping*.

The mapping problem is certainly not new and there already existed many tools that perform mappings for Sanger data. Also, conventional software tools such as BLAST can be used for mapping to a certain degree. However, since it was not designed for mapping short sequences onto a single reference sequence, BLAST will take up to thousands of CPU hours to align the number of reads that are produced in a typical sequencing project [TS09]. Due to limitations of existing tools which result in long runtimes, the methods were adapted to the demands of the new data. Mainly, the decrease in read length and the much greater amount of data had to be considered, but also new sequence characteristics and specific error distributions.

New alignment tools follow more or less the following workflow: First, possible alignment positions of the reads in the reference genome are detected. Next, read sequences are aligned against these regions and the consensus sequence of overlapping aligned reads is calculated.

There exist two indexing strategies that can speed up the process of assigning reads to possible alignment regions: *hash table* indexing or indexing with the *Burrows Wheeler Transformation* (BWT) [FB09]. In the hash table approach, either the reference genome or the set of sequence reads are indexed using *seeds* or *spaced seeds*. While a seed is a sequence region that has to match exactly, a spaced-seed allows mismatches at specified positions. Thus, not all characters of the seed have to be compared but only the positions that are required to match. With a hash function, reads are associated to the seeds that match their sequence. For each seed, the associated read identifiers are stored in the hash table. The matched seeds between reference genome and read sequences represent potential alignment regions of the reads to the reference. By indexing the reads, the reads are associated with positions in the reference genome and can be quickly looked up with the help of the index. The same is possible for the reference genome if it is indexed. Then, the set of reads is used to quickly scan the hash table of the reference genome. Short read alignment tools based on hash tables are MAQ [LRD08], SOAP [LLKW08], ELAND [BBS+08], SHRiMP [RLD+09], ZOOM [LZZ+08], BFAST [HMN09], MOSAIK [Mos] and GenomeMapper [SHO+09]. In contrast, tools like Bowtie [LTPS09], BWA [LD10] and SOAP2 [LYL+09] create a BWT of the reference genome and index this transformation using a suffix array. This index is called *FM index* or *compressed suffix array*. Possible alignment regions of reads in the reference genome are detected by scanning for seeds of the reads in the FM index. The FM index takes advantage from the compression of the reference genome by the BWT. The FM index is often smaller than the index of the original reference genome, while it still allows to search for substrings at the same level of speed. Consequently, short read mappers

based on the BWT are much faster than hash-based methods at the same level of sensitivity.

After detecting possible alignment positions, accurate alignments of the reads to the reference genome are calculated at these positions by a gapped or ungapped version of the Smith-Waterman algorithm. In addition, quality values of the sequenced bases can be taken into account. To obtain reliable alignment positions of the reads, a threshold for the number of allowed mismatches is defined. For each read, the alignments with the highest alignment score, representing the quality of the alignment, are considered as *best alignments*. If there exists only one best alignment for a read, the read can be uniquely assigned to the reference sequence. If there exist several best alignments, the read is marked as belonging to a repetitive region. An alignment of mate-pair reads to the reference sequence can only be classified as correct, if both reads align in a correct orientation to each other and show a distance that matches their insert size.

Finally, the consensus sequence is determined. Either the determination is an integrated part of the mapping software like in SOAPsnp [LLF+09] or it is handled by a separate software like SAMtools [LHW+09]. For reads that are uniquely aligned onto the reference sequence, the consensus sequence can be determined to detect reliable differences between the target genome and the reference genome such as SNPs and short indels. Quality scores of the consensus bases are taken into account. Thresholds for the reliability of bases in the consensus sequence can be adjusted to the type of read data used.

The mapping-consensus strategy has some limitations in the detection of longer insertions and highly polymorphic regions in the target genome. Using conventional mapping software, only a limited amount of these regions are detected depending on their length. Thus, more sophisticated approaches are required for covering long insertions or highly polymorphic regions. Moreover, rearrangements like reversals, i.e., regions in the target sequences that are reversed compared to the reference sequence, can not be found by mapping-consensus approaches.

2.3. De Novo Assembly Approach

In contrast to the mapping-consensus approach, *de novo* assembly does not utilize a reference sequence. Reads are assembled in order to reconstruct the original sequence of the target genome. The read sequences are overlapped and merged with each other into a colinear arrangement without any additional sequence information. In this process, mate-pair information can be utilized.

Initially, the *de novo* assembly problem was formulated as the *shortest superstring problem* [Pop04].

Definition 1. Shortest Superstring Problem (SSP)
For a given set S of read sequences, the shortest string that contains each read of S as substring is the shortest superstring.

The problem of finding the shortest superstring is known to be NP-hard, which means that for this problem no exact solution can be found in polynomial time using a deterministic algorithm, unless P=NP. This simplified definition of the assembly problem does not consider repetitive regions and sequencing errors. Thus, even the exact solution for the SSP might not correspond to a correct solution of the assembly problem.

Three approaches have been developed to approximate the assembly problem: A greedy approach, the *overlap-layout-consensus* approach, and the *de Bruijn graph approach*. Several tools based on one of these approaches have been designed to assemble Sanger reads. However, tools that have been developed for Sanger reads cannot be directly applied to short read data without adjusting their algorithms to the shorter read length, higher coverage and the specific error characteristics of SGS technologies.

2.3.1. Greedy Assembly Approach

When the assembly problem came up, several greedy algorithms [PSTU73, TY02, AS98, KS05] were developed that worked according to Algorithm 2.3.1.

Algorithm 2.3.1: GREEDYASSEMBLYSANGER(*readSequences*)

```
contigs ← readSequences
while there exist two sequences in contigs that overlap with each other
do { contigPair ← randomly choose two sequences in contigs
                  with best overlap
   { newContig ← merged contigPair according to alignment
   { contigs.delete(contigPair)
   { contigs.add(newContig)
return (contigs)
```

Due to the complex structure of most genomes and to errors in the read sequences, the genome sequence can only be approximated by a greedy algorithm. Nevertheless, most sequencing errors in the reads can be handled in the greedy strategy by allowing mismatches in the overlap alignments of reads. Unfortunately, there exists no extension to the algorithm to handle repeat regions that occur in the target sequence. Regions that are adjacent to such repeats are likely to be combined in a wrong order during the greedy assembly process, leading to false contigs.

Algorithm 2.3.2: GREEDYASSEMBLYSHORT(*reads*)

prefixTree ← Prefix Tree for *reads*, 5'-prefixes of read sequences and their reversed complements
contigs ← empty List
newContig ← empty String
while there exists a sequence in *reads* that is not contained in *contigs*
do
- *newContig* ← randomly choose sequence in *reads* that is not contained in *contigs*
- EXTENDCONTIGAT3'END(reads,newContig,prefixTree)
- *newContig* ← *newContig.reversedComplement*()
- EXTENDCONTIGAT3'END(reads,newContig,prefixTree)
- *contigs.add*(*newContig*)

return (*contigs*)

Algorithm 2.3.3: EXTENDCONTIGAT3'END(*reads*, *newContig*, *pTree*)

nextReads ← utilization of *pTree* to find a set of sequences in *reads* such that their 5' ends match exactly to the 3' end of *newContig* by maximizing the length of the exact matching
while *nextReads* is not empty and contains only similar sequences
do
- newContig← extension of *newContig* by multiple alignment of *nextReads*
- *reads.delete*(*nextReads*)
- *nextReads* ← find a set of reads such that their 5' ends match exactly to the 3' end of *newContig* by maximizing the length of the exact matching using *pTree*

return (*newContig*)

With the rise of short read data, a similar greedy strategy has been developed, see Algorithm 2.3.2. To deal with the large number of reads, a prefix tree that organizes the read sequences by their first bases is built such that potential start regions of overlaps between reads can be detected efficiently. Sequencing errors in the read data are taken into account by favoring read sequences with a high sequencing depth for extending a contig. Mis-assemblies caused by repeats are avoided by cutting the assembly if overlapping reads do not show a similar elongation-sequence. Such reads could arise from different copies of a repeat. Often they match at one end, indicating the repeat region, but have dissimilar regions at the other end. Utilizing this approach often results in a large number of very small contigs and a long runtime, for example 6 h to 19 h for a single bacterial genome assembly. Variants of this algorithm are implemented in the assembly tools SSAKE [WSJH07], VCAKE [JRB+07] and SHARCGS [DLBH07].

2.3.2. Overlap-Layout-Consensus Approach

In order to address problems in assembly like sequencing errors and the resolution of repeats, the local nature of the greedy strategy has to be overcome. A more sophisticated approach follows the overlap-layout-consensus paradigm. It has been introduced for Sanger reads by Peltola *et al.* [PSU84] in 1984 and further developed by Kececioglu and Myers [KM95, Mye95] in 1995. Here, the overlap information between reads is represented in a graph structure, the *overlap graph*. In such a graph, each read is represented by a vertex and each overlap between reads is represented by an edge between the corresponding vertices, see Figure 2.4 C. With this representation, the assembly problem can be formulated as the problem of finding a *Hamiltonian Path* [Pop04].

Definition 2. Hamiltonian Path
In a given graph, a Hamiltonian Path visits each vertex exactly once.

The problem of determining the existence of a Hamiltonian Path is NP-complete. Problems that are NP-complete lie in NP and any NP-hard problem can be reduced to any of these problems. For a problem that lies in NP a provided solution can be verified in polynomial time. For a problem that is NP-complete an exact solution cannot be found in polynomial time using a deterministic algorithm, unless P=NP.

Similar to the SSP, solving the assembly problem by finding a Hamiltonian Path does not consider sequencing errors and repeat regions in the target genome. However, with the overlap-layout-consensus approach, good approximations of the real world assembly problem can be obtained by finding paths in the overlap graph. Paths that are induced by sequencing errors or repeats have to be cut and perhaps re-linked in the graph. For the final paths, the represented consensus sequence is calculated and reported as contig. In general, the overlap-layout-consensus approach executes the following three steps:

1. **Overlap phase:** Overlap alignments between read sequences are determined and represented in a graph structure.

2. **Layout phase:** A path is selected representing an alignment that assembles the reads in a linear order.

3. **Consensus phase:** For the final path the consensus sequences, which correspond to the most likely original sequence of the target genome, is calculated.

This workflow is illustrated in Figure 2.4. In the overlap phase, pairs of overlapping reads are determined by calculating overlap alignments. In a naïve approach, all read sequences are compared with each other to detect all pairwise overlap alignments. To reduce the number of comparisons, a filtering step is often applied revealing pairs of reads that possibly overlap. Often, reads share an identical subsequence, a k-mer, if they overlap with each other. These k-mers are detected

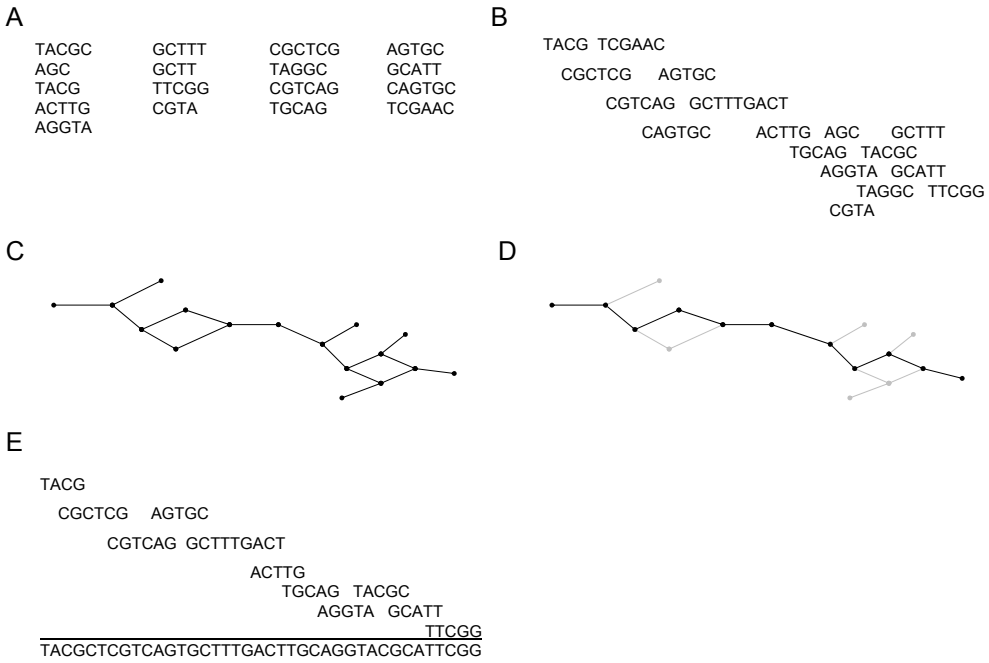

Figure 2.4.: The workflow of the overlap-layout-consensus approach is shown using a simple example. (A) Read sequences are obtained from a sequencing platform. In this example, the read lengths vary between three and five bp. (B) In the overlap phase, overlaps between the read sequences are determined. In a filtering step, which is not shown here, pairs of reads that share a common k-mer are determined . For these candidates, the pairwise overlaps are determined. (C) The overlap phase results in a representation of the overlap information as an overlap graph. (D) In the layout phase, cycles in the graph that arise from repeats, sequencing errors and polymorphisms are handled. Here, the overlap graph shows two cycles arising from sequencing errors in the reads CAGAGC and TATGC. These cycles are handled by deciding for one of their paths. Furthermore, short paths arising from reads that overlap with only one other read are deleted. (D) Finally, unique paths are extracted from the overlap graph. (E) In the consensus phase, for each extracted path the consensus sequence of the underlying read sequences is determined and reported as a contig.

using an indexing technique like a suffix-array. Only for pairs that share a k-mer, an overlap alignment is determined.

To further reduce the time for the calculation of the overlap alignments, a *banded alignment* algorithm can be applied. It reduces the time complexity of an alignment from quadratic to nearly linear time by avoiding alignments that have a longer gap in either sequence. This is managed by constraining the calculations of the dynamic programming matrix to a diagonal band. The position of the band can be determined by the positions of the equal k-mers that have been detected by means of indexing in the beginning of the overlap phase.

For the actual overlap alignment calculation, constraints are set such as the maximum number of mismatches, the minimum alignment length and gap bounds. Unfortunately, there still exist false overlaps due to erroneous read sequences or accidentally aligned reads that belong to different copies of a repeat. False alignments can be induced by repeats, since reads that contain the same part of a repeat sequence do not necessarily belong to the same copy of the repeat in the target genome. To reduce the number of false overlaps, quality values that are given for the read sequences can be used to penalize mismatches between high quality bases stronger than mismatches between bases of lower quality. Not all false overlaps can be detected with this approach and, thus, have to be handled later in the workflow. In addition, it has to be considered that read sequences that originate from the same region in the target genome are almost identical except for sequencing errors. If they do not have the same length, one sequence might be contained in another sequence. Thus, we have to merge these reads. Finally, the calculated overlap information is represented in an *overlap graph*.

In the layout phase, final paths are selected from the overlap graph. The paths represent alignments of reads in which the reads are linearly ordered. Different optimization strategies can be chosen to select edges for these paths, e.g., the score of the alignments, their quality or more sophisticated alignment statistics. The layout problem has been shown to be NP-complete for the different optimization targets [PSU84, KM95].

The size of the overlap graph tends to grow drastically, even for Sanger reads, to up to tens of thousands or tens of millions of vertices for bacterial or mammalian-sized genomes, respectively. Nevertheless, there exist several implementations of the layout phase that calculate good approximations. Most of these greedy methods transform the overlap graph in a graph of lower complexity by simplifying subgraphs in the overlap graph for which the layout problem can be easily solved [Mye95]. Often, these subgraphs represent regions between repeats in the genome sequence. Then, more sophisticated approaches are applied to the reduced graph by taking advantage of mate-pair information. The graph can be further simplified by placing mate-pairs such that their distance and orientation in the graph is consistent with their insert size and their real orientation to each other. Variations of this idea are implemented in several assembly tools like the Celera Assembler [MSD+00] or Arachne [BJS+02].

Often, several mate-pair libraries with different insert sizes are available, de-

pending on the technique the library was produced with. Repeat regions can only be spanned by mate-pairs that have an appropriate insert size. Thus, different mate-pair libraries are required for handling repeats of different sizes. In addition to mate-information, the sequence depth offers the possibility to detect subgraphs in the overlap graph that belong to repetitive regions. Usually, the sequencing depth at a repetitive region depends on the number of its copies in the target genome and is by this factor higher than the mean sequencing depth of all other regions in the genome. The reads of each copy of a repeat align to each other and, thus, will collapse into the same subgraph in the overlap graph. In order to optimize the runtime, assembly tools often skip the assembly of longer repeat regions by omitting all related overlap alignments.

In the consensus phase, the consensus sequence is calculated for the aligned reads that are represented by the final selected paths. The consensus sequence is determined from a multiple alignment that is constructed for the reads such that the overall alignment score is maximized. The multiple alignment is often only an approximation since an exact solution cannot be determined efficiently. One heuristic to create a multiple alignment uses the already provided pairwise alignments of reads. Often, only a subset of the pairwise alignments is used such that no redundant alignment information is contained. More sophisticated approaches calculate a multiple alignment of the reads iteratively by adding a read sequence to the multiple alignment in each step. The placement of the newly inserted read is guided by the already available pairwise alignments. A similar approach by Anson and Myers [AM97] improves an initial multiple alignment of the reads by an iterative procedure. In each step, a read sequence is deleted from the multiple alignment and re-aligned again. The process terminates when the multiple alignment cannot be improved any further. Other algorithms take the overlap alignments of the reads as a first layout of the multiple alignment and solely optimize local regions that contain mismatches. In order to obtain an appropriate multiple alignment, the consensus is calculated for each position by choosing the base with the highest relative frequency.

Besides the already mentioned Celera Assembler and Arachne, there exist various other sophisticated assembly tools that follow the overlap-layout-consensus paradigm like Atlas [HCD+04], CAP3 [HM99] , PCAP [HWA+03], Phrap [DLBM07], Phusion [MN03] and for short read data Edena [HFF+08] and the here presented LOCAS. Until now, this approach has not been adapted to SGS data very often since the number of pair-wise comparisons in the overlap phase is huge due to the large amount of data. Furthermore, the reliability of short overlap alignments that are required for short reads is not very high. At present, the state-of-the-art approach for developing assembly tools for SGS data is the *de Bruijn graph approach.*

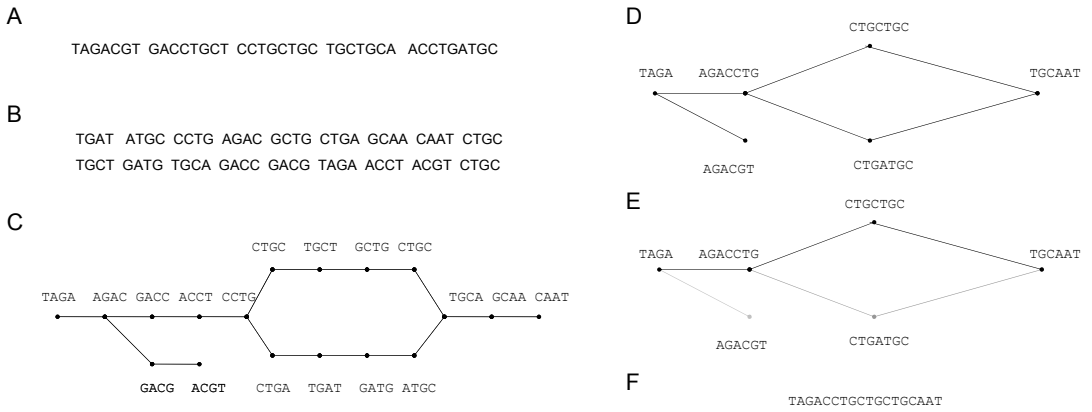

Figure 2.5.: An example de Bruijn graph assembly is shown. (A) Sequence reads are obtained from a sequencing platform. In this example, the read lengths vary between seven and nine bp. (B+C) In the next step, the k-mers, in this example 4-mers, are hashed, k-mers and their overlaps are represented in a de Bruijn graph. (D) The graph is simplified such that each path that has no branching vertices is reduced to a single vertex. From now on, vertices can represent sequences of different lengths. (E) In the next step, the graph is reduced by handling cycles that arise from sequencing errors. In addition, short paths resulting from k-mers that do not overlap at both sides are deleted. Also, cycles induced by repeats are handled, which is not shown here. Finally, a graph is obtained that approximates the genome sequence. (F) The sequence is determined via the consensus sequences of the linear paths that can be extracted from the graph.

2.3.3. De Bruijn Graph Approach

Idury and Waterman [IW95] were the first to introduce the de Bruijn graph approach for sequence assembly. All overlapping k-mers, i.e., k-mers that overlap exactly with a length of $k-1$, in the set of sequence reads are detected using an index structure. This information is converted into a *de Bruijn* graph by introducing a vertex for each k-mer and an edge between two vertices if the respective k-mers overlap in $k-1$ positions. Using this model, the sequence of the original genome can be approximated by an *Eulerian Path* in the de Bruijn graph [Pop04].

Definition 3. Eulerian Path
In a given graph, a Eulerian Path visits each edge of the graph exactly once.

The Eulerian Path problem can be solved in linear time and returns a solution to the assembly problem assuming error-free read sequences and a target genome without repetitive regions. In order to consider also repeats and sequencing errors in the assembly solution, a variation of the Eulerian Path has to be calculated. This path has to traverse subgraphs that represent repeats several times. Subgraphs that belong to sequencing errors must not be included in the final path. Furthermore, the problem of selecting such a path becomes even harder. For example, each sequencing error in a read leads to a number of false k-mers, which increases the number of vertices in the de Bruijn graph. It also may happen that k-mers that are not adjacent to each other in an input read are represented by connected vertices in the graph. Thus, incorrect sequence information is represented in the graph.

The first implementation of the approach was released by Pevzner *et al.* [PTW01b, PT01, PTW01a] addressing several of the above issues. In Figure 2.5, the workflow of the de Bruijn graph approach is illustrated. In addition, Pevzner *et al.* developed an error correction method. Input reads are corrected by counting the frequency of their k-mers in all read sequences and replacing less frequent k-mers by similar k-mers of a high frequency. Furthermore, reads are used to reduce the complexity of the de Bruijn graph. On one hand, edges between vertices that represent k-mers that are not present in one read sequence are deleted. On the other hand, small repetitive regions can be handled by utilizing original read sequences that fully contain these repeats. In addition, Pevzner and Tang developed a sophisticated method that handles longer repeats by using mate-pair information [PT01].

A key advantage of the de Bruijn approach is the construction time, which is linear in the number of reads, while the overlap-layout-consensus approach has a quadratic runtime using a naïve implementation.

The de Bruijn approach has been first adapted to SGS data in the software tool Newbler (Roche's 454 assembler), which assembles reads produced by the 454/Roche technology. Several assembly tools that are compatible with reads of the Illumina technology followed like VELVET [ZB08], EULER-SR [CP08], ABySS [SWJ+09], ALLPATH [BMK+08, MPG+09], SOAPdenovo [LLZ+09] and

recently Contrail [SSK+]. They all use a de Bruijn graph but their approaches for error treatment and for taking advantage of mate-pair information differ.

2.3.4. Prospects of De Novo Assembly with Short Reads

The natural limits of a short read assembly under the assumption of error-free reads have been investigated by Whiteford *et al.* [WHW+05]. Excellent results could be produced for bacterial genomes even with a read length of 30 bp. For *Escherichia coli*, 75% of the genome was covered by contigs with at least 10,000 bp and 96% of the genes were assembled within single contigs. For eukaryotic genomes, 51% of the genome could be assembled into contigs with at least 10,000 bp using reads of length 50 bp.

In *de novo* assembly of complex and large sized genomes, large amounts of memory are required in the assembly process. With the de Bruijn graph approach, which is used by most current assemblers, the data is presented in a compressed fashion. However, additional data structures are required for assigning reads to the graph, which produces a memory overhead for larger genomes. The high demand for memory has been addressed in more sophisticated assembly approaches such as SOAPdenovo by using compressed data structures that can be retained to disk. To take advantage of multiple computers, a parallelized, MPI-cluster-based approach is used by the assembler ABySS. Further, there exist homology-guided approaches that use a reference sequence to partition the assembly problem into smaller sub-problems in order to save memory. This approach will be discussed in Section 2.4.

Repetitive regions still yield the main problems for assembly. Different libraries of mate-pairs are required to handle repeats of different sizes. Consequently, at least a portion of the read data should be provided with mate-pair libraries of short and long insert sizes. The quality of the assembly will benefit from this data. Nevertheless, the assembly depends strongly on the kind of target genome. While bacterial genomes have often only a small number of repetitive regions with a length of at most 200 bp, eukaryotic genomes such as the human genome are more complex. The repeat length of these genomes depends on the appearance of active SINE and LINE transposable elements, which have a length of 500 bp to 1 kbp and about 4 kbp, respectively. The longer insert sizes of mate-pairs that have recently been provided by the SGS technologies will facilitate *de novo* assemblies of large complex genomes.

2.3.5. Scaffolding

Contigs produced by *de novo* assembly need to be ordered and oriented to each other to obtain their correct position in the target genome. This is done in a similar fashion to the mapping-consensus approach in a process called *scaffolding*.

Constraints defined by given mate-pairs are used in the scaffolding process by ordering contigs according to adjacency information of their mate-pairs. The

orientation of mated reads to each other yields restrictions to the orientation of the respective contigs. With the approximate distance between two mate reads, the contigs can be ordered and their distance to each other can be determined. Often, multiple libraries with mate-pairs of different insert sizes are generated, which provides even more constraints.

Although some assemblers include scaffolding, there exist stand-alone scaffolding tools like Bambus [PKS04]. It was originally developed for Sanger reads but is also widely used for SGS data. The tool makes use of mate-pair information and can utilize a complete genome sequence of a related organism to guide the contig placement.

Further approaches for scaffolding are contig overlapping and the use of gene synteny. The first uses additional information on detected overlaps between contigs to order them and extract information on their orientation. Knowledge of gene synteny is used to scaffold contigs that contain co-located genes. Therefore, genes are detected within contigs that usually occur in clusters in most organisms. Contigs containing the same gene sequence or genes from one cluster are placed close to each other in the scaffolding solution.

2.4. Homology-Guided Assembly Approach

Similar to the mapping-consensus approach, *homology-guided* or *comparative* assembly approaches exploit available reference genomes from the same or closely related species. Often, different strains of the same species are sequenced in order to identify polymorphisms and indels [RSP+02]. The comparative assembly strategy works best when the genomes of two species are more than 90% identical [PS08].

Homology-guided assembly adapts from both, the mapping-consensus approach as well as *de novo* assembly. The traditional overlap-layout-consensus approach is transformed in a mapping-layout-consensus approach. The layout phase additionally handles indels and rearrangements that occur between target and reference genome. Usually, these regions complicate the mapping of the reads to the reference since reads may only partially match the reference genome or reads may match non-adjacent regions of the reference. The overall workflow is as follows:

1. **Read mapping:** Align reads to the reference genome and utilize possible mate-pair information.

2. **Layout refinement:** Rearrange reads to handle indels, divergent positions and genome rearrangements.

3. **Consensus phase:** Generate consensus sequence for reads of the final layout.

4. **Scaffolding (optional):** Orientate and order contigs.

Insertions appearing in the target genome are handled in the layout phase. Reads belonging entirely to these regions will not align to the reference genome while reads that align only partially with the insert region can align to the reference genome. The assembly will stop at the point of this insertion, resulting in two separate contigs. In the case of smaller insertions that have at most the length of a read, some reads from the edge of the insertion will align to the reference genome. These insertion regions can be resolved with the homology-guided assembly approach and are reported as a single contig.

An advantage of placing reads on a reference genome instead of overlapping them with each other is that more sequencing errors can be handled and, thus, a considerable lower amount of read data is required to perform assembly. Also, regions that lead to major problems in *de novo* assembly like repeats can be handled more easily. The ability to detect expansions and contractions of long tandem repeats is increased. Tandem repeats are repeated pattern sequences of at least two nucleotides that are directly adjacent to each other. Moreover, isolated repeats do not cause breaks in the assembly as they would do in a *de novo* strategy. Reads belonging to repeats are randomly assigned to one copy of the repeat to which they have a high quality alignment in the reference sequence [PS08].

One of the first comparative assemblers was AMOScmp [PPDS04]. It was first created for Sanger reads and later adjusted to SGS data (AMOScmp-shortReads, unpublished). Another tool developed for short read data is Crossbow [LSL+09], a software pipeline for whole genome resequencing analysis. It utilizes the short read aligner Bowtie and the genotyper SOAPsnp and applies them in a parallel fashion exploiting multiple computers and CPUs wherever possible.

3. An Extended Homology-Guided Assembly Approach (SHORE)

The homology-guided assembly approach uses a reference sequence to guide the assembly process. Homologous regions are assembled and short polymorphic regions and indels can be resolved with this approach, see Section 2.4. However, longer insertion and highly polymorphic regions still raise problems.

In this chapter, we introduce a sophisticated method that extends the traditional homology-guided assembly approach to additionally address the assembly of longer insertion and highly polymorphic regions. The main steps of the workflow are a mapping of all reads onto the reference and a pooling of all *left-over reads*, which are reads that could not be mapped onto the reference sequence. In the following, the mapped reads are assembled by incorporating left-over reads. This leads to a new kind of assembly problem, which we call *reassembly problem*. Two kinds of read sets have to be assembled. While the whole set of mapped reads has to be assembled following the ordering of their mapping positions, only some left-over reads have to be incorporated in the assembly. A left-over read is only incorporated if it has a high quality overlap with a mapped read in order to recruit only left-over reads that belong to the respective region. Though the mapped reads have already an order defined by their mapping positions they can be rearranged in the reassembly process. This happens if a deletion or small insertion occurred in the target genome regarding the reference genome or a better placement of a read becomes reasonable due to a replacement or an incorporation of other reads. We will discuss methods for reassembly in Chapter 4 and Chapter 5, while we present the main workflow of the homology-guided assembly approach in this chapter.

3.1. Workflow

After the read data is filtered for erroneous reads, the high quality reads are mapped against a reference genome to receive a first ordering. Some reads belonging to homologous regions can contain a smaller number of SNPs and short indels and still map onto the reference genome, while reads that belong to highly polymorphic regions or insertion regions in the target genome will not align to the reference. This is illustrated in Figure 3.1.

In the following, mapped reads are *reassembled* in order to accurately reveal deletions and small insertions and additionally incorporate some left-over reads that belong to longer highly polymorphic and longer insertion regions. The left-

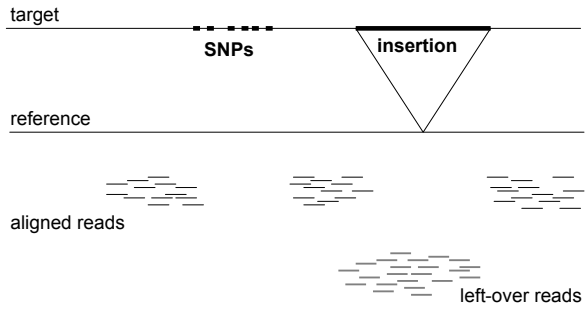

Figure 3.1.: The target genome differs from the reference sequence in one highly polymorphic region containing SNPs and one insertion region. The read sequences of the conserved regions are aligned to the reference genome, while the reads belonging to the polymorphic and insertion region do not align to the reference genome. They are pooled together with erroneous reads and repetitive reads in the set of left-over reads.

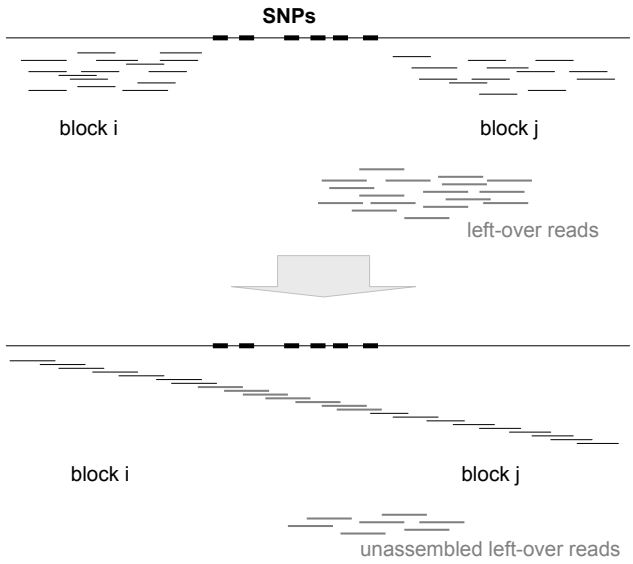

Figure 3.2.: Reads that are mapped to the reference genome are partitioned into blocks. Here, reads are pooled into block i and j. All non-alignable reads are pooled as left-over reads. The left-over read set contains erroneous and repetitive reads as well as reads that belong to divergent regions. The region between the blocks i and j is highly polymorphic and, thus, no read of the target sequence was mapped to the reference in the respective region. In the next step, the blocks i and j are reassembled by incorporating left-over reads that overlap with reads from the block. In an ideal scenario, the assembled sequence covers the regions of the blocks as well as the highly polymorphic region between them.

over reads are a mixture of erroneous reads, repetitive reads and those that are referring to highly diverged regions and it is not obvious which read belongs to which of these classes. Unfortunately, the set of left-over reads can have the size of 5% to 25% of all reads.

It is self-evident that the reassembly problem and the incorporation of left-over reads in particular gets harder with increasing number of reads. To keep the amount of reads to be reassembled within a reasonable limit, the mapped reads are partitioned into regions of overlapping alignments, called *blocks*. The reads are assigned to their respective block and each block is reassembled separately. In detail, the mapped reads are partitioned using a static region size or dynamically by using regions with zero coverage or repetitive regions as natural borders. Two consecutive blocks are assembled together by incorporating the left-over reads that overlap with the reads of these blocks. An illustration of a reassembly of two blocks is shown in Figure 3.2. Afterward, the produced contigs can be further scaffolded. In the process of scaffolding, neighbored contigs can be merged within one scaffold using mate-pair information.

The whole approach is implemented in the short read analysis framework SHORE [OSC+08, SOO+] that is specifically designed for resequencing projects with short read data. SHORE executes the following main steps:

1. **Filtering:** Trimming and quality filtering of reads.
2. **Alignment:** Reads are aligned to reference genome and left-over reads are pooled.
3. **Partitioning:** Aligned reads are partitioned into blocks of reads belonging to the same local region.
4. **Reassembly:** Assembly of the blocks to contigs by incorporating left-over reads.
5. **Scaffolding:** Merging and scaffolding of the contigs.

The pipeline includes several tools developed for short read data. In the alignment step, alignment tools like BWA [LD10], Bowtie [LTPS09] and GenomeMapper [SHO+09] are applied. GenomeMapper aligns reads simultaneously against multiple reference genomes after integrating these references in a single data structure. The other mapping tools, BWA and Bowtie, are discussed in Section 2.2. For the reassembly step, *de novo* assemblers like VELVET [ZB08], ABySS [SWJ+09] and EULER-SR [CP08] are supported as well as LOCAS and SUPERLOCAS, which are specifically designed to reassemble low sequencing depth data. In addition SUPERLOCAS is the only assembly tool that provides an efficient method for the incorporation of left-over reads. LOCAS and SUPERLOCAS will be presented in detail in Chapter 4 and Chapter 5. In the scaffolding step of SHORE, the scaffolding tool BAMBUS [PKS04] is utilized.

The presented methods have been mainly developed and implemented by Stephan Ossowski and Korbinian Schneeberger.

4. Short Read Assembly with a Low Sequencing Depth (LOCAS)

This chapter gives an overview of the workflow of the assembler LOCAS and implemented algorithms.

4.1. Overview

The assembly tool LOCAS is designed to assemble short to medium sized reads at a low sequencing depth. Therefore, the assembler extends the classical overlap-layout-consensus approach, which was originally developed to assemble Sanger reads. The overlap graph is reduced by a transformation into a *path graph* following the ideas of Myers et al. [Mye95, MSD+00, Mye05]. In the path graph, regions that can be unambiguously assembled are merged and represented as one vertex.

The main steps of the workflow of LOCAS are as follows:

1. **Preprocessing:** Detection of pairs of reads that have an identical k-mers
2. **Overlap phase:** Calculation of overlap alignments for detected pairs and construction of the overlap graph
3. **Layout phase I:** Transformation of the overlap graph into a path graph
4. **Layout phase II:** Cycle handling and extraction of final paths
5. **Consensus phase:** Determination of consensus sequences as contigs

To adjust the general overlap-layout-consensus approach to short-read data of low sequencing depth, we modified the traditional algorithm that selects the final paths in the path graph in the layout phase II: In comparison to Sanger reads, the graph size is increased due to the higher amount of reads. In addition, the short overlap length in comparison to Sanger reads leads to more overlaps, including also false overlaps. This increases the graph size additionally and leads to a higher complexity of the graph. We modified the algorithm such that it handles the increased number of false overlaps and the larger graph as well as the relative increase of branches in the graph. Paths are selected in the final graph using a greedy strategy that aims at maximizing the total sequence depth and the total quality of overlap alignments of the final paths.

4.2. Preprocessing

After all read sequences have been loaded, identical read sequences are eliminated. Identical read sequences are detected with an enhanced suffix array of the SeqAn library [DWRR08]. The frequency of each read sequence is counted and one copy is kept for each sequence. Optionally, a filter for repetitive reads can be applied that deletes all reads that contain a k-mer that occurs more often than defined by a threshold in the original read set. The size of the k-mer and the threshold for the minimum frequency can be set by the user. Next, pairs of reads are detected that have the same k-mers. The k-mers are searched with an enhanced suffix array that is built for all reads. The detected read pairs are reported as candidate pairs that overlap potentially. The identical k-mers between candidate pairs are the seeds for an overlap alignment following on phase two.

4.3. Overlap Phase

Overlap alignments are calculated for each candidate read pair with a minimal overlap length allowing a maximal number of mismatches. Both values are input parameters to the assembly tool. The number of maximal mismatches is either static or it depends of the actual length of the alignment.

Unfortunately, the information to which DNA strand a read belongs to is not provided by current sequencing machines since the produced reads are sequenced randomly from both strands of the DNA. Thus, the overlap alignment of two reads is calculated by aligning the sequences directly to each other and by aligning the reversed complement of one sequence to the other sequence, which is called an alignment with *similar orientation* or *dissimilar orientation*, respectively. Usually, the alignment with the higher alignment score, which indicates a lower number of mismatches, is preferred.

The calculated overlap alignment information is represented by an overlap graph. Each read is represented by a vertex and each alignment by an edge. Two types of edges exist: An *overlap* edge corresponds to an overlap alignment. A *containment* edge represents a global alignment and indicates that one read is aligned over its full length to the other read. If these two reads differ in their length, one read is contained in the other read. A weight that corresponds to the length of the alignment is assigned to each edge. Further, a score is assigned that is equal to the alignment score and describes the quality of the alignment.

For each overlap edge, the attributes *orientation* and *direction* are stored. The orientation is set to true if the corresponding reads align in similar orientation or to false if they align in dissimilar orientation, meaning that one sequence is aligned to the reversed complement of the other. The direction is set to right if the first read is elongated at its 5'-end by the second read. Otherwise, the direction is set to left. Thus, the overlap graph is implicitly a directed graph if we consider only overlap edges and leaving out the containment edges: the overlap edges of a

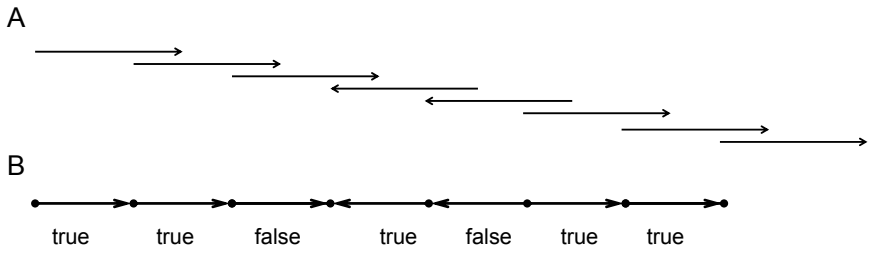

Figure 4.1.: (A) Eight reads are shown that align pairwise to each other. Except the fourth and the fifth read all reads are from the same DNA strand. Thus, the third reads aligns to the reversed complement sequence of the fourth read. The fourth and the fifth read align without building the reversed complement to each other. The reversed complement of the fifth read aligns with the sixth read. (B) For these reads and their alignments a seq-path is introduced in the overlap graph. The edge between the vertices of the third and the fourth read and the edge between the vertices of the fifth and the sixth read have a false orientation. For each of these edges the direction of the next edge is opposite to their direction.

vertex that are directed to the left correspond to ingoing edges and edges that are directed to the right to outgoing edges.

The consensus sequence of a path in the overlap graph is the consensus sequence of the aligned reads that are represented by the vertices along the path and the alignment information of the edges of the path. Since edges in the overlap graph are assigned with a direction and orientation, a path is not simply a list of edges that are connected with each other. For example, two edges that are directed to the right can only be part of a path if the orientation of the first edge is true and hence the underlying reads are aligned in the same orientation. The same holds for two left directed edges on a path. In contrast, if the orientation of the first edge is false, the corresponding reads align in dissimilar orientation and the direction of the second edge has to be opposite of the first edge. For an illustration, see Figure 4.1.

We define a path that considers the orientation and direction of edges a *seq-path*. It is defined as follows:

Definition 4. A *seq-path* is a sequence of consecutive overlap edges in the overlap graph such that for each pair of consecutive edges e_1 and e_2 holds that the edges e_1 and e_2 have the same direction if the orientation of edge e_1 is true and the opposite direction if the orientation of edge e_1 is false.

After the graph construction, the number of edges is decreased by reducing the redundancy in the graph. For vertices that are connected with a containment edge and, thus, represent the same sequence information, an *exemplar vertex* is chosen and the other vertices are assigned to it. For vertices assigned to the exemplar vertex, all edges are deleted except the containment edge to the exemplar vertex. Thus, the exemplar represents now additionally the read sequences of its contained

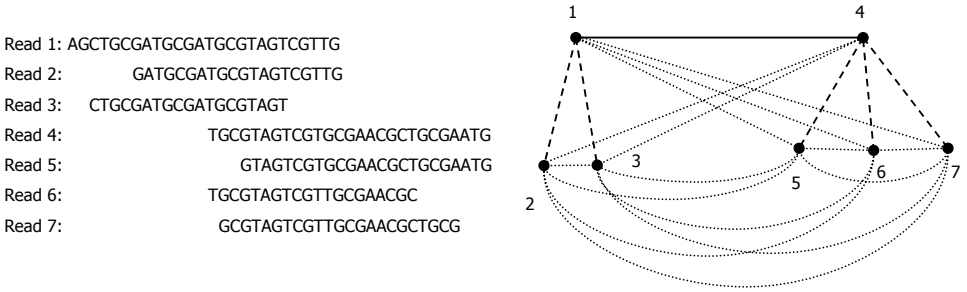

Figure 4.2.: Seven read sequences that are aligned to each other are shown. Most of the reads overlap with each other while read 2, read 3 and read 5, read 6, read 7 align globally with read 1 and read 4, respectively. The overlap graph of the reads is shown on the right. Each read is represented by a vertex and each overlap alignment by an overlap edge (solid). The global alignments are represented by containment edges (dashed). Read 1 and read 4 are exemplar vertices. For the other reads, all edges except the edge to the exemplar vertex are deleted (dotted).

reads.

The exemplar vertices are chosen using the following algorithm. A list L of all potential exemplars is built. The read sequence of such a potential exemplar vertex has to be longer than each read sequence of the vertices that are connected to it with a containment edge. A score is assigned to each potential exemplar vertex in L that is the average alignment score of its containment edges. L is sorted according to this score. The vertex x with the highest score is iteratively taken from L and assigned as exemplar vertex to each vertex v that has a containment edge to it and to that no exemplar vertex has been assigned until now. Further, the vertex x and all vertices that have been assigned to it are deleted from L.

Consequently, for vertex v an exemplar vertex x is chosen that has the highest average alignment score of its containment edges and its read sequence is at least as long as the read sequence represented by v. For all vertices that are assigned to an exemplar vertex, all edges are removed except the edge to the exemplar vertex. See Figure 4.2 for an illustration of the reduction.

4.4. Reduction and Path Graph Construction

In layout phase I, the overlap graph is further reduced by deleting transitive edges that are defined as follows:

Definition 5. An overlap edge $e_t = (v1, v2)$ with weight w_t is *transitive* if the following holds:

1. two edges $e_1 = (v_1, x)$ and $e_2 = (x, v_2)$ with weights w_1 and w_2, respectively, exist such that $w_1 \geq w_t$ and $w_2 \geq w_t$,

2. e_1 and e_2 are a seq-path,

3. the direction of e_t is equal to the direction of e_1 and

4. the orientation of e_t is true if the orientations of e_1 and e_2 are equal or the orientation of e_t is negative if the orientations of e_1 and e_2 are not equal

For a transitive edge e_t it holds that the edges e_1 and e_2 can exist in four different cases regarding their orientation and direction. These cases and the underlying read alignments are illustrated in Figure 4.3.

Algorithm 4.4.1: deleteTransitiveEdges()

```
for each vertex v
  do {sort edges adjacent to v in decreasing order of their weight
for each vertex v
  do { for each seq-path p = {e1, e2} adjacent to v
       do { w ← end vertex of p
            et ← edge between v and w
            if et exists and has the same direction as e1
              then { weight ← weight(et)
                     if weight ≥ weight(e1) or weight ≥ weight(e2)
                       then break
                       else if et is transitive with respect to e1
                                 and e2
                            then { et is marked as transitive
                                   transitive edge weights of e1 and e2
                                   are increased
for each edge e
  do { if e is marked
       then delete e
```

A transitive edge represents redundant alignment information. Thus, each transitive edge e_t can be deleted from the overlap graph. The information of each transitive edge e_t is saved in form of a *transitive edge weight* of edges e_1 and e_2. The transitive edges are found and reduced using Algorithm 4.4.1. For each vertex v, the adjacent edges are sorted in decreasing order of their weight. Then, seq-paths that consist of two edges $e_1 = (v, x)$ and $e_2 = (x, w)$ are enumerated. Note that the seq-paths are enumerated in decreasing order of the weight of their edges. For each seq-path $p = e_1, e_2$, it is checked whether an edge e_t with $e_t = (v, w)$ exists.

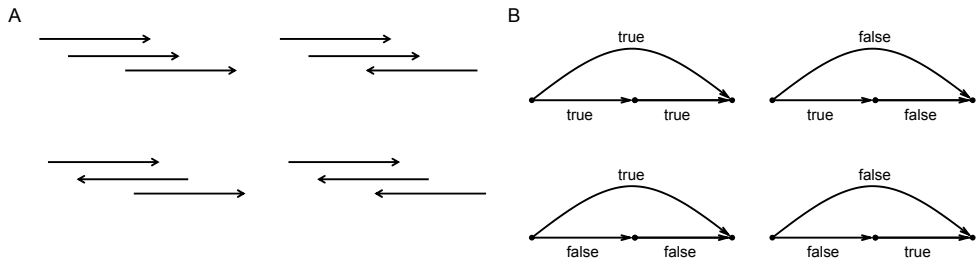

Figure 4.3.: If three read sequences show pairwise overlaps with each other, a transitive edge can occur in the overlap graph. (A) Four pairwise alignment settings of reads are shown that induce a transitive edge in the overlap graph. A read is drawn as an arrow pointing to right if the read sequence is aligned or as an arrow pointing to left if the reversed complement sequence of the read is aligned. For example, in the lower right alignment the first read is aligned to the reversed complement of the second read and the third read. The second and third read align to each other and thus also their reversed complements do. (B) For each alignment case, the corresponding sub-graph is shown. The three edges represent the pairwise alignments. The direction of all edges is right, which is indicated by an arrow, since all reads are elongated at their 5'-end. The orientation is given by a label. In the lower right alignment case, two edges with false orientation appear in the sub-graph. These edges represent the alignments of the first read to the second and third read. Their orientations are set to false since the first read is aligned with the reversed complements both reads. The remaining edge, with true orientation, represents the alignment of the second and the third read.

If edge e_t is transitive with respect to e_1 and e_2 following Definition 5, we mark e_t as transitive and increase the transitive edge weight of e_1 and e_2. This procedure is repeated until we find an edge e_t that has the same direction like e_1 but greater weight than e_1 or e_2. Then the next seq-path is considered for v. After all vertices are processed, marked edges are deleted.

The reduced overlap graph is transformed in a path graph. Therefore, all *unique* seq-paths of maximal length are determined in the overlap graph. A seq-path is unique if there exist no other seq-path that is adjacent to an inner vertex of the seq-path. For each unique seq-path of maximal length in the overlap graph, two vertices connected by an *inner* edge are introduced in the path graph. This structure composed of two vertices and an inner edge is called *seq-vertex*. The two vertices of the seq-vertex structure represent the ends of the unique seq-path in the overlap graph or, more precisely, the outer ends of the first and the last read sequence on the path. To both vertices of a seq-vertex structure a list of reads and a list of their pairwise alignments is assigned that originate from the seq-path in the overlap graph. To one vertex the original lists are assigned and to the other vertex the reversed lists are assigned. To reduce memory, only a link to the underlying read sequences and their alignments is stored for each seq-vertex. The consensus sequence of a seq-vertex is identical to the consensus sequence of the corresponding seq-path or to its reversed complement, depending on which

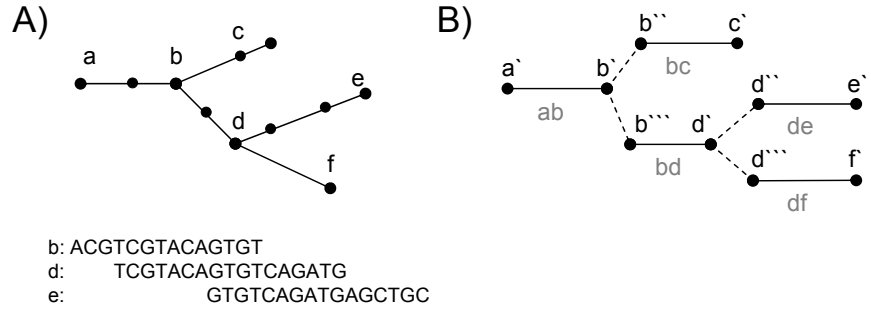

Figure 4.4.: An overlap alignment graph is transformed into a path graph. (A) The overlap graph and the aligned read sequences that are represented by the vertices b, d and e are shown. The overlap graph is transformed into a path graph, which is shown in (B). For each unique path in the overlap graph an inner edge is introduced in the path graph (drawn in solid). For example, the unique path between b and d in the overlap graph is represented by the inner edge (b''', d') in the path graph. If two vertices in the path graph represent two different ends of the same read, a real edge is introduced (drawn as dashed lines). An example is the real edge (d', d''). Vertices d' and d'' represent the same read since they both correspond to vertex d in the overlap graph. They represent different ends of the same read, since d' has an inner edge to b''' representing an overlap alignment at the 3'-end of the read and d' has an inner edge to e' representing an overlap at the 5'-end of the read.

direction the seq-vertex will be traversed. Note that every overlap edge in the overlap graph is already a unique seq-path. Consequently, every overlap edge will be represented by a seq-vertex in the path graph.

To connect seq-vertices in the path graph, for each adjacency between unique seq-paths in the overlap graph a *real* edge is inserted into the path graph. A real edge is introduced between two vertices in the path graph if these vertices represent different ends of the same read, see Figure 4.4.

To consider only valid paths for the consensus sequence, we have to make sure that a read sequence that has been traversed from one side is not traversed again from the other side. Therefore, we define *c-paths* as follows:

Definition 6. A *c-path* is a sequence of consecutive edges in the path graph that alternates strictly between inner edges and real edges starting with a real edge.

In other words, a c-path is a sequence of seq-vertex structures alternating with real edges. The consensus sequence of a c-path is determined using the reads and the alignment information of the seq-vertices along the c-path. The read list of a c-path is determined by appending the read lists of the seq-vertices while the last read of each seq-vertex is omitted. Only for the last seq-vertex of the c-path the whole read list is added. The corresponding list of overlap alignments is received

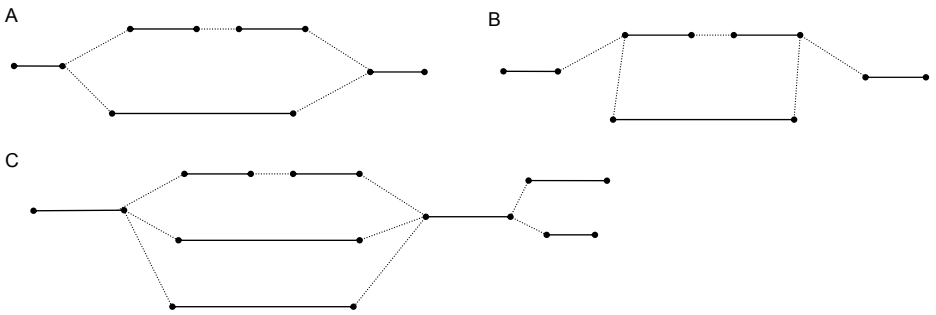

Figure 4.5.: In the path graph, different cycle types can occur. (A) An undirected cycle is shown. The two c-paths connect the same vertices. If the consensus sequences of both c-path are similar to each other, the structure arises from a sequencing error or SNP. If the consensus sequences of both c-paths are dissimilar a false alignment is in most of the cases the reason. (B) A directed cycle that usually arises from repeats or polymorphisms is shown. The c-path can be traversed an infinite number of times and thus should be cut. (C) A composed structure is shown. An undirected cycle offers two alternative paths through the graph. In addition, two directed cycles exist due to the alternative routes. A repeat region is represented by the c-path that does belong two both directed cycles but not to the undirected cycle. If the undirected cycle is similar then the repeat regions occurs with two copies in the target genome. Otherwise, the c-paths of the undirected cycle represent the two region that appear between three copies of the repeat in the target genome. Nevertheless, both directed cycles have to be cut since they can also originate from sequencing errors and lead to errors in the assembly.

by appending the list of overlaps of the seq-vertices with each other. This is feasible since two seq-vertices connected by a real edge represent the same read, which is the first and last read of the second and first seq-vertex, respectively. The length of a c-path is the length of its consensus sequence. Further, the length of a c-path regarding to two selected reads s and t is defined as the length of the consensus sequence of the c-path between the reads s and t.

4.5. Cutting Cycles and Similar Structures

In the next step, cycles in the path graph that arise from sequencing errors and repetitive regions in the target genome are handled. To this end, we distinguish between three different types of cycles. A *directed cycle* is a valid c-path that enables to traverse a vertex more than once. In fact, every vertex in the cycle can be traversed an infinite number of times. These cycles represent repeats and have to be cut since the consensus sequence of c-paths adjacent to directed cycles represent regions that originate from different regions in the genome. These regions should not be associated with each other in an assembly and, thus, connecting edges are deleted. An *undirected cycle* contains two distinct c-paths that

connect the same start and end vertices. This is called undirected, because both c-paths together create no c-path and hence cannot be traversed infinite times. Instead, it represents alternative routes through the path graph. The c-paths of an undirected cycle have either a similar or a dissimilar consensus sequence. The first kind of cycle is called *similar undirected cycle* and arises from sequencing errors or polymorphisms. The second kind of cycle is called *dissimilar undirected cycle* and arises from repeats or false alignments. For both cycle types, one of the c-paths has to be cut since the consensus sequence can contain only one of the alternatives. In Figure 4.5 all three cycles types are shown.

Algorithm 4.5.1: CALCULATESPANNINGTREE()

```
PriorityQueue ← create Priority Queue of all real edges based on edge score
UnionFind ← create Union Find structure by defining each seq-vertex as set
SpaTree ← empty List of real edges
DisEdges ← empty List of real edges
while PriorityQueue not empty
do { candidateEdge ← PriorityQueue.pop()
     verL ← seq-vertex v of candidateEdge = (v, w)
     verR ← seq-vertex w of candidateEdge = (v, w)
     edgeToInsert ← true
     if UnionFind.find(verR) = UnionFind.find(verL)
       then { undirCycle ← GETUNDIRECTCYCLE(verR, verL, SpaTree)
              if undirCycle exists
                then { edgeToInsert ← false
                       if dissimilar consensus sequence of undirCycle
                         then DisEdges.add(candidateEdge) }
              if CUTDIRECTCYC(candidateEdge, SpaTree, DisEdges)
                then edgeToInsert ← false }
     if edgeToInsert
       then { SpaTree.add(candidateEdge)
              UnionFind.union(verR, verL) }
return (SpaTree)
```

Cycles are handled by creating a spanning tree in the path graph such that no directed and no undirected cycle is part of the solution. Consequently, c-paths belonging to distant regions in the target genome are disconnected. To determine the spanning tree, Algorithm 4.5.1 is applied to the path graph by selecting iteratively real edges. The algorithm is a variation of Kruskal's algorithm [KJ56] which finds a minimum spanning tree for an undirected weighted graph. First, real edges are scored by the quality of overlap alignments and sequencing depth assigned to the two seq-vertices connected by the real edge. Then, all real edges are enumerated in a decreasing order of their score such that regions of high sequencing depth

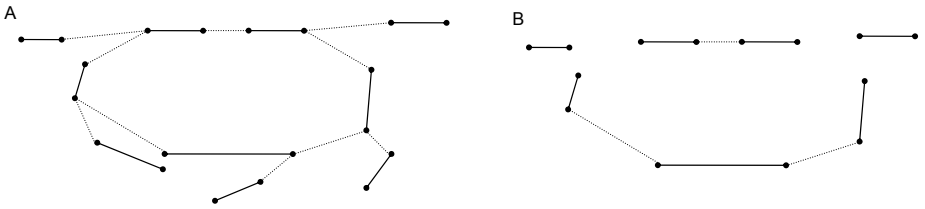

Figure 4.6.: A directed cycle is cut in a path graph. (A) A directed cycle with short dangling paths is displayed. (B) The cycle is shown after all short paths have been cut off and the cycle has been cut open at each branch vertex.

and of high quality alignments are preferred for the assembly. An edge is only selected if neither an undirected cycle nor a directed cycle is introduced in the current solution, which is checked by Algorithm 4.5.2 and 4.5.3. Algorithm 4.5.2 determines whether a given edge introduces an undirected cycle in the current spanning tree and returns eventually the cycle. If the returned cycle is dissimilar, the edge is marked. Algorithm 4.5.3 checks whether an edge introduces a directed cycle in the current spanning tree. Finally, the spanning tree, which is induced by the selected edges in the path graph, is reported by Algorithm 4.5.1.

In the following, we will discuss Algorithm 4.5.2 and 4.5.3 in more detail. Both algorithms are called with a spanning tree and an edge. Algorithm 4.5.2 determines the existence of an undirected cycle in the spanning tree that is introduced with the given edge. All seq-vertices with degree of at least three in the neighborhood of the given edge are determined where the neighborhood is restricted by a heuristic threshold of 1000 bp. For each pair of the found seq-vertices, c-paths are determined that connect both seq-vertices. If there exist two disjoint c-paths that connect both seq-vertices such that only one of them traverses the given edge, the two c-paths constitute an undirected cycle. The detected cycle is reported.

Algorithm 4.5.3 reports if a directed cycle is introduced in the spanning tree with the given edge. For this purpose, marked edges will also be considered as part of the spanning tree. This is done to detect more existing directed cycles in the path graph, even those that are part of an undirected cycle that has been cut already. As a consequence, read sequences from repetitive regions will be isolated. The c-path of each directed cycle is cleaned of all short dangling paths, which can be adjacent to vertices with degree larger than two, called *branch vertices*, see Figure 4.6. Then, the c-path is cut at all *repeat branches*. Repeat branches are branch vertices of the directed cycle that are adjacent to at least one longer c-path that is not part of the cycle. The directed cycle and the repeat branches are recorded for each cycle. The consensus sequence of a directed cycle represents the repetitive sequence and the sequence between the copies of the repeat in the target genome. In an ideal scenario, there exist two repeat branches on the c-path of the cycle that indicate the start and end of the repetitive region. C-paths that are adjacent to these repeat branches but are not part of the directed cycle represent

other regions that occur before, between or after the copies of the repeat in the target genome. Repeat branches can be easily detected under the assumption of error free reads and assuming that the copies of a repeat region in the target genome are identical. In practice, the detection is complicated by branch vertices that lie on the directed cycle and are adjacent to paths that do not belong to the cycle. However, these paths are often short such that the branch vertices can be distinguished from repeat branches. Unfortunately, directed cycles can arise also due to sequencing errors. Since we cannot determine the correct order of the paths that are adjacent or on the directed cycle, we simply cut the cycle and do not relink the adjacent paths. Such a remodeling becomes possible if additional information is available, for example mate-pair information.

Algorithm 4.5.2: GETUNDIRECTCYCLE($verL, verR, SpaTree$)

$branchVerticesR \leftarrow$ seq-vertices with degree ≥ 3 near $verR$ in $SpaTree$
$branchVerticesL \leftarrow$ seq-vertices with degree ≥ 3 near $verL$ in $SpaTree$
for each pair (x, y) with $x \in branchVerticesR$ and $y \in branchVerticesL$
do
- $path1 \leftarrow$ c-path between x and y on $SpaTree$
- **if** $path1$ exists
  - **then**
    - $path2 \leftarrow$ c-path from x to y on $SpaTree$ via $edge = (verR, verL)$
    - **if** $path2$ exists
      - **then**
        - $undirectedCycle \leftarrow$ merge $path1$ and $path2$
        - **return** ($undirectedCycle$)

return (**false**)

Algorithm 4.5.3: CUTDIRECTCYC($verL, verR, SpaTree, DisEdges$)

$directedCycle \leftarrow$ c-path from verL to VerR on $SpaTree$ traversing at most one edge of DisEdges
if $directedCycle$ exists
then
- $branches \leftarrow$ all seq-vertices on directedCycle with degree ≥ 3
- **for each** $b \in branches$
  - **do**
    - $longestPath \leftarrow$ longest c-path from b to a leaf in $SpaTree$ without edges of $directedCycle$
    - **if** $length(longestPath) < tresholdMaxLength$
      - **then** delete all real edges of the subtree of b without considering edges in $directedCycle$
      - **else** delete all real edges adjacent to repeat branch b
- **return** (**true**)

else return (**false**)

By assuming seq-vertices as vertices and real edges as edges Algorithm 4.5.1 returns an acyclic graph, the spanning tree on the path graph.

After Algorithm 4.5.1 has been applied, we search for *fragmented directed cycles*. A fragmented directed cycle is a c-path that has the same start and end vertex but one edge of the c-path is missing. Such a c-path occurs if an overlap between reads has not been detected and the related real edge is missing in the spanning tree. As for directed cycles that are not fragmented, the consensus sequences of adjacent paths represent regions of distant locations in the target genome that should not be assembled to each other. Thus, fragmented directed cycles have to be cut at the corresponding repeat branches.

The algorithm to detect and cut repeat branches of fragmented directed cycles works as follows: The spanning tree is searched for repeat branches by searching for vertices that are connected to at least two longer c-paths. Such paths need to have a consensus sequence with a length of at least 125 bp. We found this threshold to be a good trade off between false positive and true positive identified repetitive regions. Edges adjacent to these vertices are deleted.

4.6. Resolving Repeats Using Mate-Pair Data

With the additional information provided by mate-pairs, some repeat induced cycles can be resolved in the path graph following the strategy of Pevzner and Tang [PT01]. At present, the approach is restricted to repetitive regions that occur only with two copies in the target genome.

The repeat sequence and the sequence between the two copies of the repeat in the target genome are represented by a directed cycle in the path graph. In particular, the repeat sequence is represented by a sub-path of the directed cycle, which is called *repeat-path*. A repeat-path is either a single seq-vertex consisting of repeat branches or a path of two repeat branches connected by a c-path.

The sequence regions before and after the repeat copies are represented by paths that are adjacent to the repeat-path but not part of the directed cycle. They are called *framing-paths*. An example of a repetitive region and the corresponding directed cycle is illustrated in Figure 4.7 A.

A directed cycle can be re-modeled such that it represents the original order of the sequence in the target genome. Therefore, the repeat-path has to be duplicated and re-linked, which is also illustrated in Figure 4.7. However, these new adjacencies have to be confirmed by mate-pairs in the directed cycle and the framing-paths that restrict the order of the paths with their insert size.

Without mate pair information, directed cycles are detected, short dangling paths are deleted and the cycles are cut using Algorithm 4.5.3. If mate-pair information is available, Algorithm 4.6.1 is called instead. Beforehand, the detected directed cycle is cut from all short dangling paths.

With Algorithm 4.6.1 all possible repeat-paths of the directed cycle are determined. For each of these repeat-paths it is investigated whether the adjacent

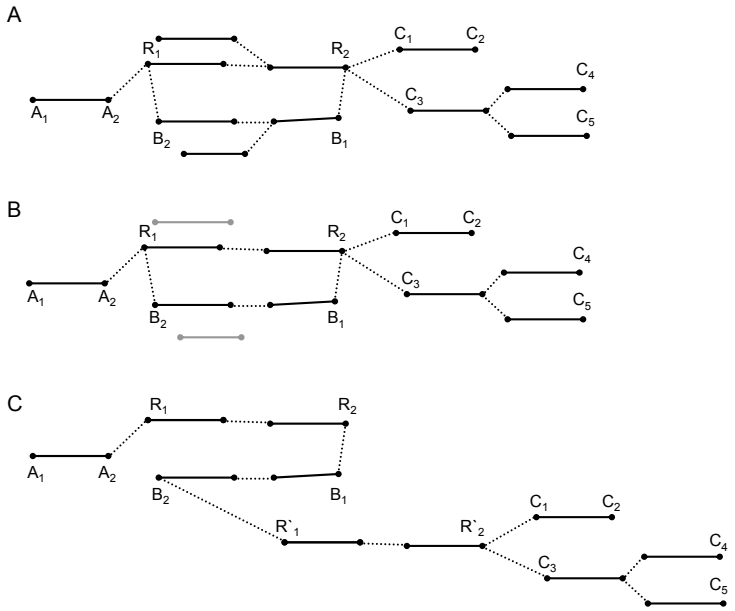

Figure 4.7.: (A) A directed cycle in a path graph is shown. The repeat region is represented by the repeat-path between the repeat branches R_1 and R_2. The sequence between the two copies of the repeat in the target genome is represented by the path between B_1 and B_2. The path from A_1 to A_2, which represents the sequence before the repeat region in the genome, and the paths between C_1 and C_2, C_3 and C_4, and C_3 and C_5, which represent the sequence behind the copies of the repeat region in the genome, are called framing-paths. (B) Short dangling paths in the directed cycle are cut off. (C) The directed cycle is remodeled. The repeat-path is copied and re-inserted as path between $R`_1$ and $R`_2$. All framing-paths of R_2 are disconnected from the vertex and linked to $R`_2$. The vertex $B_2$ is disconnected from $R_1$ and linked to $R`_1$.

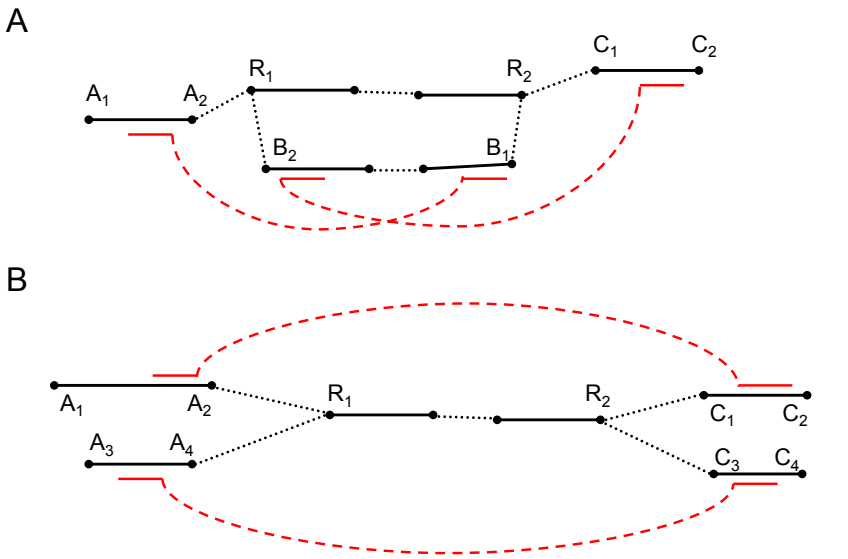

Figure 4.8.: (A) For a path graph, a directed cycle with two assigned mate-pairs (red) is shown. The repeat-path of the cycle is between the repeat branches R_1 and R_2. The path between B_1 and B_2 represents the sequence between the copies of the repeat in the target genome. The framing-paths are the paths between A_1 and A_2 and between C_1 and C_2. The reads of the left mate-pair are assigned to the path between A_1 and A_2 and the seq-vertex of B_1, respectively. The insert size of the left mate-pair has approximately the same length as the path between them connecting consecutively A_2, R_1, R_2 and B_1. Thus, the mate-pair confirms this connection. The path that connects the right mate-pair by traversing consecutively the vertices B_2, R_1, R_2, C_1 and C_2 is confirmed if the insert size has approximately the same length as this path. (B) A fragmented directed cycle is shown. The repeat-path is between the branch vertices R_1 and R_2. The paths between A_1 and A_2, A_3 and A_4, C_1 and C_2 and between C_3 and C_4 are adjacent to the repeat-path. In addition, two mate-pairs (red) are assigned to these adjacent paths. The upper mate-pair assigns the path between A_1 and A_2 to the path between C_1 and C_2. The insert size of the mate-pair has approximately the same length as the path that connects them by traversing consecutively A_2, R_1, R_2, C_1 and C_2. This connection is confirmed by the mate-pair. The lower mate-pair confirms its connecting path that traversed consecutively A_4, R_1, R_2, C_3 and C_4.

paths of the repeat-path can be assigned to each other. Therefore, we search for reads in the framing-paths that mate with reads in the respective beginning of the directed cycle not considering the repeat-path. This is illustrated in Figure 4.8 A. If the insert size of the mate-pair matches about the length of the repeat-path plus the distance to the mate-pair reads, one copy of the repeat-path can be assigned to the respective adjacent paths. If there are enough mate-pairs that confirm the assignment, the directed cycle is remodeled. Therefore, Algorithm 4.6.2 duplicates the repeat-path and re-links it according to the assignment.

Algorithm 4.6.1: RESOLVEDIRECTEDCYCLE(*directedCycle*)

---

branches ← all vertices on directedCycle with degree > 2
for each pair v_1 and $v_2 \in$ *branches* that are the ends of a repeat-path
do {
  repeatPath ← repeat-path between v_1 and v_2
  *paths*1 ← c-paths starting in v_1 with each edge $\notin$ *directedCycle*
  *paths*2 ← c-paths starting in v_2 with each edge $\notin$ *directedCycle*
  *cycPath*1 ← short c-paths starting in v_1 with each edge $\in$ *directedCycle*
  *cycPath*2 ← short c-paths starting in v_2 with each edge $\in$ *directedCycle*
  $MateCount1c2 \leftarrow 0$
  $MateCount2c1 \leftarrow 0$
  for each mate-pair (m_1, m_2) with m_1 assigned to *paths*1 and m_2 assigned to *cycPath*2
  do {
    pathBetween ← c-path connecting seq-vertex of m_1 with *repeatPath* and *cycPath*2
    lengthB ← consensus sequence of *pathBetween* from m_1 to m_2
    if $lengthB \approx m_1.length() + m_2.length() + (m_1, m_2).size()$ **then** $MateCount1c2 \leftarrow MateCount1c2 + 1$
  }
  for each mate-pair (m_1, m_2) with m_1 assigned to *cycPath*1 and m_2 assigned to *paths*2
  do {
    pathBetween ← c-path connecting seq-vertex of m_2 with *repeatPath* and *cycPath*1
    lengthB ← consensus sequence of *pathBetween*from m_1 to m_2
    if $lengthB \approx m_1.length() + m_2.length() + (m_1, m_2).size()$ **then** $MateCount2c1 \leftarrow MateCount2c1 + 1$
  }
  if $MateCount2c1 \geq minNum$ **and** $MateCount1c2 \geq minNum$
  then {
    REMODELDIRECTEDCYCLE(directedCycle, repeatPath)
    return (**true**)
  }
}
return (**false**)

Algorithm 4.6.2: REMODELDIRECTEDCYCLE($directedCycle, repeatPath$)

$vertex1 \leftarrow$ start vertex of $repeatPath$
$vertex2 \leftarrow$ end vertex of $repeatPath$
$duplicatedRepeatPath \leftarrow repeatPath.copy()$
$insert(duplicatedRepeatPath)$
$vertexDu1 \leftarrow$ start vertex of $duplicatedRepeatPath$
$vertexDu2 \leftarrow$ end vertex of $duplicatedRepeatPath$
for each edge $e = (vertex2, otherVertex2)$ with $e \notin directedCycle$
do $\begin{cases} deleteEdge(e) \\ insertEdge(vertexDu2, otherVertex2) \end{cases}$
$dirEdge \leftarrow$ edge $dirEdge = (vertex1, otherVertex1)$
with $dirEdge \in directedCycle$
$deleteEdge(dirEdge)$
$insertEdge(vertexDu1, otherVertex1)$

After Algorithm 4.5.1 has been applied, fragmented directed cycles are resolved by using mate-pair information. The procedure is similar to Algorithm 4.5.3. It is investigated whether adjacent paths of one end of the repeat-path can be assigned to adjacent paths of the other end of the repeat-path. If two reads of such paths, are mated in a way that their insert size matches approximately the distance between them when incorporating the repeat-path, they are assigned to each other, see Figure 4.8 B. If all adjacent paths can be assigned unambiguously and confirmed by a reasonable number of mate-pairs, the fragmented cycle is remodeled. Therefore, the paths that are assigned to each other are connected with one copy of the repeat-path. Thus, an alternative version of Algorithm 4.6.2 can be applied.

Finally, all fragmented directed cycles that could not be resolved by using mate-pair information are detected and cut by scanning for potential repeat-paths that do not lie on directed cycles as introduced in Section 4.5.

4.7. Contig Extraction

Since directed cycles and undirected cycles in the path graph were handled as described in Section 4.5 and Section 4.6 the resulting graph is a disjoint set of spanning trees, a spanning forest.

For each spanning tree in the forest, the longest path is determined to maximize the length of the consensus sequence. The longest paths in a spanning tree can be determined in polynomial time with the algorithm of Bulterman et al. [BvdSZ+02]. First, the longest c-path starting at each leaf in each tree are determined and the end vertices are reported. In a second step, the longest c-paths starting at each end vertex are determined. These final c-paths are longest paths in the forest.

The consensus sequence is determined for each final path. Therefore, a multiple alignment is calculated for the read sequences assigned to each final path. The reads are ordered to a multiple alignment by considering their pairwise overlap alignments. The consensus sequence of the multiple alignment is determined by choosing the nucleotide with the highest relative frequency at each position in the alignment.

4.8. Software Architecture

The assembly tool LOCAS is written in C++ with use of the SeqAn library [DWRR08]. Figure 4.9 shows an UML diagram that presents the class hierarchy of LOCAS. The software operates in a serial manner. The class `Assembler` instantiates and calls consecutively the classes `ReadCollection`, `Aligner`, `Reducer`, `Pather` and `ConsensusBilder`. These classes represent different steps in the workflow of the software. Each class fulfills its function via the main function `apply()`, which calls private functions of the class. The class `ReadCollection` loads the read sequences, the class `Aligner` handles the construction of the overlap graph, the class `Reducer` transforms the overlap graph in a path graph, the class `Pather` determines the final paths in the path graph, and the class `ConsensusBilder` determines the contigs. Except for `ReadCollection`, the workflow classes instantiate and/or use an object of either the class `AlignmentGraph`, `PathGraph` or `ContigCollection`. All three classes do not represent steps in the workflow but rather blue prints of real data containing objects. The `AlignmentGraph` represents an overlap graph, the `PathGraph` represents a path graph and the `ContigCollection` a set of contigs. For each of the listed classes, only one instance exists during the workflow of the tool.

The `ReadCollection` holds several instances of the classes `Mate` and `SingleRead`. The class `SingleRead` is a blue print of a read. The attributes describe a read sequence with its nucleotide sequence, its id, its number of copies in the data set, its fasta entry and which reads are contained in it. The class `Mate` describes a mate-pair with the ids of the mate reads and the ids of the seq-vertices to that the mate reads are assigned later in the workflow.

The standard workflow of the tool starts with the main function of the `ReadCollection`. The read sequences and the mate-pair information are loaded from the input files. Each read is stored as a `SingleRead` object and for each mate-pair a `Mate` object is instantiated. Then, the set of `SingleReads` is reduced by calling private functions of `ReadCollection` which delete identical reads and assign contained reads to an exemplar read. After this step, the `ReadCollection` object is utilized in the workflow as a data bank of the reads and the mate-pairs.

In the overlap phase, the `Assembler` object instantiates an `Aligner` object. The main function of Aligner handles the construction of the overlap alignment graph that is represented by an `AlignmentGraph` object. The `SingleRead` objects of `ReadCollection` represent the vertices of the graph by their id. For each

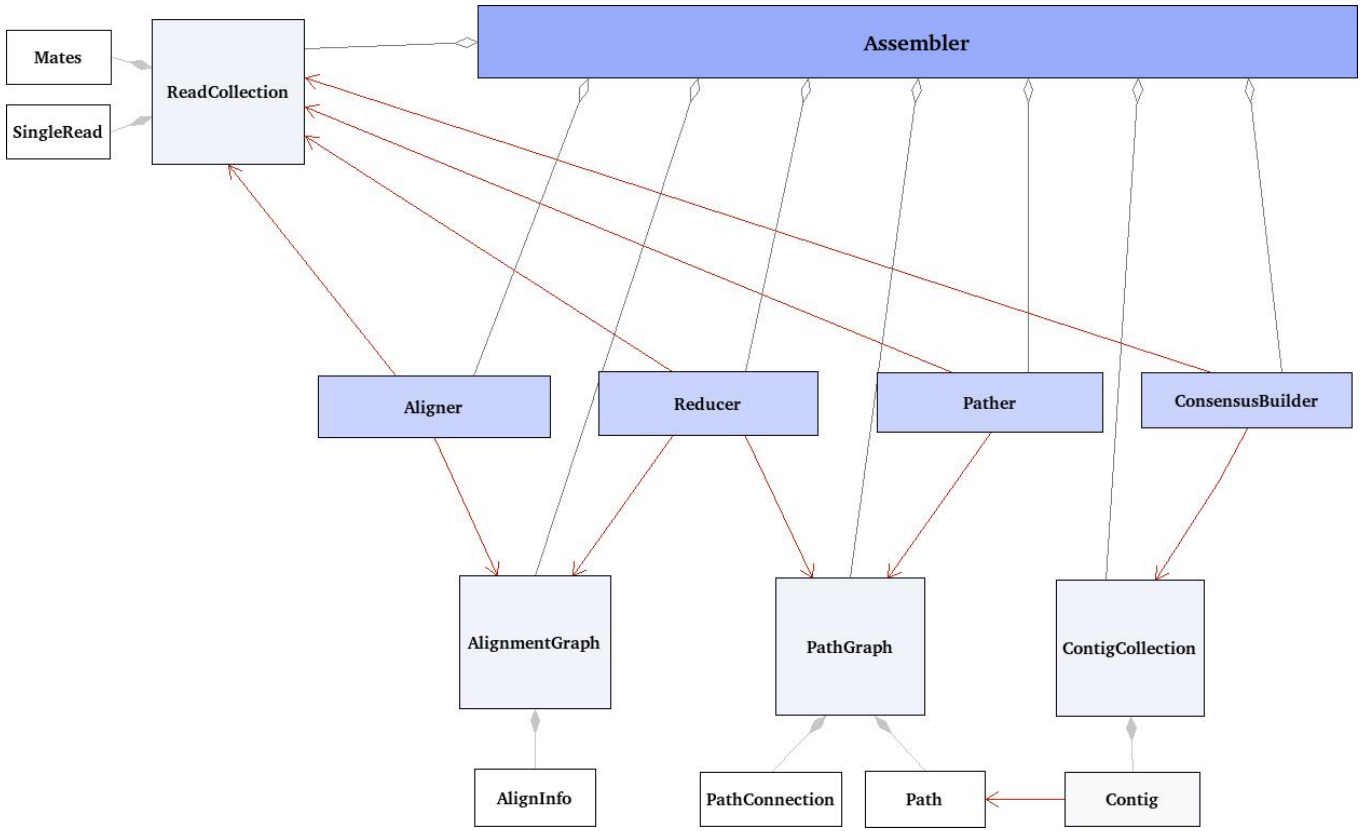

Figure 4.9.: The UML diagram shows the class hierarchy of the assembly tool LOCAS. The class `Assembler` manages the workflow by calling consecutively the classes `ReadCollection`, `Aligner`, `Reducer`, `Pather` and `ConsensusBilder`. These classes handle different steps in the workflow and are allowed to modify the class `ReadCollection`. This class has the classes `Mate` and `SingleRead`, which represent blue prints of mate-pairs and reads, respectively. The classes `AlignmentGraph`, `PathGraph` and `ContigCollection` are instantiated by the class `Assembler`. The class `AlignmentGraph`, which represents the overlap graph, is modified by the class `Aligner` and `Reducer`. The class `PathGraph`, which is a blue print for a path graph, is modified by the class `Reducer` and `Pather`. The class `ContigCollection` is modified by the `ConsensusBilder` and represents the set of final contigs.

overlap, an `AlignInfo` object is instantiated in the `AlignmentGraph` object. The `AlignInfo` class is a blue print for an overlap alignment between reads and, consequently, represents an edge of the graph. The attributes describe features of the overlap alignment like its length, its score, its number of mismatches as well as features of the edge like the direction, orientation and the ids of the connected vertices.

In layout phase I, an object of the class `Reducer` is instantiated in the `Assembler` object. The `Reducer` object has the right to modify an `AlignmentGraph` object to reduce the represented overlap graph. The actual reduction is performed by the main function of `Reducer`. Then, the reduced graph is transformed in a path graph. To this end, a `PathGraph` object is instantiated containing several `Path` objects that represent the vertices in the seq-vertices of the path graph and several `PathConnection` objects that represent the real edges. Each seq-vertex is represented by two `Path` objects with a consecutive id. The inner edge connecting them is not stored and exists implicitly.

The main workload of the tool is handled by the `Pather` object, which is instantiated by the `Assembler` object, in layout phase II. The `Pather` object has the right to modify the `PathGraph` object. Its main function calculates the spanning tree of the path graph in the `PathGraph` object. Further, the longest paths are calculated and stored as `Path` objects.

The `Pather` object hands over the `Path` objects containing the longest paths, to an instantiated `ConsensusBuilder` object. The `ConsensusBuilder` instantiates the `ContigCollection` object, which is only a container for `Contig` objects. For each `Path` object, a `Contig` object, which represents a contig, is instantiated. Each `Contig` object is responsible for extracting the contig of the assigned `Path` object. Finally, it contains the consensus sequence and the position-wise sequencing depth. The final set of contigs, which presents the final assembly solution, is collected and reported in an output file by the `ContigCollection`.

5. Extension of Homology-Guided Assembly (SUPERLOCAS)

In this chapter, we introduce algorithms for reassembly that are implemented in the software tool SUPERLOCAS.

In our approach reads are assembled by utilizing a mapping onto a reference genome. However, when using a mapping approach, regions of high divergence as well as long insertions can often not be mapped to the reference, leaving a set of left-over reads. These regions are reconstructed with SUPERLOCAS by incorporating high quality left-over reads in the assembly.

We developed SUPERLOCAS as an extension of the regular assembly algorithm implemented in LOCAS. SUPERLOCAS is adapted to the homology-guided assembly approach of the SHORE pipeline [OSC+08, SOO+], which we introduced in Chapter 3. With SHORE, reads are mapped against a reference genome. The mapped reads are partitioned into blocks of given length and all left-over reads are pooled. Similar to LOCAS, SUPERLOCAS handles the reassembly step. It assembles each block individually while incorporating relevant left-over reads. In addition, the tool takes advantage of given mapping positions of reads.

5.1. Incorporating Left-Over Reads

In this section, we discuss the overall workflow of the assembly tool SUPERLOCAS.

The algorithm of SUPERLOCAS is an extension of the basic assembly algorithm that is implemented in LOCAS. In particular, SUPERLOCAS applies extended algorithms of LOCAS to calculate a pre-assembly of the left-over reads and to assemble mapped reads while incorporating a part of the pre-assembled left-over reads. The workflow is shown in Figure 5.1. First, an overlap graph based on the left-over reads is created, which is called *left-over graph*. The left-over graph is re-used in later steps of the workflow. Then, each block is assembled separately following the basic assembly algorithm with one additional operation: all connected components of the left-over graph whose reads overlap with reads of the current block are recruited and linked to the overlap graph of the block. The resulting overlap graph is processed with the standard algorithms to extract the longest paths and to determine the final contigs.

In the following, we will give more details of the workflow of SUPERLOCAS in Algorithm 5.1.1. First, overlap alignments are calculated for all left-over reads

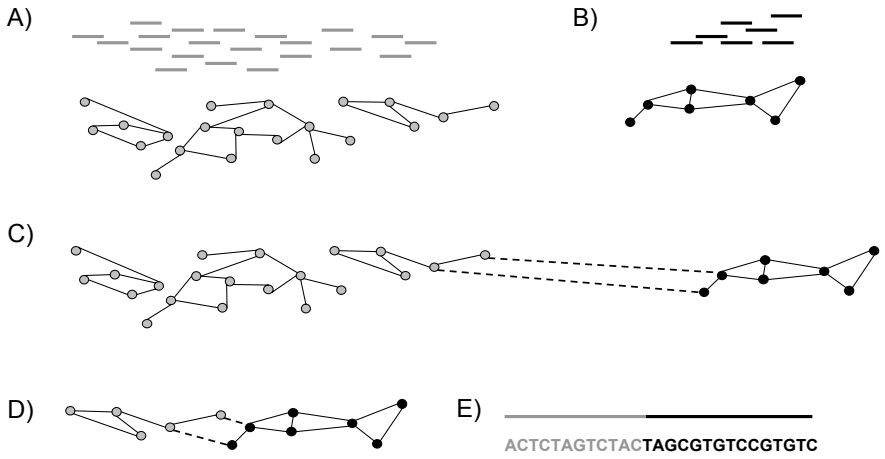

Figure 5.1.: Workflow of SUPERLOCAS. (A) For the left-over reads, an overlap graph, called left-over graph, is constructed. (B) For the reads that are assigned to one block, another overlap graph is constructed. (C) Overlaps are detected between the reads of the block and the left-over reads. For each detected overlap, an edge is inserted between the graphs. (D) For each inserted edge, the adjacent connected components of the left-over graph are copied into the overlap graph of the block. (E) The extended overlap graph is processed with the standard algorithms of LOCAS. Final paths are extracted and the respective contigs are reported. The process from (B) to (E) is repeated for each block.

and represented in a left-over graph. Then, the following procedure is applied for each block: The overlap alignments between the reads of the block are calculated and the overlap graph is created. Next, overlap alignments between left-over reads and block reads are calculated. For each of these overlaps alignments, a new edge and a new vertex are inserted in the overlap graph representing the alignment and the left-over read, respectively. The new vertex is a copy of the vertex in the left-over graph and represents the same read.

In addition, all other vertices of the respective connected component in the left-over graph are copied into the overlap graph including all edges. This is done by traversing the connected components via a breadth-first search starting with the vertex that has been copied in the previous step. Note that each edge and each vertex of the left-over graph is copied at most once to the overlap graph. With this procedure the overlap graph of each block is extended such that overlapping left-over reads are also represented in the graph. Hence, some of the left-over reads are taken into account and can substantially elongate the resulting contigs.

Algorithm 5.1.1: ASSEMBLEWITHLEFTOVERREADS($leftis, blocks$)

$overlapsLeftis \leftarrow$ calculate overlaps between $leftis$
$graphLeftis \leftarrow$ construct overlap graph for $overlapsLeftis$
while $blocks$ **not** empty
do
- $blockReads \leftarrow blocks.getReads()$
- $overlapsBlock \leftarrow$ calculate overlaps between $blockReads$
- $graphBlock \leftarrow$ construct overlap graph for $overlapsBlock$
- $newOverlaps \leftarrow$ calculate overlaps between $blockReads$ and $leftis$
- $newEdges \leftarrow$ construct edges for $newOverlaps$
- $newVertices \leftarrow$ vertices of $newEdges$ in $graphLeftis$
- **for each** v in newVertices
  - **do** mark all edges and vertices visited during a breadth-first-traversal starting at v in $graphLeftis$
- $graphBlock \leftarrow$ copy $newVertices$, $newEdges$ and all marked vertices and edges into $graphBlock$)
- **return** ($graphBlock$)

5.2. Making Use of Mapping Positions of Reads

In this section, we discuss extensions of the assembly workflow of LOCAS to make use of provided mapping positions of reads. These extensions are implemented in SUPERLOCAS. The read positions are used in the pre-processing as well as in the overlap phase of the workflow.

In the regular pre-processing of LOCAS, pairs of reads are determined that have an identical k-mer in their sequence. These pairs of read sequences, called

candidate pairs, overlap potentially with each other. Their common k-mer is used as seed for their alignment. This pre-processing step is extended SUPERLOCAS. In addition, to identical k-mers between read sequences, given mapping positions are analyzed to detect possible candidate pairs. Each pair of reads with a small distance to each other regarding the mapping positions is defined as candidate pair.

In the overlap phase, overlap alignments are calculated for each candidate pair. In the regular algorithm of LOCAS, the best alignment is calculated using the same alignment constraints like minimal overlap length and maximal number of mismatches for each candidate pair. However in case of SUPERLOCAS, five types of candidate pairs are distinguished:

1. both reads are block reads and have a small distance to each other
2. both reads are block reads, have a regular distance to each other and have an identical k-mer
3. both reads are block reads and have an identical k-mer while showing very distant alignment positions
4. both reads are left-over reads
5. one read is a left-over read and the other one is a block read

For each type, different alignment constraints are used and can be changed optionally by the user. Usually, more mismatches and a smaller overlap length are allowed for reads with a small distance to each other such that more overlap alignments can be detected in regions of low sequencing depth. To avoid false overlap alignments between distant reads, strict overlap constraints are set for these read pairs. With a reduced number of false overlaps, also the number of dissimilar undirected cycles decreases in the path graph. Overlap constraints for candidate pairs of type four and five are also very strict to avoid false overlaps of left-over reads with each other or with block reads.

5.3. Software Architecture

In this section, we give details about the architecture of the software SUPERLOCAS. The tool is written in C++ with use of the SeqAn library. Most classes are shared with LOCAS, while the classes `Merger` and `AlignFinder` are unique to SUPERLOCAS.

Figure 5.2 shows the classes of SUPERLOCAS in an UML diagram. The class `Merger` handles the workflow of the software by calling the class `Assembler` for the assembly step and calling the class `AlignFinder` to handle the incorporation of left-over reads. The class `Assembler` loads the read sequences and

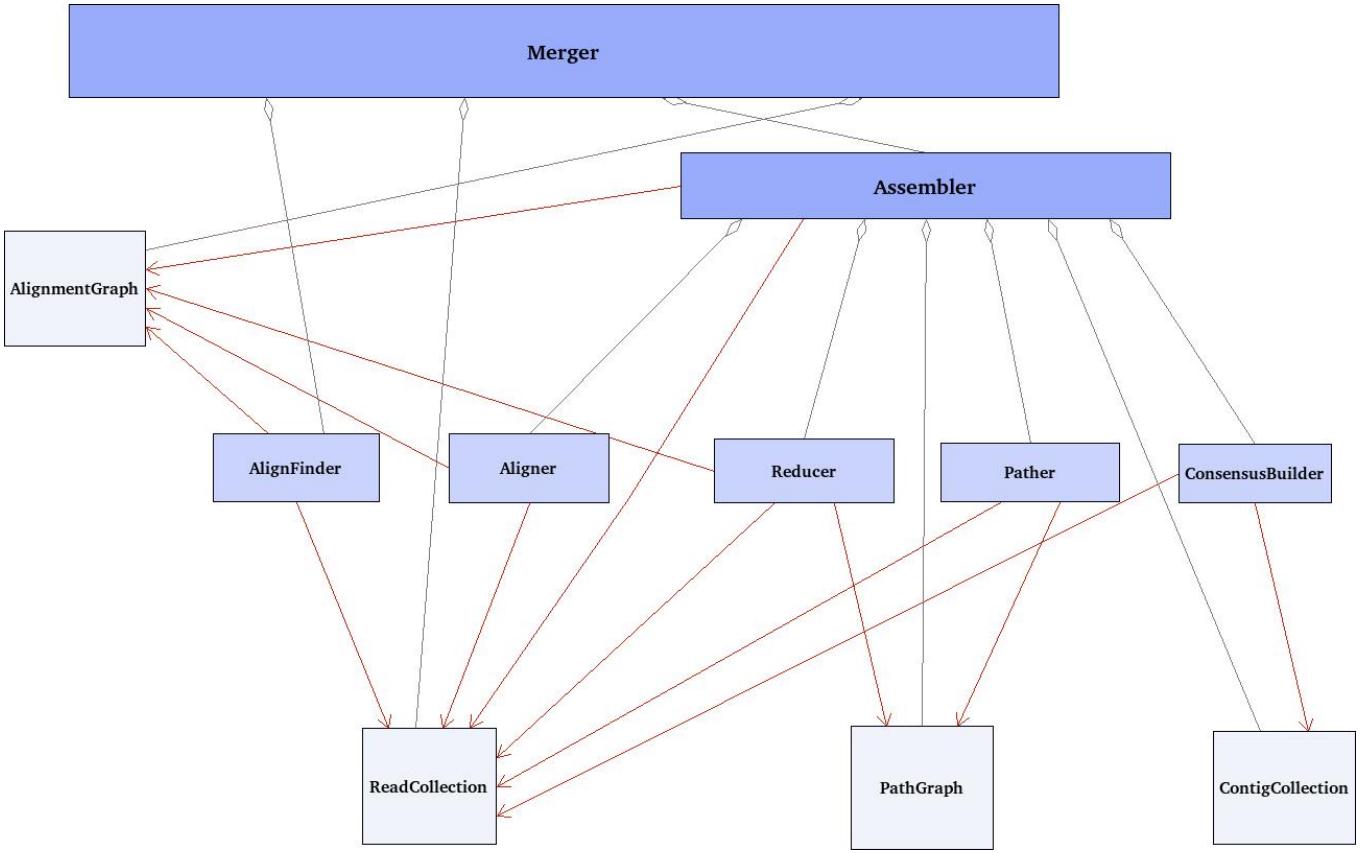

Figure 5.2.: The class hierarchy of SUPERLOCAS is shown. The class `Merger` handles the workflow by calling the class `Assembler` that handles the regular assembly process and the class `AlignFinder` that handles the incorporation of the left-over reads in the assembly process. Except the classes `Merger` and `AlignFinder` all classes are shared with the software LOCAS.

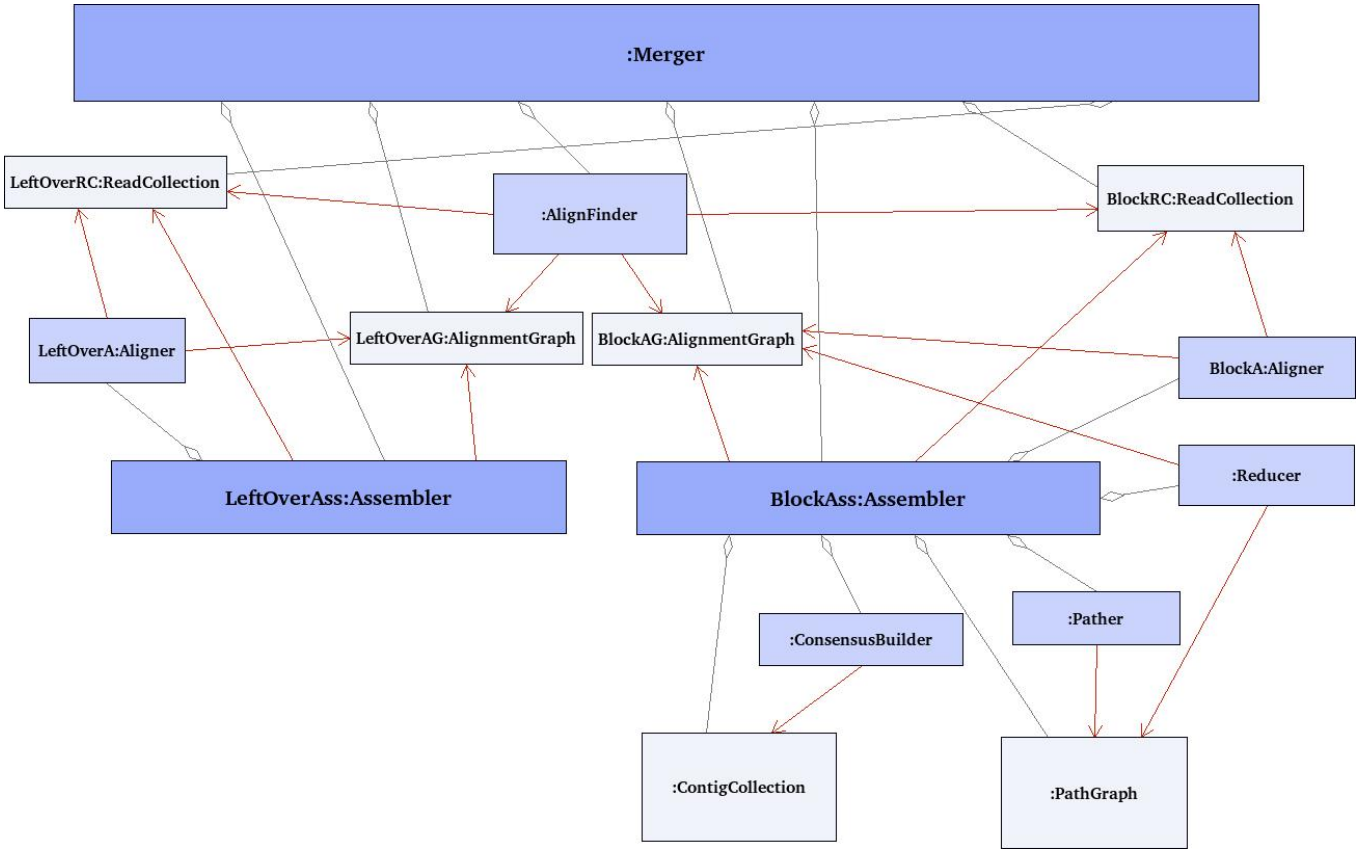

Figure 5.3.: The figure shows the object class of the software SUPERLOCAS. The object `:Merger` handles the overall workflow. The objects belonging to the first step are shown on the left. This step is handled by the object `LeftOverAss:Assembler`. The left-over reads are stored in the object `LeftOverRC:ReadCollection` and overlaps are determined with the object `LeftOverA:Aligner`. The left-over graph is constructed and stored as object `LeftOverAG:AlignmentGraph`. In the second step each block is assembled separately. The process is handled by the object `BlockAss:Assembler`. The block reads are loaded in the object `BlockRC:ReadCollection` and the overlaps are calculated with the object `BlockA:Aligner` that constructs also the overlap graph represented by the object `BlockAG:AlignmentGraph`. Next, overlaps between the block reads and left-over reads are determined with the object `:AlignFinder`, the overlaps are added to the overlap graph represented by object `BlockAG:AlignmentGraph` and the overlapping left-over reads are copied to the `BlockRC:ReadCollection` object. Finally, the extended graph is processed and contigs are produced by the `BlockAss:Assembler` object using the `:Reducer`, `:Pather` and `:ConsensusBilder` object.

constructs the overlap graph via the classes `ReadCollection` and `Aligner`, respectively. In addition, the class `Assembler` handles the transformation of the overlap graph into a path graph and its reduction, the determination of the final paths and the determination of contigs via the classes `Reducer`, `Pather` and `ConsensusBilder`, respectively. The classes `AlignFinder`, `Aligner`, `Reducer`, `Pather` and `ConsensusBilder` represent different steps in the workflow that manipulate objects of the classes `AlignmentGraph`, `PathGraph` and `ContigCollection`, which represent blue prints of the overlap graph, path graph and the set of contigs, respectively. The class `ReadCollection` is used to load the input reads and to store the read data for the rest of the workflow.

In Figure 5.3, the used objects of the software is shown. The software works in a serial manner, executing two main steps. In the first step, the left-over reads are loaded and overlap alignments between them are calculated. In the second step, each block is assembled individually by incorporating left-overs reads.

The first step handles the left-over graph construction and works as follows. The object `:Merger` instantiates the objects `LeftOverRC:ReadCollection`, `LeftOverAG:AlignmentGraph` and `LeftOverAss:Assembler`. The `LeftOverAss:Assembler` object calls `LeftOverRC:ReadCollection` to load the left-overs reads and instantiates the `LeftOverA:Aligner` object that creates the left-over graph. The left-over graph is stored as `LeftOverAG:AlignmentGraph` object. While the objects `LeftOverRC:ReadCollection` and `LeftOverAG:AlignmentGraph` are kept until the end of the overall workflow, the object `LeftOverAss:Assembler` is immediately deleted by `:Merger`.

In the second step, the overlap graph for each block is constructed, extended with left-over reads and contigs are reported. First, the objects `BlockRC:ReadCollection`, `BlockAG:AlignmentGraph` and `BlockAss:Assembler` are instantiated by the object `:Merger`. The `BlockAss:Assembler` object handles the following steps: The reads of the block are loaded by the object `BlockRC:ReadCollection`. The object `BlockA:Aligner` is instantiated to construct and modify the overlap graph of the block reads. The overlap graph is stored in the `BlockAG:AlignmentGraph` object.

Next, the object `:Merger` instantiates the object `:AlignFinder`, which detects overlap alignments between the left-over read set and the block read set, represented by the objects `LeftOverRC:ReadCollection` and `BlockRC:ReadCollection`, respectively. All left-over reads that overlap with the block reads are added to the object `BlockRC:ReadCollection`. Further, the overlap graph in the object `BlockAG:AlignmentGraph` is updated with the new overlap information by copying the corresponding vertices and edges from the `LeftOverRC:ReadCollection` and `LeftOverAG:AlignmentGraph`.

The object `BlockAss:Assembler` handles the last steps of the workflow. The objects `:Reducer`, `:Pather` and `:ConsensusBilder` are instantiated and executed iteratively. The object `:Reducer` transforms the overlap graph in the object `BlockAG:AlignmentGraph` into a path graph that is stored as `:PathGraph` ob-

ject. The reduction and determination of the final paths in the path graph are handled by the object `:Pather`. Finally, the object `:ConsensusBilder` determines and reports the consensus sequences of the contigs that are represented by the object `:ContigCollection`.

6. Evaluation and Comparison with Existing Assemblers

In this chapter, we present evaluations of LOCAS for *de novo* assemblies at a low sequencing depth. In addition, we evaluated SUPERLOCAS in various resequencing scenarios. Therefore, SUPERLOCAS was applied as part of the SHORE [OSC+08, SOO+] framework to handle its reassemble step. For both tools, we present comparisons to current state-of-the-art assembly tools.

LOCAS and SUPERLOCAS were evaluated in three studies using short read data at low sequencing depths. In the first study, *de novo* assemblies of small genomic regions were simulated. The performance of LOCAS was compared to the short read assembly tools VELVET [ZB08, ZMMB09], EULER-SR [CP08], ABySS [SWJ+09] and SOAPdenovo [LLZ+09, LZR+10]. In the second study, we simulated a homology-guided assembly of a divergent strain of Arabidopsis *thaliana*. Since the other assemblers ABySS, SOAPdenovo and EULER-SR did not perform well enough for data with a low sequencing depth (shown in the first study), we only compared SUPERLOCAS and VELVET. In the third study, we evaluated LOCAS and SUPERLOCAS in a homology-guided assembly using Illumina reads from the resequencing project of *A. thaliana*, the 1001 Genomes Project (http://1001genomes.org/). In this study, we compared both tools with VELVET.

6.1. De Novo Assembly of Simulated Data

For the simulation studies of *de novo* assemblies, we generated Illumina GAIIx reads using METASIM [ROA+08] for the first and forth chromosome of *A. thaliana* Col-0. We used an error model for Illumina reads with a read length of 80 bp, which was estimated by resequencing the strain Col-0 and aligning the reads against the sequence of Col-0 [F Ott, pers. comm.]. Paired end reads were generated with an insert size of 300 bp and 200 bp, and a standard deviation of 30 bp and 20 bp for the first and fourth chromosome, respectively.

The simulated reads were assigned to the reference sequence corresponding to their origin positions and partitioned into blocks of length 10 kb. If two reads of the same mate-pair were assigned to different blocks, both reads were assigned to the block with the smaller index. Assemblies of the blocks, which correspond to local regions in the target genome, were performed separately using the assembly tools LOCAS, ABySS, EULER-SR, VELVET and SOAPdenovo. We ran all assemblers

using a wide range of parameter settings to show the achievable results with the respective assemblers.

For the performance analysis of the assembly tools, each local assembly was investigated separately. We used different measures to evaluate the assemblies. For each measure, the average value of all local assemblies was calculated. Measures that consider the length of the contigs are the $avgN50$ size, $avgN90$ size, the average mean, minimal and maximal contig size. In our study, the $N50$ size is defined as the length of the longest contig such that all contigs of equal or longer length cover at least 50% of the positions of the block sequence. The $N90$ size is defined analogous to the $N50$ size. The $avgN50$ size and the $avgN90$ size are defined as the average $N50$ and $N90$ size, respectively. For measures that consider the contig lengths, only *valid* contigs, which match the target sequence with at most 10% mismatches and have a minimal length of 100 bp, were considered. In addition, the average coverage $avgCOV$ of the original block sequence with all valid contigs and the average error rate $avgERR$ were determined. The error rate is the total number of errors divided by total length of all contigs of a block. The total number of errors comprises the number of mismatches in the alignment of all valid contigs plus the lengths of other contigs that could not be aligned.

The combination of $avgN50$ size and $avgERR$ rate showed to be a good estimate of assembly quality. Thus, we restricted the comparisons that are presented in the following to these two measures. For the other measures, see Chapter C.

6.1.1. Evaluation of Assembly for the First Chromosome of A. thaliana at a Sequencing Depth of 7.5x

For the first study, we used the first chromosome of *A. thaliana* Col-0 as target genome and simulated reads at a sequencing depth of 7.5x. The results are shown in Figure 6.1. For $avgERR$ values lower than 1.5%, LOCAS performed best with a maximum $avgN50$ size of $4,558$ bp. For an $avgERR$ higher than 1.5%, VELVET performed best with a maximum $avgN50$ size of $5,500$ bp. EULER-SR performed well in respect to the $avgN50$ size, but had high $avgERR$ values of 5 to 11%. The $avgERR$ values of ABySS were with at most 1.2% very low, but the tool showed a low $avgN50$ size of at most $2,577$ bp. The $avgERR$ values were even lower for SOAPdenovo, while the $avgN50$ size was the lowest for all assemblers with a maximum of $1,991$ bp. We examined also CPU time and RAM needed to assemble the whole data set for each tool. The best performing method regarding CPU time was VELVET with 10 min in average, followed by LOCAS with 19 min, SOAPdenovo with 22 min and EULER-SR with 140 min. LOCAS and EULER-SR used only 18 MB of RAM, VELVET used 87 MB and SOAPdenovo used 236 MB.

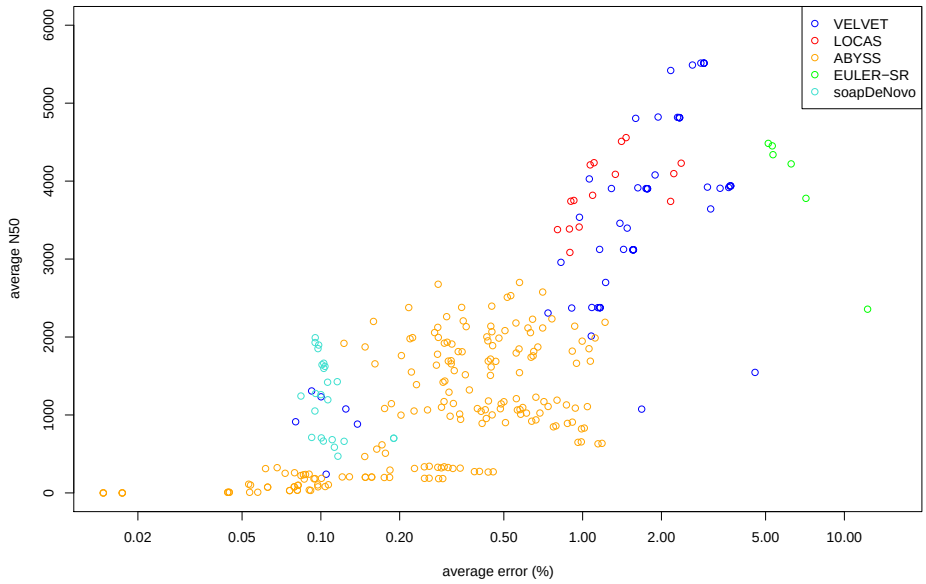

Figure 6.1.: Performance comparison of low sequencing depth assemblies. Illumina GAIIx reads of the first chromosome of *A. thaliana* Col-0 were simulated at a sequencing depth of 7.5x. The reads were assigned to the reference sequence corresponding to their origin position and partitioned into blocks of a length of 10 kb. The $avgN50$ size (average $N50$) is plotted against the $avgERR$ (average error) for the assembly tools LOCAS, EULER-SR, ABySS, VELVET and SOAPdenovo. For each assembler, several runs are displayed corresponding to the different parameter settings. The data points of ABySS are drawn in orange, EULER-SR in green, LOCAS in red, VELVET in blue and SOAPdenovo in turquoise. Each point corresponds to a run.

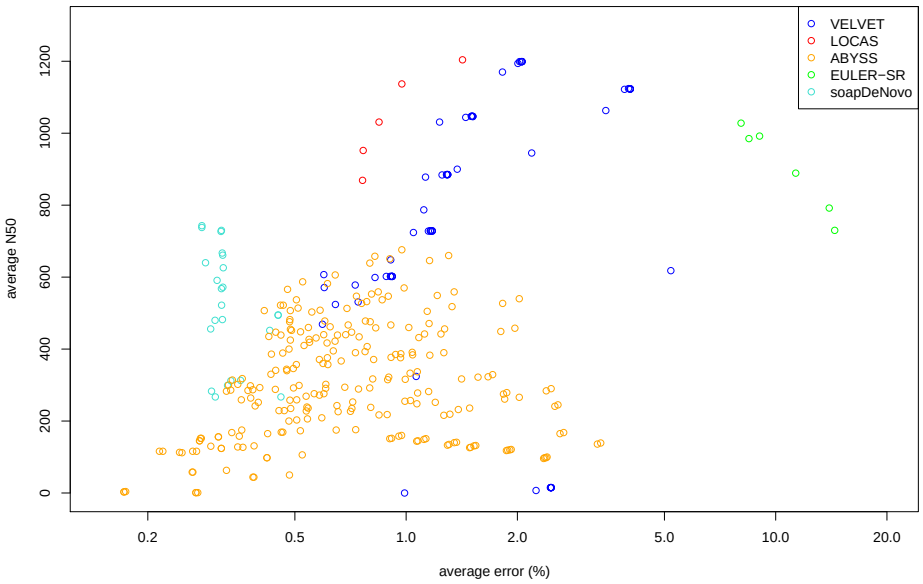

Figure 6.2.: Performance comparison of assemblies with a sequencing depth of 5x for the first chromosome of *A. thaliana* Col-0. The reads were assigned to the reference sequence corresponding to their origin position and partitioned into blocks of a length of 10 kb. The $avgN50$ size (average $N50$) is plotted against the $avgERR$ (average error rate) for the assembly tools LOCAS, EULER-SR, ABySS, VELVET and SOAPdenovo. For each assembler, several runs are displayed corresponding to the different parameter settings. The data points of ABySS are drawn in orange, EULER-SR in green, LOCAS in red, VELVET in blue and SOAPdenovo in turquoise.

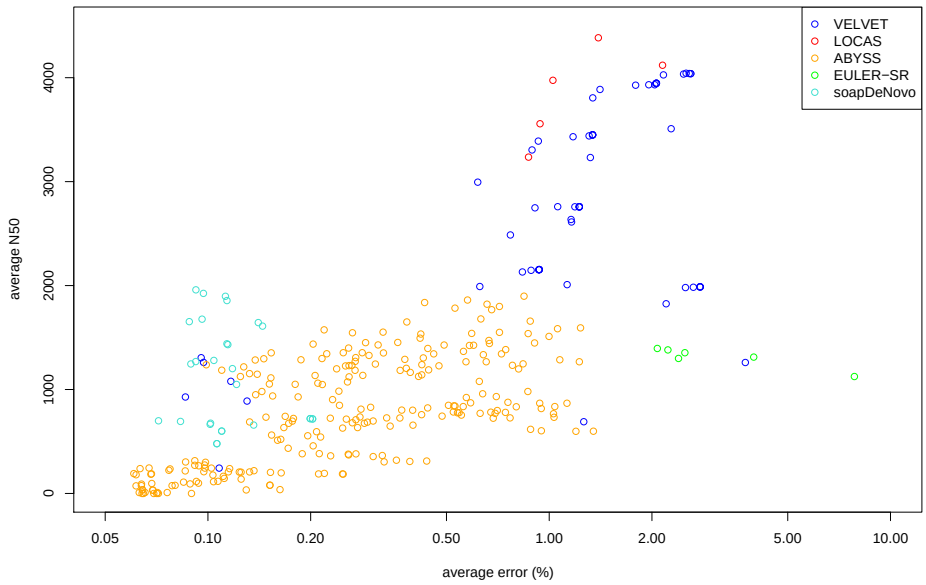

Figure 6.3.: Performance comparison of assemblies with a sequencing depth of 7.5x for the fourth chromosome of *A. thaliana* Col-0. After the reads were assigned to their origin position in the reference sequence, they were partitioned into blocks of a length of 10 kb. The $avgN50$ size (average $N50$) is plotted against the $avgERR$ (average error) for the assembly tools LOCAS, EULER-SR, ABySS, VELVET and SOAPdenovo. For each assembler, several runs are displayed corresponding to the different parameter settings. Each data point corresponds to one run. The data points of ABySS are drawn in orange, EULER-SR in green, LOCAS in red, VELVET in blue and SOAPdenovo in turquoise.

6.1.2. Evaluation of Assembly for the First Chromosome of A. thaliana at a Sequencing Depth of 5x

The second study was performed like the previous, while reads were simulated at a lower sequencing depth of 5x. The results are shown in Figure 6.2. LOCAS performed best with an $avgN50$ size of $1,204$ bp and an$avgERR$of 1.4% in the best run. VELVET showed a maximum $avgN50$ size of $1,199$ bp with an$avgERR$of 2%. For smaller $avgERR$ sizes of less than 2%, the best $avgN50$ size of VELVET was $1,170$ bp. EULER-SR showed an $avgERR$ that ranged between 8% and $14,4$%. The$avgERR$values were low for SOAPdenovo while the maximum $avgN50$ size was 743 bp. ABySS showed the lowest $avgN50$ sizes in this comparison.

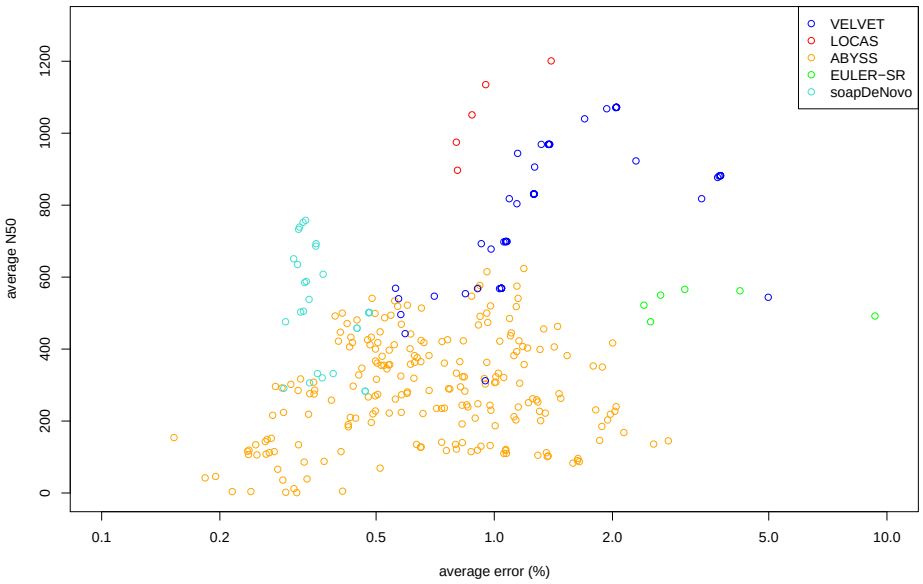

Figure 6.4.: Performance comparison of assembly with a sequencing depth of 5x for the fourth chromosome of *A. thaliana* Col-0. The reads were assigned to their origin position in the reference sequence that was partitioned into blocks of length 10 kb. The $avgN50$ (average $N50$) is plotted against the $avgERR$ (average error) for the assembly tools LOCAS, EULER-SR, ABySS, VELVET and SOAPdenovo. For each assembler, several runs are displayed corresponding to the different parameter settings. Each data point corresponds to one run. The data points of ABySS are drawn in orange, EULER-SR in green, LOCAS in red, VELVET in blue and SOAPdenovo in turquoise.

6.1.3. Evaluation of Assembly for the Fourth Chromosome of A. thaliana at a Sequencing Depth of 5x and 7x

In order to validate the results of the previous studies, we performed a similar study by changing the target genome from the first to the fourth chromosome of *A. thaliana*. The reads were simulated at a sequencing depth of 5x and 7.5x. The results are shown in Figure 6.3 and 6.4. For a sequencing depth of 7.5x, LOCAS performed best with an $avgN50$ size of $4,384$ bp and an $avgERR$ of 1.4% in its best run. At the same average error range, VELVET showed an $avgN50$ size of $3,887$ bp. VELVET showed a maximum $avgN50$ size of $4,043$ bp showing an $avgERR$ of 2.5%. The $avgERR$ of EULER-SR ranged between 2% and 7.8%. Its maximum $avgN50$ size was the lowest with $1,395$ bp. The maximum $avgN50$ sizes of SOAPdenovo and ABySS were $1,959$ bp and $1,898$ bp, respectively. Both assemblers showed again a low $avgERR$.

In the second study with a sequencing depth of 5x, LOCAS was the best performing assembly tool regarding the $avgN50$ size and $avgERR$. It showed an $avgN50$ size of 1135 bp and an $avgERR$ of 0.095% in its best run. VELVET performed second best with an $avgN50$ size of 1072 bp and an $avgERR$ of 2% in its best run. The other assembly tools showed the same tendencies in their results as in the previous studies. EULER-SR showed high error rates ranging between 2.4% and 9% with a best $avgN50$ size of 566 bp. SOAPdenovo and ABySS showed again low $avgN50$ sizes while performing good considering the error rate. SOAPdenovo achieved in its best run an $avgN50$ size of 758 bp while ABySS showed an $avgN50$ of 768 bp in its best run. The $avgERR$ of SOAPdenovo ranged between 0.15% and 2.7%. For ABySS the $avgERR$ ranged between 0.29% and 0.48%.

6.2. Homology-Guided Assembly of Simulated Data

To evaluate performance in a homology-guided assembly approach, we simulated a resequencing study of an artificial *A. thaliana* strain using a sequencing depth of 7.5x. First, we artificially generated a target genome by introducing SNPs, insertions and deletions into the reference sequence. The frequency of SNPs, deletions and small insertions was modeled according to a set of polymorphism from *A. thaliana* strains produced by the 1001 Genomes Project (www.1001genomes.org). This synthetic strain was used to simulate paired-end Illumina reads using METASIM. Paired-end reads were generated with an insert size of 200 bp and a standard deviation of 20 bp.

In the next step, reads were aligned to the reference genome Col-0 using SHORE [OSC+08, SOO+]. A read is deemed alignable if the alignment contains a maximum of six mismatches and three gaps. Reads of a long insertion or reads spanning a position with a long deletion in the genome are usually not alignable. Of several alignment tools that are supported by SHORE, we preferred GenomeMapper [SHO+09] for this study because it allows for high edit distances,

allows gaps and has a high sensitivity.

Next, the chromosomes were partitioned into blocks of 25 kb for the reassembly step using SHORE. This was done using regions with zero coverage or repetitive regions as natural borders and by using a static maximum block size. Almost 2.5% of the reads were non-alignable and were defined as left-over reads. The left-over reads of all chromosomes were pooled since they could also not be separated in a real resequencing project.

The analysis was performed as for the first study in Section 6.1, with the following alteration: a contig was determined as valid if the global alignment to the target sequence had at most 10% mismatches and if the length of the contig was at least 500 bp. The measures were calculated not for each single block but for the pooled contigs of all 100 sequential blocks, called the *100-block*. Thus, contigs were also considered in the analysis if they span two blocks with the help of left-over reads.

6.2.1. Evaluation of Homology-Guided Assembly for an Artificial A. thaliana Strain

Assemblies of the blocks incorporating the left-over reads were performed by applying the assembly tools SUPERLOCAS and VELVET. As VELVET does not provide a special mode for left-over incorporation, we used VELVET as follows: for each local assembly the complete set of left-over reads was given as an additional input. We omitted EULER-SR, SOAPdenovo and ABYSS due to their insufficient performance in the first study on data with low sequencing depth. In Figure 6.5, the $avgN50$ sizes and $avgERR$ values are shown for different runs of VELVET and SUPERLOCAS. SUPERLOCAS performed best regarding $avgN50$ size and $avgERR$. In addition, the results vary only slightly for different runs. We observed a maximum $avgN50$ size of $3,132$ bp and $2,446$ bp for SUPERLOCAS and VELVET, respectively. The error rates range from 0.17% to 0.21% for SUPERLOCAS and from 0.09% to 0.53% for VELVET. The average maximum contig size of SUPERLOCAS was larger (with up to $17,821$ bp) in comparison to VELVET (up to $12,996$ bp). The CPU runtimes per run ranged from 3h 12min for SUPERLOCAS to 7h 51min for VELVET. SUPERLOCAS took 433 MB of RAM and VELVET took 224 MB. In addition, we examined the contigs of both tools regarding the appearance of insertion regions. These regions are present in the target genome but not in the reference genome. Most insertion regions with a length of at least 100 bp can only be assembled with the help of left-over reads. Figure 6.6 shows the number of insertion regions of different length that were assembled without errors by SUPERLOCAS and VELVET. Both tools performed in this task equally well, assembling a similar number of the insertions.

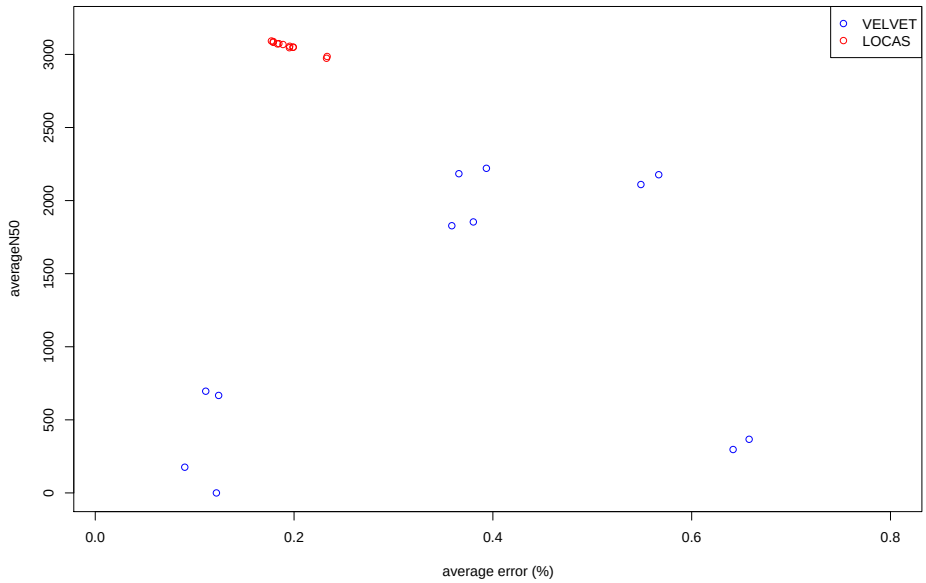

Figure 6.5.: Performance comparison of homology-guided assemblies on simulated data. We simulated a resequencing study of an artificial *A. thaliana* strain using a sequencing depth of 7.5x. The simulated Illumina reads were aligned to the reference genome Col-0 and partitioned into blocks of 25 kb using SHORE. The assembly tools SUPERLOCAS and VELVET were applied to assemble the mapped reads of the first chromosome and the remaining left-over reads. The $avgN50$ size (average $N50$) for the assembly tools SUPERLOCAS and VELVET (in left-over incorporation mode) is plotted against the $avgERR$ (average error). SUPERLOCAS is displayed in red and VELVET in blue.

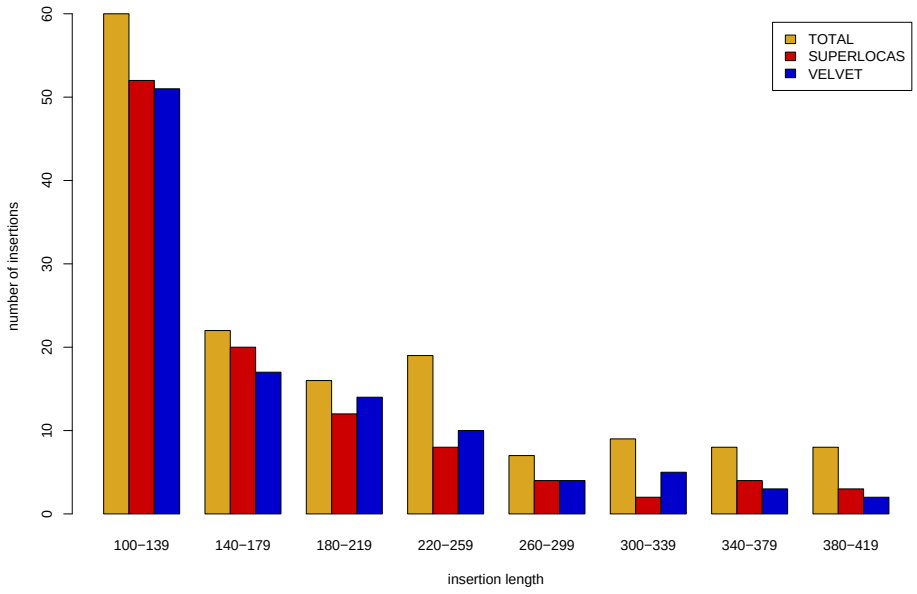

Figure 6.6.: Number of detected insertion regions in a homology-guided assembly on simulated data. For the artificial *A. thaliana* strain in the simulated resequencing study, the total insertion regions in the target genome are plotted for different region lengths. In addition, the number of error-free regions assembled by VELVET and by SUPERLOCAS are shown.

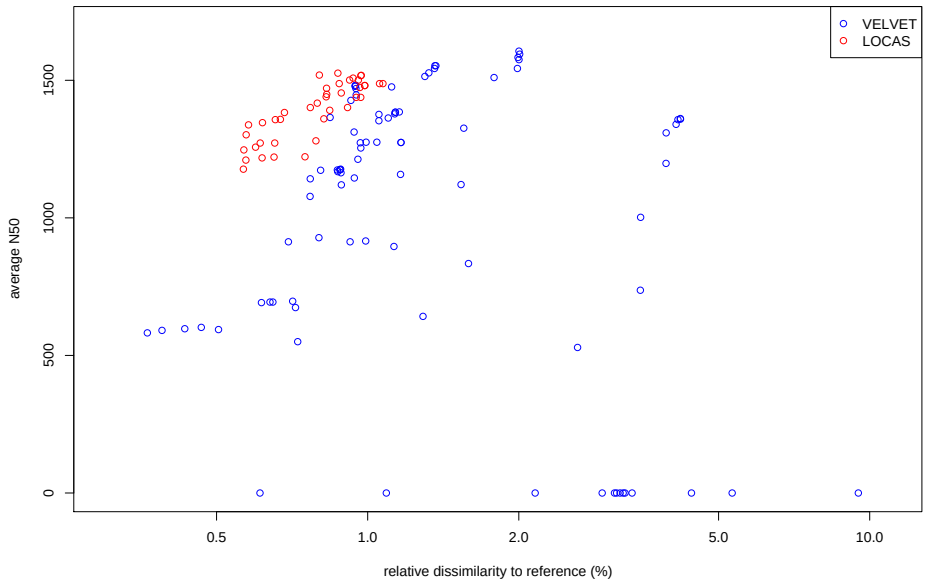

Figure 6.7.: Performance comparison of reassemblies on real world data without utilizing left-over reads. Paired-end reads were produced by Illumina GAIIx with a length of 80 bp to a depth of 7.5x for the first chromosome of *A. thaliana* Ler-1. Reads were aligned against the complete reference sequence (Col-0) and partitioned into blocks of length 25 kb using SHORE. LOCAS and VELVET are applied in paired-end mode for all blocks. The x-axis shows the $avgN50$ size (average $N50$) and the y-axis the $avgERR$ (average error). The runs of LOCAS produced with different parameter setting are drawn in red and those of VELVET in blue.

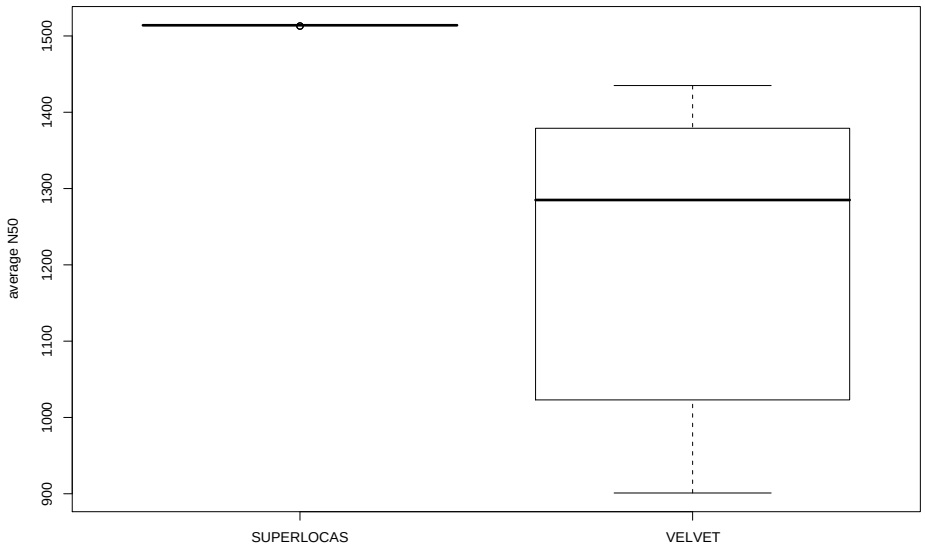

Figure 6.8.: Performance comparison of reassemblies on real world data utilizing left-over reads. Illumina reads of the first chromosome of *A. thaliana* strain Ler-1 were aligned against the reference sequence (Col-0) and partitioned into blocks of length 25 kb using SHORE. Local assemblies were performed with SUPERLOCAS and VELVET by incorporating left-over reads. While SUPERLOCAS provides algorithms specifically adjusted to this task, VELVET had to assemble each block with the complete set of left-over reads. A boxplot of the $avgN50$ (average $N50$) sizes of both assemblers is shown.

6.3. Application to Real Data

To test performance on real world data, we used sequence reads from the first chromosome of *A. thaliana* strain Landsberg erecta (Ler-1) produced within the 1001 Genomes Project [WM09]. Ler-1 was sequenced on the Illumina GAIIx with 80 bp paired-end reads to a depth of 7x. Reads were aligned against the complete reference sequence and the first chromosome of the reference sequence was partitioned into blocks of at most 40 kb using GenomeMapper [SHO+09] and SHORE [OSC+08].

For sequencing data of the *A. thaliana* Ler-1 strain, the original genome sequence was not available and, thus, the performance analysis differed in some points from the analysis of simulated data. As a proxy for the original sequence, we used the reference sequence. Assembled contigs of all blocks were pooled and aligned against the whole reference genome. If no left-over reads were provided for assembly, the performance was evaluated as for the simulated data. If left-over reads should be incorporated in the assembly, the evaluation differed. The error rate was calculated by considering only contigs that had a minimum similarity of 75% with the sequence of their $100 - block$. All non-alignable contigs were not considered for analysis since these contigs do not have to be erroneous but can belong to other regions in the Ler-1 strain covered only by left-over reads.

6.3.1. Evaluation of Homology-Guided Assembly Without Incorporating Left-Over Reads

We applied both tools to assemble the reads of each block separately. Left-over reads were not provided to the assemblers. Instead of the $avgERR$, we estimated the average relative dissimilarity to the reference sequence over all blocks, denoted as $avgDISS$. For $avgDISS$ values higher than 1%, VELVET performed best considering $avgN50$ size, while for $avgDISS$ values lower than 1%, LOCAS showed the best $avgN50$ sizes, see Figure 6.7. We observed a maximum $avgN50$ size of $1,606$ bp and $1,526$ bp for VELVET and SUPERLOCAS, respectively.

6.3.2. Evaluation of Homology-Guided Assembly Incorporating Left-Over Reads

We then evaluated SUPERLOCAS and VELVET on real world data while incorporating left-over reads. For VELVET, each block was assembled with the complete set of left-over reads as in the second study. Contigs were determined as valid if they featured a similarity with the reference sequence of at least 75%. We allowed this high percentage of dissimilarity since contigs that are constructed with the use of left-over reads often belong to insertion regions not represented in the reference genome. The $avgDISS$ to the reference genome was not estimated since it would be increased by contigs that are build from left-over reads and represent

insertion regions. Consequently, the $avgDISS$ does not reflected the average error rate over all contigs. The $N50$ sizes were higher for SUPERLOCAS with values consistently about 1500 bp. The $N50$ sizes of VELVET ranged between 901 bp and $1,435$ bp, showing much higher sensitivity to parameter choice. A boxplot of the $N50$ values of both assemblers is shown in Figure 6.8. Furthermore, SUPERLOCAS performed best regarding CPU runtime. One run of SUPERLOCAS was on average completed in 2h 8min, compared to an average running time of 7h 32min for VELVET. VELVET performed best considering RAM usage with only 1.73 GB, while SUPERLOCAS used 3.99 GB of RAM.

7. Discussion

In the first evaluation study, *de novo* assemblies of the first chromosome of *A. thaliana* at a low sequencing depth of 7.5x were simulated. The assembly tools ABySS [SWJ+09] and SOAPdenovo [LLZ+09] produced the lowest contig sizes in comparison to the other tools. Possibly, these short contig sizes occur since both tools are designed for a high sequencing depth. At least internal parameters of these tools have to be adjusted before they can be applied to data of low sequencing depth. The assembler EULER-SR [CP08] performed better considering contig sizes but showed very high error rates. The assembly tool operates with an initial step for error correction that substitutes k-mers in the read sequences that occur with a low relative frequency in the whole read set by highly similar k-mers with a high relative frequency. In case of low sequencing depth, this procedure could introduce errors in the read sequences since the relative difference of k-mer frequencies is very low. Thus, correct k-mers may be substituted by false k-mers. VELVET produced longer contigs sizes in comparison to LOCAS [KOS+] while showing higher error rates at the same time. These longer sizes may result from an improved repeat handling strategy that is implemented in VELVET [ZMMB09]. In contrast to other methods, the algorithm tries to resolve also repeat regions of higher complexity. The repeat resolution algorithm is applied iteratively to resolve repeats. Some repeats become only resolve-able if other repeats have been resolved already. Thus, more ambiguous regions can be spanned in the assembly. While longer contigs can be produced, the approach seems to introduce more errors into the assembly. In contrast, LOCAS achieved a good tradeoff between low error rates and high contigs sizes.

We validated our results from the first study by performing similar studies with an even lower sequencing depth of 5x and on an additional chromosome, the fourth chromosome of *A. thaliana*. While SOAPdenovo and ABYSS showed error rates and contig sizes that differed by a constant factor between all three experiments, EULER-SR showed an even larger error rate for a sequencing depth of 5x in comparison to a depth of 7.5x. This increase in the error rate follows from the decreased amount of read data. As mentioned above, the error correction of EULER-SR relies on a high sequencing depth. Consequently, the performance of the correction will decrease with a low sequencing depth. In comparison to the other assemblers, the contig sizes of EULER-SR and VELVET decreased to a larger extend when moving from chromosome one to chromosome four. In contrast to the other assembly tools, LOCAS performed constantly regarding the contig size on both chromosomes. Thus, LOCAS seems to perform more robustly than VELVET when being applied to different target chromosomes. When we ran the

assembly tools with different sets of parameters, the results of LOCAS differ less than those from VELVET and ABySS. We conclude that LOCAS is less sensitive to the parameter choice. This is an important feature of an assembler since a manual parameter search is very time consuming and requires a certain expertise.

Considering runtime, VELVET was the best performing method in the first study. The other tools, including LOCAS, needed twice the time, expect from EULER-SR which needed 14 times as much. The different runtimes of VELVET and LOCAS are a result of their different assembly strategies. VELVET utilizes the de Bruijn graph approach that skips the calculation of pairwise overlap alignments between reads, while LOCAS calculates pairwise overlap alignments. These calculations increase the runtime of LOCAS significantly. The overlaps are represented in an overlap graph and since more overlaps are detected, the size and the complexity of the graph is larger compared to the corresponding de Bruijn graph. Consequently, the following procedures applied to the graph during the workflow require also more runtime. On the one hand, VELVET is superior regarding runtime. However, on the other hand is is very sensitive to different parameter settings. Hence, a lot of different settings have to be tested to assure that the best setting is found, which also consumes runtime. Considering RAM usage, LOCAS and EULER-SR performed best. In contrast, VELVET and SOAPdenovo required about four times and 11 times as much space, respectively. In LOCAS, local copies of objects are avoided leading to a low RAM requirement.

In the second study, which simulated a resequencing project, we compared VELVET and SUPERLOCAS [KOS+]. SUPERLOCAS performed best considering the contig size and error rate. VELVET performed worse considering the contig size since it does not distinguish between left-over reads and reads that are assigned to blocks and, thus, treats these reads equally. Consequently, also erroneous left-over reads and false overlaps are introduced in the de Bruijn graph which leads to more branches and cycles in the graph. Finally, this results in shorter contigs. Due to its specialized method for incorporating left-over reads in the assembly, SUPERLOCAS produced longer contigs with a lower error rate. SUPERLOCAS does not treat reads equally, but uses different overlap alignment constraints for different reads, resulting in a more reliable incorporation of left-over reads. Further, repetitive left-over reads, which contain k-mers of a high frequency within the left-over read set, are discarded before they are considered for incorporation. This decreases the number of repetitive regions which often introduce errors in the assembly. Both assembly tools assemble an equal amount of longer insertion regions, which are regions that do not appear in the reference but in the target genome. These regions can only be assembled utilizing left-over reads. Thus, we can conclude that both tools are able to incorporate these reads in the assembly.

In a third study, we proved our previous results on real world data at a sequencing depth of 7x. We evaluated LOCAS like in the first study by assembling reads that had been assigned to blocks. SUPERLOCAS was evaluated like in the second study by incorporating left-over reads in the assembly of the blocks. First, we compared the performance of LOCAS and VELVET: Compared to VELVET,

LOCAS performed again very robust for different parameter settings. Similar to previous studies, LOCAS showed lower error rates than VELVET. In this study, the error rate is measured as the rate of the relative dissimilarity of contigs to the reference genome, which is approximately the sum of the error rate and the relative difference between target and reference genome. While VELVET showed larger contig sizes for high error rates, LOCAS experienced larger contigs for low error rates. Again, LOCAS achieved a better compromise between contig size and error rate. In the second part of this study, we performed assemblies of blocks by utilizing left-over reads and compared SUPERLOCAS and VELVET considering the produced contig sizes. SUPERLOCAS performed superior to VELVET for this special task. By including only high quality overlaps between left-over reads and reads of blocks, SUPERLOCAS retains the low complexity and size of its overlap graph. Thus, longer contigs can be reported. Similar to the previous study, SUPERLOCAS showed a robust performance for different parameter settings while VELVET's results strongly depend on the parameter choice.

In the second and third study, we also compared SUPERLOCAS and VELVET regarding their runtime. The runtime of VELVET was 2 to 3.75 times higher than the runtime of SUPERLOCAS. VELVET had to assemble all left-over reads over and over again for each block, while SUPERLOCAS provides a method to calculate a pre-assembly of the left-over reads, which can be later used for each block. Considering RAM usage, VELVET performed best, using only half of the RAM of SUPERLOCAS.

The presented results suggest that the overlap-layout-consensus approach implemented in LOCAS and SUPERLOCAS is better suited for low sequencing depth assembly than the de Bruijn paradigm used by VELVET, EULER-SR, SOAPdenovo and ABYSS. We optimized overlap alignments between reads to span even regions that are very sparsely covered with reads. However, by calculating exact alignments instead of matches of k-mers, the number of overlaps increases leading to a graph of higher complexity compared to the respective Bruijn graph. This high complexity can not be reduced by using a coverage-based cutoff used in de Bruijn graph approaches, which rejects regions from the assembly that are covered by a very low number of reads. The insufficient amount of reads permits the estimation of a reasonable value for a coverage-based cutoff. In addition, the probability of using false overlaps increases for alignment calculations that allow for several mismatches compared to exact matches of k-mers. Since optimal alignments are calculated, the construction of the overlap graph is slower than the construction of the de Bruijn graph. Nevertheless, for lower error rates, longer contigs and a higher overall coverage of the sequenced genome are the payoff for dealing with a more complex graph.

We believe that utilizing alignment positions of reads and incorporating left-over reads, as implemented in SUPERLOCAS, is a good compromise between resource heavy *de novo* assembly and simple homology-guided assembly approaches. Similar to *de novo* assembly, the integration of left-over reads allows for identification of highly polymorphic regions and insertions. In addition, our assembly approach

makes use of exact positions of aligned reads in a reference genome. This reduces the assembly complexity and the number of false overlaps. Further, we think that the assembly produced by our approach might be less affected by repetitive regions than *de novo* assemblies because some repeats are already detected during the alignment step. Since SUPERLOCAS distinguishes between left-over reads and aligned reads, it can incorporate of left-over reads more reliably than other short read assemblers. By creating an overlap graph for all left-over reads only once, SUPERLOCAS can re-use this graph for the assemblies of the blocks and, thus, performs much faster than other assemblers.

8. Conclusion

The study of sequence variations like SNPs, indels and longer variant regions plays an important role in the investigation of disease development and changes in physical characteristics of organisms. Since the introduction of new generation of sequencing technologies in 2005, the cost of resequencing has been reduced by an order of magnitude and the number of resequenced organisms has been increasing steadily.

The aim of resequencing projects is the detection of sequence variations between closely related species. While existing approaches are capable of detecting SNPs and indels, they do not address highly polymorphic regions and longer insertions sufficiently. With the aim of also revealing longer sequence variations, we investigated and extended the existing approach of homology-guided assembly.

We designed and implemented algorithms for resequencing projects of large genomes that are performed with short read data of low sequencing depth. In our homology-guided assembly approach, reads are aligned to a reference genome of a highly related organism. The reference genome is partitioned into blocks and reads aligned to one block are reassembled separately by incorporating left-over reads. Our algorithms for reassembly are based on an overlap-layout-consensus approach, which represents the input set of reads and their possible overlaps in an overlap graph. Since mismatches are allowed in the used overlap alignment, this approach is well suited for short read data of low sequencing depth. Our algorithms for reassembly are implemented in the assembly tools LOCAS and SUPERLOCAS.

LOCAS is specifically designed for *de novo* assemblies at a low sequencing depth and, thus, it is capable to assemble reads of local regions in the context of resequencing projects. SUPERLOCAS, the extension of LOCAS, allows for the execution of multiple reassemblies of consecutive blocks and handles the incorporation of a huge amount of left-over reads in the local assemblies. SUPERLOCAS was to the specific scenario of resequencing adapted. Alignment positions of reads on the reference genome can be utilized for the calculation of overlap alignments. This reduces the complexity of the assembly and the chance of false overlaps, which leads to longer and less erroneous contigs.

The performances of both tools were evaluated in two studies that simulated resequencing projects at a low sequencing depth of at most 7.5x. Further, an additional study was performed using real world data from the 1001 Genomes Project. In these studies, local assemblies of blocks were performed. When SUPERLOCAS was applied, left-over reads were additionally incorporated into these local assemblies. We compared LOCAS and SUPERLOCAS with other state-of-

the-art assemblers for short read data. LOCAS and SUPERLOCAS achieved better results than the other short read assemblers or at least similar. For local reassemblies at a low sequencing depth, VELVET [ZB08, ZMMB09] and LOCAS performed best considering contig size and error rate. However, LOCAS seems to be less sensitive to the choice of parameters, while producing contigs that show a good trade-off between error rate and size. When incorporating left-over reads into the local reassemblies, longer insertion regions could be assembled with SUPERLOCAS and VELVET. However, SUPERLOCAS proved to be much faster and more robust to different parameter settings than VELVET. In addition, due to its ability to efficiently incorporate left-over reads, SUPERLOCAS produced longer contigs.

SUPERLOCAS has successfully been used for the assembly of various *A. thaliana* genomes such as Ler, C24, Bur-0 and Kro-0 in the context of the 1001 Genomes Project [SOO+].

Within recent years, the sequencing technologies have steadily improved. The direction of this rapid development is not always easy to anticipate, e.g., three years ago, the short length of produced reads was a challenging task for assembly whereas today this read length has been already tripled for some technologies. Assembly algorithms that process this kind of data have to be steadily adjusted to the fast changes. At the same time, assembly algorithms will gain from improvements such as longer read lengths, which opens up new opportunities in assembly.

We assume that our method will become increasingly valuable with future improvements in sequencing technologies such as longer reads. Our overlap-layout-consensus approach can benefit from longer reads since it is more robust to sequencing errors at the end of reads. Furthermore, computing overlap alignments rather than using exact matches of k-mers will become more important since longer sub-sequences have a higher probability of containing sequencing errors. In addition, an increased overlap length between reads will contribute to the reliability of left-over read recruitment. Finally, the potential of homology-guided assembly will grow steadily with increasing numbers of completely sequenced genomes.

Bibliography

[AM97] E.L. Anson and E.W. Myers. ReAligner: a program for refining DNA sequence multi-alignments. *Journal of Computational Biology*, 4(3):369–383, 1997.

[AR09] F. Armougom and D. Raoult. Exploring microbial diversity using 16S rRNA high-throughput methods. *J Comput Sci Syst Biol Volume*, 2(1):069–092, 2009.

[AS98] C. Armen and C. Stein. A 2 2/3 superstring approximation algorithm. 1998.

[BBS+08] D.R. Bentley, S. Balasubramanian, H.P. Swerdlow, G.P. Smith, J. Milton, C.G. Brown, K.P. Hall, D.J. Evers, C.L. Barnes, H.R. Bignell, et al. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456(7218):53–59, 2008.

[Ben06] D.R. Bentley. Whole-genome re-sequencing. *Current opinion in genetics & development*, 16(6):545–552, 2006.

[BJS+02] S. Batzoglou, D.B. Jaffe, K. Stanley, J. Butler, S. Gnerre, E. Mauceli, B. Berger, J.P. Mesirov, and E.S. Lander. ARACHNE: a whole-genome shotgun assembler. *Genome research*, 12(1):177, 2002.

[BMK+08] J. Butler, I. MacCallum, M. Kleber, I.A. Shlyakhter, M.K. Belmonte, E.S. Lander, C. Nusbaum, and D.B. Jaffe. ALLPATHS: de novo assembly of whole-genome shotgun microreads. *Genome Research*, 18(5):810, 2008.

[BvdSZ+02] RW Bulterman, FW van der Sommen, G. Zwaan, T. Verhoeff, AJM van Gasteren, and WHJ Feijen. On computing a longest path in a tree. *Information Processing Letters*, 81(2):93–96, 2002.

[CP08] M.J. Chaisson and P.A. Pevzner. Short read fragment assembly of bacterial genomes. *Genome Research*, 18(2):324, 2008.

[DLBH07] J.C. Dohm, C. Lottaz, T. Borodina, and H. Himmelbauer. SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome research*, 17(11):1697, 2007.

[DLBM07] M. De La Bastide and WR McCombie. Assembling genomic DNA sequences with PHRAP. *Current protocols in bioinformatics/editoral board, Andreas D. Baxevanis...[et al.]*, 2007.

[DWRR08] Andreas Doring, David Weese, Tobias Rausch, and Knut Reinert. Seqan an efficient, generic c++ library for sequence analysis. *BMC Bioinformatics*, 9(1):11, 2008.

[EFG+09] J. Eid, A. Fehr, J. Gray, K. Luong, J. Lyle, G. Otto, P. Peluso, D. Rank, P. Baybayan, B. Bettman, et al. Real-time DNA sequencing from single polymerase molecules. *Science*, 323(5910):133, 2009.

[FB09] P. Flicek and E. Birney. Sense from sequence reads: methods for alignment and assembly. *Nature Methods*, 6:S6–S12, 2009.

[HBB+08] T.D. Harris, P.R. Buzby, H. Babcock, E. Beer, J. Bowers, I. Braslavsky, M. Causey, J. Colonell, J. DiMeo, J.W. Efcavitch, et al. Single-molecule DNA sequencing of a viral genome. *Science*, 320(5872):106, 2008.

[HCD+04] P. Havlak, R. Chen, K.J. Durbin, A. Egan, Y. Ren, X.Z. Song, G.M. Weinstock, and R.A. Gibbs. The Atlas genome assembly system. *Genome research*, 14(4):721, 2004.

[HFF+08] D. Hernandez, P. François, L. Farinelli, M. Østerås, and J. Schrenzel. De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. *Genome Research*, 18(5):802, 2008.

[HM99] X. Huang and A. Madan. CAP3: A DNA sequence assembly program. *Genome research*, 9(9):868, 1999.

[HMN09] N. Homer, B. Merriman, and S.F. Nelson. BFAST: an alignment tool for large scale genome resequencing. *PLoS One*, 4(11):e7767, 2009.

[HWA+03] X. Huang, J. Wang, S. Aluru, S.P. Yang, and L.D. Hillier. PCAP: a whole-genome assembly program. *Genome research*, 13(9):2164, 2003.

[IW95] R.M. Idury and M.S. Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2(2):291–306, 1995.

[JRB+07] W.R. Jeck, J.A. Reinhardt, D.A. Baltrus, M.T. Hickenbotham, V. Magrini, E.R. Mardis, J.L. Dangl, and C.D. Jones. Extending assembly of short DNA sequences to handle error. *Bioinformatics*, 23(21):2942, 2007.

[KJ56] J.B. Kruskal Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[KM95] J.D. Kececioglu and E.W. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13(1):7–51, 1995.

[KOS+] J. D. Klein, S. Ossowski, K. Schneeberger, D. Weigel, and D.H. Huson. LOCAS - A low coverage assembly tool for resequencing projects. *Genome Biology and Evolution, under review.*

[KS05] H. Kaplan and N. Shafrir. The greedy algorithm for shortest superstrings. *Information Processing Letters*, 93(1):13–17, 2005.

[KUA+07] J.O. Korbel, A.E. Urban, J.P. Affourtit, B. Godwin, F. Grubert, J.F. Simons, P.M. Kim, D. Palejev, N.J. Carriero, L. Du, et al. Paired-end mapping reveals extensive structural variation in the human genome. *Science*, 318(5849):420, 2007.

[LD10] H. Li and R. Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26(5):589, 2010.

[LHW+09] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078, 2009.

[LLF+09] R. Li, Y. Li, X. Fang, H. Yang, J. Wang, K. Kristiansen, and J. Wang. SNP detection for massively parallel whole-genome resequencing. *Genome research*, 19(6):1124, 2009.

[LLKW08] R. Li, Y. Li, K. Kristiansen, and J. Wang. SOAP: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713, 2008.

[LLT+03] J.H. Leamon, W.L. Lee, K.R. Tartaro, J.R. Lanza, G.J. Sarkis, A.D. deWinter, J. Berka, and K.L. Lohman. A massively parallel PicoTiterPlate based platform for discrete picoliter-scale polymerase chain reactions. *Electrophoresis*, 24(21):3769–3777, 2003.

[LLZ+09] R. Li, Y. Li, H. Zheng, R. Luo, H. Zhu, Q. Li, W. Qian, Y. Ren, G. Tian, J. Li, et al. Building the sequence map of the human pangenome. *Nature biotechnology*, 28(1):57–63, 2009.

[LRD08] H. Li, J. Ruan, and R. Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome research*, 18(11):1851, 2008.

[LSL+09] B. Langmead, M.C. Schatz, J. Lin, M. Pop, and S.L. Salzberg. Searching for SNPs with cloud computing. *Genome Biol*, 10(11):R134, 2009.

[LTPS09] B. Langmead, C. Trapnell, M. Pop, and S.L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[LYL+09] R. Li, C. Yu, Y. Li, T.W. Lam, S.M. Yiu, K. Kristiansen, and J. Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966, 2009.

[LZR+10] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome research*, 20(2):265, 2010.

[LZZ+08] H. Lin, Z. Zhang, M.Q. Zhang, B. Ma, and M. Li. ZOOM! Zillions of oligos mapped. *Bioinformatics*, 24(21):2431, 2008.

[MEA+05] M. Margulies, M. Egholm, W.E. Altman, S. Attiya, J.S. Bader, L.A. Bemben, J. Berka, M.S. Braverman, Y.J. Chen, Z. Chen, et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380, 2005.

[Met09] M.L. Metzker. Sequencing technologies-the next generation. *Nature Reviews Genetics*, 11(1):31–46, 2009.

[MG80] AM Maxam and W. Gilbert. Sequencing end-labeled DNA with base-specific chemical cleavages. *Methods in enzymology*, 65(1):499, 1980.

[MN03] J.C. Mullikin and Z. Ning. The phusion assembler. *Genome research*, 13(1):81, 2003.

[Mos] Mosaik. Website. Available online at `http://bioinformatics.bc.edu/marthlab/Mosaik`; visited on November 4th 2010.

[MPG+09] I. MacCallum, D. Przybylski, S. Gnerre, J. Burton, I. Shlyakhter, A. Gnirke, J. Malek, K. McKernan, S. Ranade, T.P. Shea, et al. ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads. *Genome biology*, 10(10):R103, 2009.

[MSD+00] E.W. Myers, G.G. Sutton, A.L. Delcher, I.M. Dew, D.P. Fasulo, M.J. Flanigan, S.A. Kravitz, C.M. Mobarry, K.H.J. Reinert, K.A. Remington, et al. A whole-genome assembly of Drosophila. *Science*, 287(5461):2196, 2000.

[Mye95] E.W. Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290, 1995.

[Mye05] E.W. Myers. The fragment assembly string graph. *Bioinformatics*, 21(suppl 2), 2005.

[OSC+08] S. Ossowski, K. Schneeberger, R.M. Clark, C. Lanz, N. Warthmann, and D. Weigel. Sequencing of natural strains of Arabidopsis thaliana with short reads. *Genome research*, 18(12):2024, 2008.

[PKS04] M. Pop, D.S. Kosack, and S.L. Salzberg. Hierarchical scaffolding with Bambus. *Genome Research*, 14(1):149, 2004.

[Pop04] M. Pop. Shotgun sequence assembly. *Advances in computers*, 60:193–248, 2004.

[PPDS04] M. Pop, A. Phillippy, A.L. Delcher, and S.L. Salzberg. Comparative genome assembly. *Briefings in bioinformatics*, 5(3):237, 2004.

[PS08] M. Pop and S.L. Salzberg. Bioinformatics challenges of new sequencing technology. *Trends in Genetics*, 24(3):142–149, 2008.

[PSTU73] H. Peltola, H. Soderlund, J. Tarhio, and E. Ukkonen. Algorithms for some string matching problems arising in molecular genetics. *Information Processing*, 83:53–64, 1973.

[PSU84] H. Peltola, H. Söderlund, and E. Ukkonen. SEQAID: A DNA sequence assembling program based on a mathematical model. *Nucleic Acids Research*, 12(1Part1):307, 1984.

[PT01] P.A. Pevzner and H. Tang. Fragment assembly with double-barreled data. *Bioinformatics*, 17(suppl 1):S225, 2001.

[PTW01a] P.A. Pevzner, H. Tang, and M.S. Waterman. A new approach to fragment assembly in DNA sequencing. In *Proceedings of the fifth annual international conference on Computational biology*, page 267. ACM, 2001.

[PTW01b] P.A. Pevzner, H. Tang, and M.S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences of the United States of America*, 98(17):9748, 2001.

[RKD+09] T. Rausch, S. Koren, G. Denisov, D. Weese, A.K. Emde, A. Doring, and K. Reinert. A consistency-based consensus algorithm for de novo and reference-guided sequence assembly of short reads. *Bioinformatics*, 25(9):1118, 2009.

[RLD+09] S.M. Rumble, P. Lacroute, A.V. Dalca, M. Fiume, A. Sidow, and M. Brudno. SHRiMP: accurate mapping of short color-space reads. *PLoS Comput Biol*, 5(5):e1000386, 2009.

[ROA+08] D.C. Richter, F. Ott, A.F. Auch, R. Schmid, and D.H. Huson. MetaSim - A Sequencing Simulator for Genomics and Metagenomics. *PLoS One*, 3(10):3373, 2008.

[RSP+02] T.D. Read, S.L. Salzberg, M. Pop, M. Shumway, L. Umayam, L. Jiang, E. Holtzapple, J.D. Busch, K.L. Smith, J.M. Schupp, et al. Comparative genome sequencing for discovery of novel polymorphisms in Bacillus anthracis. *Science*, 296(5575):2028, 2002.

[Rus09] N. Rusk. Cheap third-generation sequencing. *Nature Methods*, 6(4):244, 2009.

[SC75] F. Sanger and A.R. Coulson. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase* 1. *Journal of Molecular Biology*, 94(3):441–446, 1975.

[SHO+09] K. Schneeberger, J. Hagmann, S. Ossowski, N. Warthmann, S. Gesing, O. Kohlbacher, and D. Weigel. Simultaneous alignment of short reads against multiple genomes. *Genome Biol*, 10:R98, 2009.

[SOO+] K. Schneeberger, S. Ossowski, F. Ott, J.D. Klein, C. Lanz, L.M. Smith, J. Cao, J. Fitz, N. Warthmann, S.R. Henz, D.H. Huson, and D. Weigel. Homology-guided assembly of the four diverse arabidopsis thaliana genomes ler, c24, bur-0 and kro-0. *PloS Genetics, under review*.

[SPR+05] J. Shendure, G.J. Porreca, N.B. Reppas, X. Lin, J.P. McCutcheon, A.M. Rosenbaum, M.D. Wang, K. Zhang, R.D. Mitra, and G.M. Church. Accurate multiplex polony sequencing of an evolved bacterial genome. *Science*, 309(5741):1728, 2005.

[SSK+] M.C. Schatz, D. Sommer, D. Kelley, M. Pop, et al. Assembly of Large Genomes with Cloud Computing. In preparation.

[SSPL08] S.L. Salzberg, D.D. Sommer, D. Puiu, and V.T. Lee. Gene-boosted assembly of a novel bacterial genome from very short reads. *PLoS Comput Biol*, 4(9):e1000186, 2008.

[STK10] E.E. Schadt, S. Turner, and A. Kasarskis. A Window into Third Generation Sequencing. *Human molecular genetics*, 2010.

[SWJ+09] J.T. Simpson, K. Wong, S.D. Jackman, J.E. Schein, S.J.M. Jones, and İ. Birol. ABySS: A parallel assembler for short read sequence data. *Genome research*, 19(6):1117, 2009.

[TS09] C. Trapnell and S.L. Salzberg. How to map billions of short reads onto genomes. *Nature biotechnology*, 27(5):455, 2009.

[TY02] S.H. Teng and F. Yao. Approximating shortest superstrings. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 158–165. IEEE, 2002.

[WHW+05] N. Whiteford, N. Haslam, G. Weber, A. Prügel-Bennett, J.W. Essex, P.L. Roach, M. Bradley, and C. Neylon. An analysis of the feasibility of short read sequencing. *Nucleic Acids Research*, 33(19):e171, 2005.

[WM09] D. Weigel and R. Mott. The 1001 genomes project for Arabidopsis thaliana. *Genome Biology*, 10(5):107, 2009.

[WSJH07] R.L. Warren, G.G. Sutton, S.J.M. Jones, and R.A. Holt. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, 23(4):500, 2007.

[ZB08] D.R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome research*, 18(5):821, 2008.

[ZMMB09] D.R. Zerbino, G.K. McEwen, E.H. Margulies, and E. Birney. Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. *PLoS One*, 4(12):e8407, 2009.

A. Presentations

A.1. Talks

- "Algorithms to Support Resequencing with Ultra Short Reads", BCI 2008, 9th September 2008, Trieste
- "LOCAS - a low coverage assembler for short reads", Short-SIG 2009 (special interest group meeting at ISMB), 28 th June 2009, Stockholm
- "LOCAS - a low coverage assembler for short reads", TüBiT 2010, 21st May 2010, Tübingen
- "LOCAS - a low coverage assembler for short reads", GCB 2010, 21st September 2010, Braunschweig

A.2. Poster

- "LOCAS - A new low coverage assembler for short reads", presented at TüBit 2009, Tübingen
- "LOCAS - A new low coverage assembler for short reads", presented at ISMB 2009, Stockholm

A.3. Articles

- "LOCAS - a low coverage assembly tool for resequencing projects", short paper, appeared in abstract book of GCB 2010
- "Homology-guided Assembly of the Four Diverse Arabidopsis thaliana Genomes Ler, C24, Bur-0 and Kro-0", under review at Plos Genetics
- "LOCAS - a low coverage assembly tool for resequencing projects", submitted to Bioinformatics

B. Manual

B.1. Introduction

LOCAS is a program to assemble short reads of second generation sequencing technologies. It explicitly handles low coverage data by allowing mismatches in the overlap alignment of reads.

An extra module, called SUPERLOCAS, provides some additional features for resequencing projects. In a resequencing project reads are mapped onto a closely related reference genome and a consensus from the mapped reads is calculated as an approximation of the new genome sequence. (Highly polymorphic regions and insert sites are not covered with this consensus.) SUPERLOCAS can be used to incorporate unmapped reads into the assembly of mapped regions and elongate this consensus. Further, SUPERLOCAS takes advantage of given mapping positions of reads. Both tools are written in C++.

B.2. Availability

Binaries and source code can be downloaded from `http://www-ab.informatik.uni-tuebingen.de/software/locas`.

B.3. Installing

We provide binaries for LOCAS and SUPERLOCAS that are compiled on a LINUX system (Redhat, 64-bit). Additionally, the source code is available.

B.4. License Details

The source code is distributed under the terms of the GNU General Public License.

B.5. Author

Juliane D. Klein

B.6. Running LOCAS

You can run LOCAS from the command line as follows:

```
LOCAS -I input reads.fasta -O output folder -F fasta -L 23 -S 2
```

LOCAS will assemble all reads in the file "inputreads.fasta" by calculating overlap alignments with a minimal length of 23 and a maximum of 2 mismatches. The result is written to folder output-folder. LOCAS will create this folder if it does not exist.

Required options:

$-I$ $\langle string \rangle$	defines the full name (including location) of your read file
$-O$ $\langle string \rangle$	the name of the new folder which LOCAS will create for its output
$-F$ $fasta$ or	
$-F$ $fastq$	chooses between fasta and fastq file formats

Additional options:

$-C$ $\langle int \rangle$ $\langle int \rangle$	the first number defines the minimal length and the second number the minimal coverage of the contigs that should be reported (default: 0 0)
$-S$ $\langle int \rangle$	the maximal number of allowed mismatches in an overlap alignment between two reads (default: 4)
$-L$ $\langle int \rangle$	the minimal allowed length of an overlap alignment between two reads (default: 27)
$-P$ $kmer$ $\langle int \rangle$	the length of the sub-sequence (kmer) which has to be equal in two reads before an overlap alignment is calculated (default: 13)

B.6.1. How to choose the parameters kmer size and overlap length

First LOCAS searches for pairs of possibly overlapping reads. This is done by looking for equal kmers (sub-sequences of length k) between two reads. After this filtering step, the real overlap alignments are calculated for all reads which share a kmer.

The parameter $-K$ controls the filtering step. The user can define for which length equal sub-sequences are detected between reads. With the parameter $-L$ the minimal length of an overlap alignment is set.

B.6.2. Example of a LOCAS run

The data for an example can be found in the sub-folder "testset_locas". Open a shell and change into this folder. Then execute the following command:

```
LOCAS -I simulated_reads.fasta -O my_locas_out -F fasta -L 21 -S 2
```

You can compare your results to those in sub-folder "locas_out". Now, you can experiment with the parameter a little. For example, you can discard some contigs by applying:

```
LOCAS -I simulated_reads.fasta -O my_locas_out2 -F fasta
 -L 21 -S 2 -C 3 100.
```

Or you can change the minimal overlap length between reads by executing:

```
LOCAS -I simulated_reads.fasta -O my_locas_out3 -F fasta -L 19 -S 2
```

B.7. Running SUPERLOCAS

SUPERLOCAS is especially designed for use in resequencing projects. Here all reads are mapped against the genome of a highly related species. Continuously mapped sub-regions can be defined as blocks. All reads can be assigned to one block or to a set of left-over reads.

Using SUPERLOCAS, you can reassemble the blocks and try to incorporate left-over reads. Reads from different blocks should be placed in different files, and the left-over reads should be placed in their own files, too.

Before you can start SUPERLOCAS, you have to create two special input-files. The SUPERLOCAS_Inputfile describes the input for SUPERLOCAS. The SUPERLOCAS_Outputfile describes the output for SUPERLOCAS. The first line in the SUPERLOCAS_Inputfile shows the names of all read files for the first block separated by a space character. In the second line you find all names of the read files of the second block and so on. In each line of the SUPERLOCAS_Outputfile you have to write the name of the output-folder of the corresponding block. For an example file, see the files "super_input_file" and "super_output_file" in the folder "testset_superlocas/myoutput".

SUPERLOCAS is started as follows:

```
SUPERLOCAS -I SUPERLOCAS_Inputfile -O SUPERLOCAS_Outputfile
 -LO left_over_file_1 left_over_file_2 -F fasta
```

Required options:

$-I\ \langle string \rangle$	defines location and name of your SUPERLOCAS_inputfile
$-O\ \langle string \rangle$	defines location and name of your SUPERLOCAS_outputfile
$-F\ \langle fasta \rangle$ or	
$-F\ \langle fastq \rangle$	chooses between these file formats
$-LO\ \langle string \rangle$	defines left-over files

Kmer options (optional):

$-Kmerg\ \langle int\rangle$	chooses kmer size which has to be equal in a read from the block and a read of the left-over set to calculate an overlap alignment (default: 30)
$-P\ \ kmer\ \langle\ int\rangle$	chooses kmer size which has to be equal in two block reads before an overlap alignment is calculated (default: 13)
$-K\ \langle int\rangle$	chooses kmer size which has to be equal in two left-over reads (default: 33)

Overlap length options (optional):

$-Lm\ \langle int\rangle$	chooses minimal length of an overlap alignment between a block read and a left-over read (default: 30)
$-Lt\ \langle int\rangle$	chooses minimal length of an overlap alignment between two block reads (default: 21)
$-Llo\ \langle int\rangle$	chooses minimal length of an overlap alignment between two left-over reads (default: 33)

Substitutions in overlap options (optional):

$-Sm\ \langle int\rangle$	chooses maximal number of allowed mismatches in an overlap alignment between a block read and an left-over read (default: 1)
$-St\ \langle int\rangle$	chooses maximal number of allowed mismatches in an overlap alignment between two block reads (default: 3)
$-Slo\ \langle int\rangle$	chooses maximal number of allowed mismatches in an overlap alignment between two left-over reads (default: 1)

Additional options:

$-C\ \langle int\rangle\ \langle int\rangle$	chooses with the first number the minimal length and with the second number minimal coverage of the contigs which should be reported (default: 0 0)
$-DR\ \langle int\rangle\ \langle int\rangle$	determines that all left-over reads are discarded which have more than a certain number of equal kmers in the set of left-over reads, the kmer size is defined with the first number and the second number defines the number of kmer matches (default: 21 500)

B.7.1. Running SUPERLOCAS with mapping positions

If mapping positions are also available for reads you can switch on a mode of SUPERLOCAS that will take these information into account:

$-P\ \ pos\langle int\rangle$	defines kmer size as usual, but in addition an overlap alignment is also calculated if two reads are very close to each other with respect to their mapping positions

Further alignment constraints can be defined for very close or distant reads (optional):

$-Ltn\ \langle int\rangle$	defines minimal length of an overlap alignment between two block reads which are close to each other (default: 11)
$-Ltd\ \langle int\rangle$	defines minimal length of an overlap alignment between two block reads which are distant to each other (default: 25)
$-Stn\ \langle int\rangle$	defines maximal number of allowed mismatches in an overlap alignment between two block reads which are close to each other (default: 2)
$-Std\ \langle int\rangle$	defines maximal number of allowed mismatches in an overlap alignment between two block reads which are distant to each other (default: 0)

B.7.2. Understanding the parameters of SUPERLOCAS

SUPERLOCAS distinguishes between two types of reads: reads assigned to a block (because they mapped to the same region to a reference) and reads of the left-over set. The alignment constraints for these two types of reads differ and can be defined independently with the following options:

If two reads belong to a block then parameters $-Lt\ \langle int\rangle$, $-P\ kmer\langle int\rangle$, and $-St\ \langle int\rangle$ define your alignment constraints. If both reads are left-over reads then parameter $-K\ \langle int\rangle$, $-Llo\ \langle int\rangle$, and $-Slo\ \langle int\rangle$ define the overlaps. If a block read is aligned to a left-over read then the constraints are set by $-Kmerg\ \langle int\rangle$, $-Lm\ \langle int\rangle$, and $-Slo\ \langle int\rangle$.

If mapping positions are available for the block reads then SUPERLOCAS can take them into account. This option can be switched on by applying the option $-P\ pos\ \langle int\rangle$ (instead of $-P\ kmer\ \langle int\rangle$) to define the kmer size. In this case SUPERLOCAS will not only look for equal kmers between reads to calculate an overlap alignment, but it will also try to overlap read with a very close mapping position. In this mode you can manipulate the overlap alignment constraints of two reads depending on their mapping positions. Two block reads can be classified as distant or close to each other with respect to their mapping position. In the first case the constraints for the overlap alignment are defined using $-Ltd\ \langle int\rangle$ and $-Std\ \langle int\rangle$, in the case of very close block reads with $-Ltn\ \langle int\rangle$ and $-Stn\ \langle int\rangle$.

B.7.3. Example of a SUPERLOCAS run

The sub-folder "testset_superlocas" contains the files of an example application. There are 16 blocks to assemble and 29199 left-over reads. A read file of single-end Illumina reads and one of paired-end Illumina reads belongs to each block.

You can start SUPERLOCAS by changing into the folder "testset_superlocas" and execute the following command:

```
superlocas -I myoutput/super_input_file
 -O myoutput/super_output_file -LO left_over_reads.fasta -F fasta
```

This should produce the same output as you can see in the folder output.

The two files “super_input_file” and “super_output_file” are the special input and outputfile you have to generate for use with SUPERLOCAS.

C. Supplementary Tables

Tables

Table 1A - Evaluation of low sequencing depth assembly with LOCAS.

Parameter Settings	Mean	Min	Max	N50	N75	N90	Coverage	Error	Total Error	Unmapped	All
kmer:13 -L 13 -S 4	1655.55	180	6481	4230	2758	1179	0.978185	0.003291	0.023799	493	21601
kmer:13 -L 15 -S 4	1774.28	192	6739	4558	2939	1257	0.97988	0.00305	0.014654	279	21419
kmer:13 -L 17 -S 4	1680.32	174	6432	4236	2674	1172	0.981362	0.002954	0.011056	196	21364
kmer:13 -L 19 -S 4	1541.52	159	6029	3752	2312	1019	0.981487	0.002884	0.009258	153	21327
kmer:13 -L 21 -S 4	1429.91	155	5668	3386	2086	927	0.980997	0.00287	0.008904	144	21301
kmer:13 -L 13 -S 2	1281.09	137	6084	3740	2345	948	0.983857	0.00377	0.021682	474	23269
kmer:13 -L 13 -S 3	1554.05	167	6404	4096	2641	1113	0.981248	0.003365	0.022303	470	22018
kmer:13 -L 15 -S 2	1455.62	140	6345	4087	2568	1121	0.983368	0.003374	0.013336	262	22589
kmer:13 -L 15 -S 3	1726.75	183	6699	4510	2893	1264	0.98089	0.003062	0.014079	271	21663
kmer:13 -L 17 -S 2	1428.57	137	6083	3818	2414	1062	0.982809	0.003145	0.010918	201	22302
kmer:13 -L 17 -S 3	1650.79	172	6415	4208	2648	1175	0.981419	0.002904	0.010691	190	21530
kmer:13 -L 19 -S 2	1326.28	131	5711	3411	2093	926	0.981576	0.002986	0.0097	169	22096
kmer:13 -L 19 -S 3	1516.14	158	6009	3742	2295	1012	0.981554	0.002783	0.009015	150	21469
kmer:13 -L 21 -S 2	1247.87	130	5378	3085	1907	842	0.982272	0.002952	0.00894	148	22027
kmer:13 -L 21 -S 3	1411.73	153	5647	3378	2079	926	0.981649	0.002762	0.008017	127	21440

Table 1B - Evaluation of low sequencing depth assembly with VELVET.

Parameter settings: -ins_length 300 -ins_length_sd 30 -scaffolding no –shortPaired

Parameter Settings	Mean	Min	Max	N50	N75	N90	Coverage	Error	Total Error	Unmapped	All
kmer: 13 -exp_cov 5	598.52	105	3046	1075	483	86	0.861407	0.00366209	0.0168096	242	17944
kmer: 13 -exp_cov 9	2202.04	262	6216	3908	2383	859	0.897908	0.00698525	0.0335045	508	18429
kmer: 13 -exp_cov 15	2241.03	271	6247	3938	2402	864	0.896879	0.0074156	0.0367096	560	18409
kmer: 13 -exp_cov 19	2241.65	272	6247	3938	2402	864	0.89696	0.0074251	0.0367827	561	18411
kmer: 15 -exp_cov 5	1501.26	139	5819	3459	2015	609	0.92772	0.0040325	0.0138829	191	19120
kmer: 15 -exp_cov 7	2728.35	379	7482	5419	3531	1430	0.933104	0.00528194	0.021701	317	19147
kmer: 15 -exp_cov 13	2839.6	422	7515	5513	3553	1437	0.928376	0.00613774	0.0290426	444	19055
kmer: 15 -exp_cov 15	2842.26	424	7515	5513	3557	1437	0.928471	0.00616019	0.0290645	444	19057
kmer: 15 -exp_cov 17	2843.16	424	7515	5513	3557	1437	0.928518	0.00617715	0.0290814	444	19058
kmer: 15 -exp_cov 19	2843.63	424	7515	5513	3557	1437	0.928566	0.00618453	0.0290881	444	19059
kmer: 17 -exp_cov 5	1807.47	182	6306	4028	2435	839	0.940874	0.00352805	0.0106063	137	19351
kmer: 17 -exp_cov 7	2443.27	304	6917	4805	3088	1238	0.943525	0.00430522	0.0159486	226	19359
kmer: 19 -exp_cov 3	699.415	108	3357	1233	645	174	0.920507	0.00079603	0.0010019	3	18913
kmer: 19 -exp_cov 5	1684.76	149	5832	3536	2134	781	0.945449	0.00331392	0.00972579	125	19448
kmer: 19 -exp_cov 11	2045.39	224	6162	3904	2413	981	0.943914	0.00419285	0.0174666	256	19391
kmer: 19 -exp_cov 15	2047.78	224	6166	3903	2416	980	0.943835	0.00422818	0.0176849	259	19390

kmer: 19 -exp_cov 19	2048.25	224	6166	3903	2416	980	0.943889	0.00423199	0.0176887	259	19391
kmer: 21 -exp_cov 9	1671.97	162	5405	3124	1927	757	0.945886	0.00373973	0.0143337	206	19453
kmer: 21 -exp_cov 11	1674.73	164	5402	3118	1925	755	0.944869	0.00384582	0.0155348	227	19434
kmer: 21 -exp_cov 13	1676.24	164	5400	3119	1925	754	0.944829	0.00386169	0.015613	228	19434
kmer: 23 -exp_cov 5	1241.43	127	4495	2308	1411	493	0.947502	0.00283099	0.00737443	89	19524
kmer: 23 -exp_cov 7	1315.99	134	4553	2372	1473	537	0.947998	0.00316813	0.00908427	115	19523
kmer: 23 -exp_cov 13	1329.6	136	4562	2377	1477	533	0.945818	0.00345702	0.0116391	159	19484
kmer: 23 -exp_cov 15	1329.77	136	4562	2377	1477	533	0.94588	0.00346121	0.0116433	159	19485
kmer: 23 -exp_cov 19	1330.19	136	4563	2377	1477	533	0.945927	0.00347087	0.011653	159	19486
kmer: 13 -exp_cov auto	969.522	122	3485	1546	843	151	0.852562	0.00650431	0.0455667	722	17612
kmer: 23 -exp_cov auto	1147.32	140	4211	2012	1158	329	0.925807	0.00277424	0.0107875	153	19062
kmer: 13 -exp_cov 3	254.867	101	936	240	132	9	0.780609	0.000526214	0.00104781	8	16166
kmer: 13 -exp_cov 7	1974.95	195	6022	3643	2233	766	0.895958	0.00654854	0.0308403	467	18427
kmer: 13 -exp_cov 11	2232.62	270	6245	3918	2395	865	0.896861	0.00731327	0.0361419	551	18408
kmer: 13 -exp_cov 13	2238.58	271	6247	3936	2400	864	0.896906	0.00738583	0.0364864	556	18410
kmer: 13 -exp_cov 17	2241.19	271	6247	3938	2402	864	0.896909	0.00741958	0.0367772	561	18410
kmer: 15 -exp_cov 3	596.808	109	2648	913	502	144	0.903247	0.000607717	0.000800946	3	18595
kmer: 15 -exp_cov 9	2812.79	409	7509	5489	3559	1460	0.930127	0.00572539	0.0262757	399	19088
kmer: 15 -exp_cov 11	2833.32	421	7516	5514	3559	1440	0.928906	0.00605275	0.0282918	431	19066

kmer: 17 -exp_cov 3	729.16	110	3423	1308	691	183	0.917313	0.000689378	0.000922136	4	18842
kmer: 17 -exp_cov 9	2500.01	328	6945	4822	3108	1249	0.941051	0.00477098	0.0194231	284	19316
kmer: 17 -exp_cov 11	2508.06	330	6933	4818	3094	1225	0.938054	0.00501897	0.0230458	348	19258
kmer: 17 -exp_cov 13	2510.61	330	6933	4815	3085	1225	0.937851	0.00506328	0.0233695	353	19254
kmer: 17 -exp_cov 15	2512.29	331	6934	4815	3087	1224	0.937785	0.00506541	0.0234603	355	19253
kmer: 17 -exp_cov 17	2512.17	331	6933	4813	3086	1224	0.937875	0.00505987	0.0234547	355	19254
kmer: 17 -exp_cov 19	2512.76	331	6933	4813	3086	1224	0.937878	0.00506152	0.0234564	355	19254
kmer: 19 -exp_cov 7	2007.77	206	6150	3905	2409	980	0.947147	0.00389499	0.0128828	174	19460
kmer: 19 -exp_cov 9	2040.06	221	6163	3914	2419	980	0.944573	0.00420417	0.0162519	234	19409
kmer: 19 -exp_cov 13	2047.04	224	6166	3903	2416	981	0.943889	0.00421784	0.017606	258	19391
kmer: 19 -exp_cov 17	2048.17	224	6166	3903	2416	980	0.94388	0.00423134	0.0176881	259	19391
kmer: 21 -exp_cov 3	633.994	108	3068	1077	564	154	0.921076	0.000936791	0.00124573	5	18948
kmer: 21 -exp_cov 5	1481.89	136	5230	2958	1802	661	0.94825	0.00311329	0.0082637	100	19520
kmer: 21 -exp_cov 7	1654.75	158	5396	3124	1929	760	0.948203	0.00339208	0.0116115	161	19494
kmer: 21 -exp_cov 15	1676.24	164	5399	3119	1925	754	0.944855	0.00386825	0.015626	228	19435
kmer: 21 -exp_cov 17	1676.37	164	5399	3119	1925	754	0.94487	0.00387045	0.0156282	228	19435
kmer: 21 -exp_cov 19	1676.49	164	5399	3119	1925	754	0.944888	0.00387389	0.0156316	228	19435
kmer: 23 -exp_cov 3	561.892	106	2703	883	473	135	0.919774	0.00111541	0.00137958	4	18954
kmer: 23 -exp_cov 9	1327.44	135	4563	2379	1479	536	0.946467	0.00341218	0.0108418	145	19497

kmer: 23 -exp_cov 11	1328.95	136	4562	2377	1477	533	0.945923	0.00345251	0.0114455	155	19487
kmer: 23 -exp_cov 17	1330.1	136	4563	2377	1477	533	0.945911	0.00346947	0.0116516	159	19486
kmer: 15 -exp_cov auto	2075.61	285	6178	3923	2342	824	0.911033	0.00554208	0.0299812	472	18722
kmer: 17 -exp_cov auto	2090.63	284	6325	4079	2470	897	0.93168	0.00444026	0.0189144	278	19143
kmer: 19 -exp_cov auto	1780.05	209	5745	3397	2045	742	0.936179	0.00370752	0.0147992	212	19241
kmer: 21 -exp_cov auto	1453.72	166	5013	2700	1605	525	0.934052	0.00316307	0.012242	175	19213

Table 1C - Evaluation of low sequencing depth assembly with EULER-SR.

Parameter Settings	**Mean**	**Min**	**Max**	**N50**	**N75**	**N90**	**Coverage**	**Error**	**Total Error**	**Unmapped**	**All**
kmer:13	1405.71	164	4647	2357	1243	240	0.8473	0.018591	0.122636	2206	17558
kmer:15	2035.51	235	6141	3779	2225	681	0.896943	0.018707	0.071304	1135	18524
kmer:17	2291.79	303	6604	4339	2596	919	0.911728	0.020649	0.05336	724	18791
kmer:19	2374.22	339	6684	4484	2732	900	0.912173	0.023134	0.051188	584	18790
kmer:21	2374.27	349	6645	4452	2704	828	0.907206	0.026845	0.053012	520	18684
kmer:23	2295.26	303	6522	4221	2480	628	0.892354	0.03035	0.062604	639	18388

Table 1D - Evaluation of low sequencing depth assembly with ABySS.

Parameter Settings	**Mean**	**Min**	**Max**	**N50**	**N75**	**N90**	**Coverage**	**Error**	**Total Error**	**Unmapped**	**All**
kmer:13 n=2 b=26 c=1.4 e=0	618.633	106	2400	848	433	48	0.807374	0.00132551	0.00773125	118	17348
kmer:13 n=2 b=26 c=1.4 e=1	626.353	107	2381	822	406	30	0.791088	0.00141369	0.00990267	155	16951
kmer:13 n=2 b=52 c=1.2 e=0	661.052	106	2525	937	492	64	0.819076	0.00157451	0.00662996	93	17658
kmer:13 n=2 b=52 c=2.0 e=1	522.453	105	2006	636	288	7	0.770684	0.00169404	0.0118594	176	16373
kmer:13 n=6 b=26 c=1.4 e=0	301.688	101	1255	322	148	8	0.747779	0.000499733	0.00288505	37	15528
kmer:13 n=6 b=26 c=2.0 e=1	282.768	101	1140	268	108	0	0.713912	0.000483216	0.00435704	57	14811

kmer:13 n=6 b=52 c=1.0 e=0	107.949	68	276	8	0	0	0.274155	0.000346943	0.000446663	0	5643
kmer:13 n=6 b=52 c=1.4 e=1	306.087	101	1257	318	140	4	0.735762	0.000601071	0.0034066	43	15274
kmer:13 n=10 b=26 c=2.0 e=0	225.416	100	813	187	84	0	0.714339	0.000320423	0.00248768	32	14880
kmer:13 n=10 b=52 c=1.0 e=0	107.942	68	276	8	0	0	0.274152	0.000346945	0.000446664	0	5643
kmer:13 n=10 b=52 c=1.0 e=1	113.846	71	291	8	0	0	0.285315	0.000297509	0.000440918	1	5871
kmer:13 n=10 b=52 c=1.4 e=0	232.351	100	858	204	99	1	0.735133	0.00042698	0.00156194	17	15327
kmer:13 n=10 b=52 c=2.0 e=0	225.721	101	815	189	86	0	0.71686	0.000392867	0.00259216	32	14932
kmer:15 n=2 b=30 c=1.0 e=0	156.061	91	431	37	0	0	0.430557	0.000592753	0.000902728	3	8833
kmer:15 n=2 b=60 c=1.2 e=0	1285.01	139	4391	2228	1343	419	0.922521	0.00154413	0.00643759	100	19309
kmer:15 n=2 b=60 c=2.0 e=1	774.969	117	2811	1107	615	77	0.867652	0.00165657	0.0104275	162	18016
kmer:15 n=6 b=30 c=1.0 e=0	153.455	91	406	35	0	0	0.428393	0.000552913	0.000812961	2	8787
kmer:15 n=6 b=30 c=1.2 e=1	1083.94	137	3817	1717	955	191	0.875401	0.0008835	0.00444816	65	18038
kmer:15 n=6 b=30 c=1.4 e=1	974.966	127	3499	1508	835	157	0.872142	0.000886096	0.00444411	64	17946
kmer:15 n=6 b=30 c=2.0 e=1	704.642	112	2677	1009	539	64	0.854341	0.000913508	0.00579969	85	17566
kmer:15 n=6 b=60 c=1.4 e=1	996.531	129	3539	1543	863	165	0.877332	0.0012555	0.0057383	82	18078
kmer:15 n=10 b=60 c=2.0 e=1	627.469	109	2503	902	475	54	0.852761	0.000897237	0.00507287	72	17525
kmer:17 n=2 b=68 c=1.2 e=0	1472.93	154	4786	2577	1545	486	0.924247	0.00149577	0.0070371	113	19188
kmer:17 n=2 b=68 c=1.2 e=1	1420	182	4386	2189	1268	272	0.885655	0.00147603	0.012182	215	18356
kmer:17 n=6 b=34 c=1.4 e=0	1138.85	132	4053	1933	1126	302	0.908508	0.000881995	0.00303901	41	18678
kmer:17 n=6 b=68 c=1.4 e=0	1172.41	134	4111	1986	1167	321	0.913599	0.00131358	0.00478942	67	18826
kmer:17 n=6 b=68 c=2.0 e=1	760.735	118	2745	1069	584	73	0.861261	0.0012687	0.00576477	79	17725
kmer:17 n=10 b=34 c=2.0 e=0	707.062	113	2758	1065	597	115	0.885226	0.000707599	0.00255526	33	18180
kmer:17 n=10 b=34 c=2.0 e=1	696.879	114	2642	984	517	61	0.852651	0.000628796	0.003117	43	17496
kmer:17 n=10 b=68 c=1.0 e=1	293.545	103	1036	252	102	0	0.721835	0.000465136	0.000730331	3	14790
kmer:17 n=10 b=68 c=2.0 e=0	717.567	114	2775	1082	611	118	0.8887	0.000966802	0.0039614	54	18261
kmer:19 n=2 b=38 c=1.4 e=0	1234.24	138	4202	2068	1204	318	0.908453	0.00121349	0.0045018	64	18805
kmer:19 n=6 b=38 c=1.4 e=1	1103.51	141	3732	1691	949	188	0.877394	0.000815607	0.00306824	42	18009
kmer:19 n=6 b=76 c=1.4 e=0	1195.76	133	4245	2124	1299	483	0.950731	0.00127938	0.00279693	30	19602
kmer:19 n=6 b=76 c=2.0 e=0	752.814	116	2835	1172	707	200	0.925958	0.00117703	0.00295503	33	19069
kmer:19 n=10 b=38 c=1.0 e=0	173.683	99	479	97	0	0	0.541871	0.00079925	0.000816189	0	11155
kmer:19 n=10 b=38 c=2.0 e=0	691.614	113	2718	1083	638	175	0.918133	0.000704966	0.00175166	20	18859
kmer:13 n=10 b=26 c=2.0 e=1	225.961	101	809	183	77	0	0.700283	0.00032107	0.00281744	36	14584
kmer:13 n=10 b=26 c=1.2 e=1	234.455	101	862	201	93	0	0.721023	0.000333209	0.00147942	17	15032
kmer:13 n=6 b=52 c=1.0 e=1	113.857	71	291	8	0	0	0.285318	0.000297508	0.000440916	1	5872
kmer:13 n=10 b=52 c=1.4 e=1	233.096	101	856	199	91	0	0.719778	0.000397486	0.00183067	21	15002

kmer:13 n=10 b=26 c=1.0 e=0	31.0096	21	74	0	0	0	0.0572849	0.000122125	0.00014746	0	1175
kmer:15 n=2 b=30 c=1.2 e=0	1251.79	134	4334	2180	1289	397	0.917431	0.00110599	0.00556166	90	19232
kmer:13 n=10 b=52 c=1.2 e=1	235.088	101	868	203	95	0	0.724311	0.000404167	0.00156518	17	15101
kmer:13 n=2 b=26 c=1.0 e=1	33.3345	22	79	0	0	0	0.060689	0.000108163	0.000174454	0	1245
kmer:15 n=2 b=30 c=2.0 e=1	763.783	116	2792	1088	599	74	0.863059	0.00117025	0.00937568	151	17892
kmer:15 n=2 b=60 c=1.0 e=0	317.506	101	1260	293	100	0	0.717079	0.000889377	0.00183652	13	15084
kmer:15 n=6 b=30 c=1.2 e=0	1086.7	125	4014	1910	1106	315	0.906071	0.000934629	0.00316592	41	18676
kmer:15 n=6 b=30 c=2.0 e=0	713.21	112	2774	1080	602	120	0.881589	0.000932658	0.00484998	70	18132
kmer:15 n=6 b=60 c=1.0 e=1	246.433	101	844	187	60	0	0.677952	0.000424978	0.00100532	7	13916
kmer:15 n=10 b=30 c=1.2 e=0	890.399	114	3541	1551	878	235	0.899675	0.000784739	0.00221735	26	18503
kmer:15 n=10 b=60 c=1.0 e=0	230.412	101	765	178	66	0	0.686389	0.000575268	0.000864052	4	14125
kmer:17 n=2 b=34 c=1.0 e=0	179.534	97	544	82	0	0	0.513658	0.000738313	0.00103859	3	10548
kmer:17 n=2 b=68 c=1.0 e=1	441.934	104	1756	509	176	0	0.758317	0.000753954	0.00176447	15	16104
kmer:17 n=2 b=68 c=2.0 e=1	794.44	120	2812	1129	620	78	0.865787	0.00146747	0.00867351	133	17869
kmer:17 n=6 b=34 c=1.2 e=1	1245.42	157	4094	1951	1099	216	0.875391	0.000756864	0.00434511	66	17983
kmer:17 n=6 b=68 c=1.4 e=1	1131.9	144	3788	1741	979	178	0.877439	0.00117106	0.00635387	97	18061
kmer:17 n=10 b=68 c=1.4 e=1	1030.29	134	3637	1616	886	151	0.872781	0.000877649	0.00447204	65	17922
kmer:19 n=2 b=38 c=1.4 e=1	1173.54	147	3837	1758	963	167	0.866869	0.00114381	0.00646855	100	17897
kmer:19 n=2 b=76 c=1.0 e=1	506.333	105	1951	617	228	0	0.772032	0.000750917	0.00171212	15	16432
kmer:19 n=6 b=38 c=1.0 e=1	190.985	100	552	101	0	0	0.551583	0.000487211	0.000538634	0	11288
kmer:19 n=6 b=38 c=1.2 e=1	1267.52	155	4140	1995	1127	244	0.88089	0.000790807	0.00280971	38	18091
kmer:19 n=6 b=76 c=1.2 e=0	1419.95	143	4882	2677	1627	647	0.956181	0.00134926	0.00280601	31	19733
kmer:19 n=6 b=76 c=2.0 e=1	730.407	117	2693	1067	621	106	0.893034	0.0010733	0.00424626	60	18369
kmer:19 n=10 b=38 c=1.2 e=1	1162.18	144	4007	1919	1130	298	0.90836	0.000600674	0.00122535	11	18625
kmer:19 n=10 b=38 c=2.0 e=1	681.257	113	2612	998	565	93	0.885518	0.00056256	0.00202335	27	18157
kmer:19 n=10 b=76 c=1.2 e=0	1250.49	129	4579	2380	1436	534	0.95084	0.00110827	0.00344969	49	19561
kmer:19 n=10 b=76 c=1.2 e=1	1233.82	149	4199	2057	1220	328	0.912934	0.000865545	0.00271929	37	18740
kmer:13 n=6 b=26 c=1.2 e=0	306.993	101	1280	335	155	11	0.752554	0.000524783	0.00248523	31	15632
kmer:15 n=6 b=60 c=1.4 e=0	1006.85	124	3723	1691	979	273	0.907474	0.00128356	0.00435885	58	18714
kmer:13 n=2 b=26 c=1.0 e=0	31.033	21	74	0	0	0	0.0573087	0.000122269	0.000147604	0	1176
kmer:15 n=2 b=30 c=1.2 e=1	1236.72	150	4083	1947	1110	239	0.885055	0.00118906	0.00996046	175	18514
kmer:15 n=2 b=60 c=1.4 e=0	1130.47	134	3947	1872	1112	329	0.917482	0.00148301	0.00674362	105	19169
kmer:13 n=6 b=26 c=1.4 e=1	304.702	101	1252	314	138	4	0.732585	0.000489172	0.00319495	42	15207
kmer:15 n=2 b=30 c=2.0 e=0	775.915	115	2901	1171	672	137	0.890941	0.00110183	0.00711426	114	18488
kmer:15 n=2 b=60 c=1.0 e=1	339.434	102	1351	314	96	0	0.708527	0.000768475	0.0022784	22	14930

kmer:15 n=6 b=30 c=1.0 e=1	160.531	94	426	30	0	0	0.434516	0.000387457	0.000759749	3	8905
kmer:13 n=2 b=52 c=1.4 e=1	630.795	107	2391	832	416	30	0.795344	0.00165013	0.0101471	156	17051
kmer:13 n=10 b=26 c=1.2 e=0	233.487	100	868	205	101	2	0.736607	0.000370546	0.00120682	12	15362
kmer:13 n=2 b=52 c=1.4 e=0	623.1	106	2408	860	443	49	0.81168	0.00155459	0.00790557	118	17451
kmer:15 n=6 b=60 c=2.0 e=0	724.467	112	2795	1096	619	125	0.886818	0.00131708	0.00590636	84	18258
kmer:17 n=2 b=34 c=1.0 e=1	188.625	99	587	78	0	0	0.514437	0.000535101	0.000975073	4	10552
kmer:17 n=2 b=68 c=1.4 e=1	1214.14	154	3902	1850	1058	202	0.882703	0.00147195	0.0105776	179	18278
kmer:17 n=2 b=68 c=2.0 e=0	817.62	120	2968	1228	716	149	0.898974	0.00144594	0.00662885	98	18571
kmer:17 n=6 b=34 c=2.0 e=1	743.929	117	2718	1047	563	69	0.856497	0.000842455	0.00409443	57	17592
kmer:17 n=10 b=34 c=1.2 e=1	1121.54	145	3883	1779	982	185	0.871741	0.00060924	0.00279041	39	17883
kmer:17 n=10 b=34 c=1.4 e=1	1006.2	132	3586	1569	857	144	0.868568	0.000592404	0.00323248	48	17819
kmer:17 n=10 b=68 c=2.0 e=1	707.598	115	2659	1001	532	62	0.856249	0.000882067	0.00451206	63	17577
kmer:19 n=2 b=38 c=1.2 e=0	1445.91	147	4759	2531	1494	428	0.912634	0.00122978	0.00531515	81	19003
kmer:19 n=2 b=38 c=2.0 e=0	777.393	119	2843	1142	649	122	0.885835	0.0011555	0.004885	69	18268
kmer:19 n=2 b=38 c=2.0 e=1	749.558	118	2679	1024	541	58	0.848288	0.00111395	0.00610242	89	17461
kmer:19 n=2 b=76 c=1.4 e=0	1261.63	140	4258	2139	1285	404	0.926861	0.00148333	0.00445885	60	19200
kmer:19 n=6 b=38 c=1.2 e=0	1324.3	138	4576	2378	1404	445	0.92205	0.000989264	0.00216632	22	18978
kmer:19 n=6 b=38 c=1.4 e=0	1153.74	132	4094	1978	1164	343	0.917808	0.000973102	0.002191	23	18873
kmer:19 n=6 b=38 c=2.0 e=0	740.985	116	2812	1145	676	181	0.91642	0.000848013	0.0018599	18	18834
kmer:19 n=10 b=76 c=1.0 e=1	323.893	103	1157	312	155	0	0.770038	0.000518042	0.00061476	1	15768
kmer:19 n=10 b=76 c=1.4 e=0	1074.02	126	4020	1921	1162	413	0.945581	0.00105202	0.00296293	38	19452
kmer:19 n=10 b=76 c=1.4 e=1	1050.32	135	3701	1696	993	245	0.908409	0.000834801	0.00314566	46	18641
kmer:13 n=10 b=26 c=1.4 e=1	232.841	101	854	198	90	0	0.717028	0.000319341	0.00174675	21	14945
kmer:13 n=10 b=26 c=1.4 e=0	232.085	100	856	202	98	1	0.732418	0.000350529	0.00147822	17	15271
kmer:17 n=6 b=34 c=1.0 e=0	172.95	97	477	78	0	0	0.508416	0.000660917	0.000791307	1	10428
kmer:13 n=2 b=26 c=2.0 e=0	515.827	104	2023	649	301	16	0.781526	0.00135664	0.00959922	143	16622
kmer:17 n=2 b=68 c=1.0 e=0	403.068	103	1632	467	175	0	0.772055	0.000931405	0.00147128	8	16371
kmer:15 n=2 b=60 c=1.4 e=1	1110.28	140	3723	1690	971	198	0.886176	0.00159586	0.0107016	179	18485
kmer:13 n=2 b=52 c=2.0 e=0	518.401	104	2027	655	306	16	0.785341	0.00158639	0.00988469	145	16703
kmer:15 n=6 b=60 c=2.0 e=1	715.98	113	2698	1025	556	67	0.85952	0.00130808	0.00686898	99	17689
kmer:17 n=2 b=34 c=1.2 e=1	1377.76	173	4309	2139	1227	258	0.88235	0.000995425	0.00932602	165	18349
kmer:17 n=6 b=68 c=1.2 e=0	1348.92	142	4599	2396	1413	421	0.917928	0.00132769	0.0044961	62	18921
kmer:15 n=10 b=60 c=1.2 e=1	933.996	122	3537	1516	831	153	0.874223	0.00094632	0.00356391	47	17971
kmer:13 n=6 b=26 c=1.0 e=1	33.3015	22	78	0	0	0	0.0606537	0.000107906	0.000174197	0	1244
kmer:15 n=10 b=60 c=1.2 e=0	923.556	115	3662	1639	931	251	0.904495	0.00100949	0.00276317	32	18610

kmer:13 n=6 b=52 c=1.4 e=0	303.031	101	1260	326	150	8	0.750921	0.000611136	0.00306061	39	15595
kmer:15 n=10 b=60 c=2.0 e=0	629.999	109	2578	955	524	101	0.879813	0.000938488	0.00428565	60	18093
kmer:15 n=10 b=60 c=1.4 e=1	839.435	116	3220	1321	722	122	0.870586	0.000922195	0.00369286	49	17892
kmer:17 n=2 b=68 c=1.4 e=0	1266.9	143	4268	2116	1272	365	0.92022	0.00150171	0.00618145	92	19079
kmer:15 n=10 b=60 c=1.4 e=0	837.432	113	3351	1420	805	205	0.900123	0.000983743	0.00292983	35	18513
kmer:13 n=10 b=52 c=1.2 e=0	234.128	100	874	207	103	2	0.73987	0.000441465	0.00128576	13	15430
kmer:17 n=2 b=34 c=1.4 e=0	1240.66	140	4241	2082	1239	350	0.916166	0.0010389	0.00505069	78	18956
kmer:17 n=6 b=34 c=1.2 e=0	1282.26	137	4446	2261	1332	386	0.912171	0.000919744	0.00302348	40	18764
kmer:13 n=10 b=26 c=1.0 e=1	33.3015	22	78	0	0	0	0.0606537	0.000107906	0.000174197	0	1244
kmer:15 n=2 b=30 c=1.0 e=1	163.627	94	457	32	0	0	0.437017	0.000436834	0.000914015	4	8958
kmer:13 n=6 b=26 c=1.0 e=0	31.0096	21	74	0	0	0	0.0572849	0.000122125	0.00014746	0	1175
kmer:13 n=6 b=26 c=1.2 e=1	310.642	101	1278	329	145	5	0.737136	0.00050118	0.00279614	36	15307
kmer:17 n=6 b=34 c=1.0 e=1	180.618	99	504	74	0	0	0.508253	0.000436061	0.000626191	1	10404
kmer:13 n=6 b=52 c=1.2 e=0	309.988	101	1294	342	159	12	0.756395	0.000632988	0.00259464	31	15711
kmer:17 n=6 b=68 c=2.0 e=0	779.122	117	2895	1170	674	136	0.893908	0.00129029	0.00500331	68	18409
kmer:17 n=10 b=34 c=1.4 e=0	1024.06	125	3833	1762	1011	256	0.904574	0.000735406	0.0020309	24	18579
kmer:17 n=10 b=68 c=1.0 e=0	268.282	102	933	233	109	0	0.734926	0.000667116	0.000857826	2	15116
kmer:17 n=10 b=68 c=1.2 e=0	1187.91	128	4325	2133	1242	347	0.912456	0.00106223	0.00359055	49	18761
kmer:17 n=10 b=68 c=1.2 e=1	1180.21	148	4040	1889	1043	201	0.876065	0.000923884	0.00453141	68	17995
kmer:17 n=10 b=68 c=1.4 e=0	1047.9	125	3889	1813	1043	267	0.908694	0.00102404	0.00334705	43	18680
kmer:19 n=2 b=76 c=1.0 e=0	453.22	103	1793	562	217	1	0.786904	0.000995465	0.00163498	10	16743
kmer:19 n=2 b=76 c=1.2 e=0	1501.68	153	4908	2700	1619	550	0.931175	0.00151096	0.00573998	87	19294
kmer:19 n=2 b=76 c=2.0 e=0	786.123	119	2863	1180	689	158	0.903152	0.00137406	0.00436742	57	18658
kmer:19 n=2 b=76 c=2.0 e=1	758.31	119	2700	1064	585	79	0.86672	0.00132708	0.00563362	80	17869
kmer:19 n=6 b=76 c=1.2 e=1	1341.7	161	4356	2207	1327	380	0.917528	0.00109446	0.00349797	49	18884
kmer:19 n=10 b=38 c=1.0 e=1	190.973	100	553	112	0	0	0.56711	0.000480332	0.000529782	0	11607
kmer:19 n=10 b=38 c=1.4 e=1	1025.61	133	3654	1656	964	234	0.904685	0.000595072	0.00160937	20	18553
kmer:19 n=10 b=76 c=2.0 e=0	700.65	113	2736	1099	649	179	0.921022	0.000921789	0.00288952	37	18936
kmer:19 n=10 b=76 c=2.0 e=1	690.445	114	2628	1012	578	95	0.888545	0.000785791	0.00339023	48	18235
kmer:15 n=6 b=60 c=1.0 e=0	233.877	101	788	181	67	0	0.686904	0.000584658	0.000941271	4	14127
kmer:17 n=6 b=34 c=2.0 e=0	762.822	116	2871	1147	653	129	0.889303	0.000889293	0.0032079	42	18277
kmer:13 n=6 b=52 c=2.0 e=1	283.912	101	1144	271	110	0	0.716867	0.000598951	0.0045545	59	14874
kmer:13 n=10 b=52 c=2.0 e=1	226.27	101	812	184	78	0	0.70283	0.000396717	0.00292257	37	14636
kmer:17 n=2 b=34 c=2.0 e=1	781.822	119	2792	1109	603	76	0.862173	0.000989861	0.00738118	118	17753
kmer:19 n=2 b=76 c=1.2 e=1	1425.73	176	4422	2233	1290	302	0.889559	0.00141873	0.00763134	122	18369

kmer:15 n=10 b=30 c=2.0 e=0	624.631	108	2561	944	513	98	0.875984	0.000726106	0.00341932	48	18009
kmer:17 n=2 b=34 c=1.4 e=1	1188.84	150	3876	1819	1028	193	0.878719	0.00099465	0.00912932	158	18156
kmer:15 n=6 b=60 c=1.2 e=1	1128.45	140	3907	1795	1015	207	0.881097	0.00127163	0.00557384	79	18174
kmer:19 n=10 b=38 c=1.2 e=0	1181.09	128	4361	2200	1329	489	0.946647	0.000899715	0.00158872	13	19463
kmer:17 n=10 b=34 c=1.0 e=1	180.607	99	504	74	0	0	0.508252	0.000436062	0.000626193	1	10404
kmer:19 n=2 b=38 c=1.0 e=1	204.752	100	686	103	0	0	0.552518	0.000626602	0.000972795	4	11340
kmer:15 n=10 b=60 c=1.0 e=1	242.526	101	820	184	60	0	0.677365	0.000411687	0.000949459	7	13914
kmer:17 n=2 b=34 c=1.2 e=0	1427.94	149	4708	2510	1497	461	0.920443	0.00103995	0.00515405	83	19171
kmer:13 n=2 b=26 c=1.2 e=1	664.498	109	2481	892	453	39	0.797742	0.00145859	0.0087474	134	17156
kmer:15 n=10 b=30 c=1.0 e=0	153.449	91	406	35	0	0	0.428392	0.000552913	0.000812961	2	8787
kmer:13 n=6 b=26 c=2.0 e=0	280.668	101	1140	273	118	2	0.727856	0.000475321	0.00385978	51	15104
kmer:19 n=2 b=76 c=1.4 e=1	1196.86	151	3867	1815	1038	224	0.886025	0.00141474	0.00650609	99	18300
kmer:13 n=2 b=52 c=1.0 e=1	116.209	71	308	9	0	0	0.287016	0.000319241	0.000573965	2	5909
kmer:13 n=2 b=26 c=1.2 e=0	653.154	106	2507	920	480	61	0.814148	0.00134764	0.0063893	93	17552
kmer:13 n=6 b=52 c=2.0 e=0	281.766	101	1144	276	119	2	0.730775	0.000591976	0.00403636	52	15166
kmer:19 n=6 b=38 c=2.0 e=1	717.202	116	2671	1051	603	101	0.889298	0.000744529	0.00226999	28	18253
kmer:19 n=2 b=38 c=1.0 e=0	193.508	99	626	105	0	0	0.555843	0.000881757	0.00106826	2	11431
kmer:15 n=10 b=30 c=1.0 e=1	160.527	94	426	30	0	0	0.434513	0.000387458	0.000759749	3	8905
kmer:17 n=10 b=34 c=1.2 e=0	1129.78	127	4152	1992	1160	320	0.907846	0.000768709	0.00224056	27	18647
kmer:19 n=6 b=76 c=1.0 e=1	331.402	103	1200	324	159	0	0.771742	0.000544322	0.00068076	2	15799
kmer:13 n=6 b=52 c=1.2 e=1	313.537	101	1291	335	149	5	0.740965	0.000604436	0.00295846	37	15386
kmer:15 n=6 b=30 c=1.4 e=0	985.68	122	3680	1656	951	261	0.902134	0.000921569	0.00314725	41	18581
kmer:15 n=10 b=30 c=1.4 e=0	824.753	113	3309	1389	783	198	0.895721	0.000765283	0.00231866	28	18416
kmer:15 n=6 b=60 c=1.2 e=0	1129.11	128	4109	2002	1175	338	0.911786	0.0012986	0.00430535	57	18812
kmer:15 n=2 b=60 c=2.0 e=0	786.841	116	2919	1190	689	142	0.895652	0.00155268	0.00798353	122	18618
kmer:15 n=2 b=30 c=1.4 e=1	1090	138	3707	1663	947	189	0.881523	0.00116096	0.00947481	164	18351
kmer:15 n=10 b=30 c=1.4 e=1	826.297	115	3178	1292	698	118	0.86618	0.000697183	0.00308677	42	17795
kmer:17 n=6 b=68 c=1.0 e=1	298.79	103	1065	259	104	0	0.722787	0.000486008	0.000793173	4	14803
kmer:19 n=6 b=76 c=1.0 e=0	260.616	101	925	237	123	0	0.759234	0.000829696	0.000873746	0	15677
kmer:19 n=2 b=38 c=1.2 e=1	1381.06	167	4328	2116	1180	222	0.870543	0.00114793	0.00703647	115	18081
kmer:19 n=10 b=76 c=1.0 e=0	249.823	101	854	223	120	0	0.758002	0.000820838	0.000839057	0	15688
kmer:19 n=6 b=76 c=1.4 e=1	1133.41	141	3832	1811	1075	282	0.913215	0.00113292	0.00346373	47	18792
kmer:15 n=10 b=30 c=1.2 e=1	898.277	120	3414	1435	784	142	0.869407	0.000705045	0.00297579	40	17865
kmer:17 n=2 b=34 c=2.0 e=0	804.73	119	2952	1209	699	143	0.895267	0.000985586	0.00558137	86	18453
kmer:15 n=10 b=30 c=2.0 e=1	621.882	109	2487	892	465	52	0.848922	0.000685109	0.00412976	59	17441

kmer:13 n=2 b=26 c=2.0 e=1	519.322	105	2001	630	282	7	0.766659	0.00144988	0.011447	172	16288
kmer:17 n=10 b=34 c=1.0 e=0	172.935	97	477	78	0	0	0.508415	0.000660915	0.000791305	1	10428
kmer:19 n=6 b=38 c=1.0 e=0	179.379	99	504	99	0	0	0.546243	0.000784262	0.000819791	0	11224
kmer:15 n=2 b=60 c=1.2 e=1	1271.66	156	4138	1988	1149	253	0.889978	0.00165177	0.0111605	190	18604
kmer:19 n=10 b=38 c=1.4 e=0	1049.68	125	3964	1873	1126	396	0.941955	0.000833013	0.00147558	12	19366
kmer:15 n=2 b=30 c=1.4 e=0	1110.99	131	3936	1847	1084	315	0.912692	0.00106551	0.00571535	92	19036
kmer:17 n=6 b=34 c=1.4 e=1	1098.64	141	3731	1688	940	167	0.872265	0.000738528	0.00465811	73	17916
kmer:17 n=6 b=68 c=1.2 e=1	1311.74	165	4231	2056	1164	238	0.880873	0.00120307	0.00632025	97	18139
kmer:13 n=2 b=52 c=1.2 e=1	672.021	109	2501	908	464	40	0.80261	0.0016831	0.00913675	137	17251
kmer:13 n=2 b=52 c=1.0 e=0	109.742	68	289	8	0	0	0.275461	0.000365297	0.000534498	1	5671
kmer:17 n=6 b=68 c=1.0 e=0	274.513	102	971	241	111	0	0.735917	0.000682832	0.0009018	3	15122

Table 1E - Evaluation of low sequencing depth assembly with soapDeNovo.

Parameter Settings	**Mean**	**Min**	**Max**	**N50**	**N75**	**N90**	**Coverage**	**Error**	**Total Error**	**Unmapped**	**All**
Kmer: 13 -M 0	384.79	104	1540	471	273	82	0.87947	0.000472357	0.0011617	12	18176
Kmer: 13 -M 1	447.511	106	1796	586	342	112	0.893773	0.000646363	0.00112631	8	18491
Kmer: 13 -M 2	498.883	107	2077	700	404	135	0.904168	0.000782716	0.00189791	20	18723
Kmer: 13 -M 3	500.571	107	2090	703	406	136	0.904755	0.000772936	0.00189871	21	18736
Kmer: 15 -M 0	487.034	108	1950	665	392	134	0.910935	0.000535884	0.00102203	8	18772
Kmer: 15 -M 1	688.446	116	2718	1051	622	223	0.926615	0.000652513	0.000949038	5	19085
Kmer: 15 -M 2	921.761	123	3635	1598	940	338	0.937523	0.000702016	0.00102623	6	19308
Kmer: 15 -M 3	931.618	123	3681	1624	957	344	0.938597	0.000712872	0.00103654	6	19331
Kmer: 17 -M 0	507.594	108	2012	712	425	150	0.92204	0.000629373	0.000922506	5	19011
Kmer: 17 -M 1	788.243	119	3007	1243	750	282	0.938112	0.000705106	0.000839215	2	19316
Kmer: 17 -M 2	1075.43	130	4096	1929	1157	441	0.94794	0.000732059	0.000950511	4	19509
Kmer: 17 -M 3	1093.6	130	4181	1991	1183	454	0.949321	0.000738146	0.000951649	4	19540
Kmer: 19 -M 0	501.361	108	1987	706	421	153	0.927751	0.000703507	0.00100412	5	19158
Kmer: 19 -M 1	801.098	120	3041	1276	782	292	0.944447	0.000778647	0.000954282	3	19467
Kmer: 19 -M 2	1043.5	130	3961	1851	1111	424	0.952787	0.000787422	0.000975081	3	19626
Kmer: 19 -M 3	1059.86	131	4026	1897	1136	437	0.954249	0.000805661	0.000980857	3	19658
Kmer: 21 -M 0	486.973	108	1921	683	410	154	0.930678	0.000789377	0.00110459	6	19260
Kmer: 21 -M 1	788.043	119	3017	1261	780	289	0.948229	0.000869027	0.0010005	2	19577
Kmer: 21 -M 2	954.212	125	3645	1644	1003	379	0.954463	0.000876887	0.00101017	2	19690

Kmer: 21 -M 3	963.704	125	3669	1664	1017	385	0.955538	0.000892282	0.00102557	2	19714
Kmer: 23 -M 0	473.524	107	1845	662	396	153	0.931956	0.000845104	0.00122676	7	19329
Kmer: 23 -M 1	750.864	118	2869	1195	733	273	0.949735	0.000905108	0.00106301	3	19637
Kmer: 23 -M 2	848.963	121	3232	1420	865	322	0.953844	0.000919877	0.00106097	2	19704
Kmer: 23 -M 3	853.987	121	3247	1427	870	325	0.954361	0.000939317	0.00115517	4	19717

Table 2A - Evaluation of homology-guided assembly on simulated data with LOCAS.

Parameter Settings	**Mean**	**Min**	**Max**	**N50**	**N75**	**N90**	**Coverage**	**Error**	**Total Error**	**Unmapped**	**All**
kmer:19 -L 17 -S 1	2330.75	500	17101	2974	1716	382	0.852468	0.00232651	0.433293	1710539	2249098
kmer:19 -L 19 -S 1	2342.08	500	17101	2986	1726	383	0.852517	0.0023325	0.423862	1645747	2249459
kmer:19 -L 21 -S 1	2466.42	500	16905	3049	1773	391	0.854089	0.00198879	0.225333	625942	2197188
kmer:19 -L 23 -S 1	2514.92	500	16946	3068	1801	396	0.85481	0.00188842	0.0802353	189403	2179452
kmer:19 -L 25 -S 1	2522.17	500	16946	3074	1804	399	0.855518	0.00184713	0.0604777	139102	2180193
kmer:19 -L 27 -S 1	2534.58	500	17431	3088	1812	402	0.856304	0.00179155	0.0491413	111117	2180768
kmer:21 -L 17 -S 1	2491.17	500	17166	3046	1780	392	0.854877	0.00195305	0.130535	319917	2183083
kmer:21 -L 19 -S 1	2487	500	17166	3050	1780	392	0.854941	0.00199277	0.135632	335507	2185910
kmer:21 -L 21 -S 1	2497.5	500	17166	3056	1782	392	0.854875	0.00195581	0.13019	319098	2183138
kmer:21 -L 23 -S 1	2524	500	17287	3072	1809	398	0.855411	0.00183345	0.0655806	151840	2179396
kmer:21 -L 25 -S 1	2532.42	500	17188	3083	1808	442	0.856056	0.00179	0.0512035	115984	2179211
kmer:21 -L 27 -S 1	2542.58	500	17189	3092	1815	446	0.856578	0.00177216	0.0443031	99251	2179700

Table 2B - Evaluation of homology-guided assembly on simulated data with VELVET.

Parameter Settings	Mean	Min	Max	N50	N75	N90	Coverage	Error	Total Error	Unmapped	All
kmer:17 -exp_cov 17	932.833	500	5034	367	0	0	0.496019	0.00657794	0.905558	17633724	1851353
kmer:17 -exp_cov 7	918.583	500	4414	297	0	0	0.47353	0.00641662	0.906551	17156517	1783589
kmer:17 -exp_cov auto	581.25	501	874	0	0	0	0.00851731	0.0012187	0.952284	568618	25613
kmer:19 -exp_cov 17	1463	501	11789	2177	1280	146	0.814079	0.00566864	0.897822	27010232	3125195
kmer:19 -exp_cov 7	1447.92	502	11911	2110	1218	135	0.806099	0.00548915	0.8976	26810127	3109346
kmer:19 -exp_cov auto	868.917	500	4006	176	0	0	0.432637	0.000900552	0.889833	11325556	1405708
kmer:21 -exp_cov 17	1464.08	500	12502	2221	1306	187	0.825782	0.0039348	0.896998	26616877	3097753
kmer:21 -exp_cov 7	1455.83	500	12445	2184	1275	183	0.820642	0.00365806	0.897175	26550914	3083186
kmer:21 -exp_cov auto	1059.17	500	6761	696	43	0	0.607963	0.0011114	0.88395	14882691	1961838
kmer:23 -exp_cov 17	1367.25	500	11448	1854	1093	0	0.813252	0.00380335	0.895977	25824966	3038006
kmer:23 -exp_cov 7	1359.92	500	11242	1828	1071	0	0.809554	0.00358695	0.896102	25770145	3026377
kmer:23 -exp_cov auto	1064.83	500	6652	667	0	0	0.598013	0.00124133	0.884599	14694516	1934311

Table 3A – Evaluation of homology-guided assembly on real world data with LOCAS (without utilizing left-over reads).

Parameter Settings	Mean	Min	Max	N50	N75	N90	Coverage	Error	Total Error	Unmapped	All
kmer:13 -L 11 -S 4	1312	500	7312	1501	744	0	0.777362	0.00492953	0.00958249	17347	3692439
kmer:13 -L 13 -S 4	1313	500	7312	1508	752	0	0.778618	0.00488424	0.00935314	16804	3725040

kmer:13 -L 15 -S 4	1353	500	9266	1519	732	0	0.77195	0.00444431	0.00801188	13301	3698437
kmer:13 -L 17 -S 4	1342	500	9463	1471	694	0	0.761737	0.00429263	0.00828065	14629	3637860
kmer:13 -L 19 -S 4	1324	500	7046	1401	661	0	0.750992	0.00424553	0.00768585	12406	3578348
kmer:13 -L 11 -S 3	1299	500	7312	1488	746	0	0.778258	0.00484103	0.0107247	22120	3719253
kmer:13 -L 11 -S 5	1316	500	7312	1518	744	0	0.775158	0.00488811	0.00970901	18067	3711252
kmer:13 -L 13 -S 3	1302	500	7312	1488	748	0	0.778408	0.00482619	0.0105579	21561	3721987
kmer:13 -L 13 -S 5	1318	500	7312	1517	747	0	0.77544	0.00487431	0.00969151	18067	3714170
kmer:13 -L 15 -S 3	1343	500	9266	1501	730	0	0.77345	0.00453375	0.00920431	17449	3701568
kmer:13 -L 15 -S 5	1357	500	9266	1526	726	0	0.76953	0.00450161	0.0087218	15702	3688235
kmer:13 -L 17 -S 3	1333	500	9463	1454	699	0	0.764372	0.00437628	0.00885896	16490	3646018
kmer:13 -L 17 -S 5	1346	500	9463	1474	687	0	0.75997	0.0043102	0.0096489	19552	3626977
kmer:13 -L 19 -S 3	1316	500	7046	1391	670	0	0.75434	0.00437027	0.00840279	14601	3590392
kmer:13 -L 19 -S 5	1326	500	7046	1401	654	0	0.750282	0.00430952	0.00911777	17320	3569304
kmer:13 -L 21 -S 3	1303	500	7046	1346	644	0	0.745886	0.00426442	0.00617173	6792	3539052
kmer:13 -L 21 -S 4	1311	500	7046	1357	638	0	0.742303	0.0041246	0.00654625	8593	3525187
kmer:13 -L 21 -S 5	1313	500	7046	1360	627	0	0.741471	0.0042235	0.00817844	14027	3517698
kmer:13 -L 23 -S 3	1277	500	6285	1272	612	0	0.737224	0.00422341	0.00652912	8089	3485338
kmer:13 -L 23 -S 4	1284	500	6285	1272	612	0	0.734849	0.00413187	0.00610858	6915	3476874
kmer:13 -L 23 -S 5	1287	500	6285	1280	605	0	0.733268	0.00420927	0.00788639	12853	3467826
kmer:13 -L 25 -S 3	1255	500	6285	1221	581	0	0.726381	0.0041663	0.0065081	8046	3413525

								7	2		
kmer:13 -L 25 -S 4	1260	500	6285	1218	578	0	0.725407	0.0041488 3	0.0061608 1	6908	3412288
kmer:13 -L 25 -S 5	1263	500	6285	1222	573	0	0.72326	0.0041685 2	0.0075008 6	11418	3400725
kmer:13 -L 11 -S 1	1275	500	7194	1438	724	0	0.776196	0.0048743 4	0.0094887 2	17165	3684600
kmer:13 -L 11 -S 2	1292	500	7194	1481	744	0	0.778984	0.0048143 8	0.0098714 8	18992	3718443
kmer:13 -L 13 -S 1	1276	500	7194	1438	724	0	0.776237	0.0048247 3	0.0096873 6	18099	3686006
kmer:13 -L 13 -S 2	1295	500	7194	1481	747	0	0.779124	0.0048372 6	0.0098604 1	18881	3721735
kmer:13 -L 15 -S 1	1314	500	9148	1449	722	0	0.770026	0.0045015 8	0.0082867 3	14023	3674042
kmer:13 -L 15 -S 2	1337	500	9148	1488	741	0	0.774379	0.0045074	0.0087756 3	15952	3704578
kmer:13 -L 17 -S 1	1310	500	9463	1417	685	0	0.762468	0.0043010 1	0.0079357 7	13277	3623800
kmer:13 -L 17 -S 2	1327	500	9463	1440	708	0	0.764997	0.0043736 1	0.0082654 6	14315	3647791
kmer:13 -L 19 -S 1	1295	500	7046	1358	649	0	0.750828	0.0042617 3	0.0067003 8	8753	3565223
kmer:13 -L 19 -S 2	1312	500	7046	1383	676	0	0.754754	0.0043606 3	0.0068260 9	8924	3594894
kmer:13 -L 21 -S 1	1279	500	7046	1302	626	0	0.741875	0.0040700 3	0.0057268 8	5845	3507588
kmer:13 -L 21 -S 2	1296	500	7046	1338	648	0	0.746407	0.0042020 5	0.0057891	5650	3539462
kmer:13 -L 23 -S 1	1255	500	6285	1247	592	0	0.732259	0.0040434 9	0.0056667 6	5635	3451717
kmer:13 -L 23 -S 2	1273	500	6285	1257	614	0	0.73603	0.0041977 1	0.0059795	6214	3466652
kmer:13 -L 25 -S 1	1231	500	6285	1177	564	0	0.72049	0.0039882 3	0.0056544	5656	3375428
kmer:13 -L 25 -S 2	1249	500	6285	1210	582	0	0.725835	0.0041465 3	0.0057194 5	5395	3410319

Table 3B – Evaluation of homology-guided assembly on real world data with VELVET (without utilizing left-over reads).

-ins_length 200 -ins_length_sd 20 -scaffolding no

Parameter Settings	Mean	Min	Max	N50	N75	N90	Coverage	Error	Total Error	Unmapped	All
kmer:11 -exp_cov 3	737	602	873	0	0	0	0.000549257	0.00610169	0.00610169	0	1475
kmer:11 -exp_cov 5	592	505	873	0	0	0	0.00467967	0.0168647	0.0949188	1125	13045
kmer:11 -exp_cov 7	635	500	1278	0	0	0	0.0185906	0.0120378	0.0532132	2323	53415
kmer:11 -exp_cov 9	643	500	1278	0	0	0	0.0428968	0.00875858	0.0441525	4697	126847
kmer:11 -exp_cov 11	692	500	2507	0	0	0	0.0789932	0.00814128	0.0325501	6102	241854
kmer:11 -exp_cov 13	705	500	2507	0	0	0	0.108582	0.00770808	0.0336219	8972	334584
kmer:11 -exp_cov 15	720	500	2507	0	0	0	0.132746	0.0070234	0.0293147	9541	415468
kmer:11 -exp_cov 17	726	500	2507	0	0	0	0.144515	0.00684625	0.0322745	12040	458207
kmer:11 -exp_cov 19	732	500	2507	0	0	0	0.15235	0.0070263	0.0318239	12547	489874
kmer:11 -exp_cov 21	736	500	2507	0	0	0	0.155852	0.00715524	0.0313189	12547	502988
kmer:11 -exp_cov 23	737	500	2507	0	0	0	0.157612	0.0071227	0.031018	12547	508796
kmer:13 -exp_cov 3	682	500	1885	0	0	0	0.260833	0.00322693	0.0108927	6451	832369
kmer:13 -exp_cov 5	722	500	2082	0	0	0	0.379329	0.00468802	0.0215565	21653	1255967
kmer:13 -exp_cov 7	836	500	3635	529	0	0	0.523924	0.00564608	0.0261868	40441	1917260
kmer:13 -exp_cov 9	986	500	5647	737	0	0	0.623347	0.00611657	0.0349251	75387	2525435
kmer:13 -exp_cov 11	1138	500	6106	1002	0	0	0.682931	0.00671434	0.0349666	85979	2936849
kmer:13 -exp_cov 13	1254	500	7563	1198	522	0	0.707654	0.00693122	0.0392904	106152	3151537
kmer:13 -exp_cov 15	1321	500	7563	1309	550	0	0.714542	0.0072071	0.0393136	108551	3248046
kmer:13 -exp_cov 17	1339	500	7563	1340	564	0	0.718266	0.00724114	0.0411456	116052	3282079
kmer:13 -exp_cov 19	1355	500	8518	1357	578	0	0.720095	0.0073078	0.0414968	117742	3300954

								9			
kmer:13 -exp_cov 21	1354	500	8518	1359	586	0	0.721978	0.0072478 2	0.041984	119959	3308443
kmer:13 -exp_cov 23	1354	500	8518	1361	589	0	0.72332	0.0072572 3	0.0419514	119959	3312560
kmer:15 -exp_cov 3	855	500	4167	550	0	0	0.537818	0.00311812	0.0072542 8	8430	2023337
kmer:15 -exp_cov 5	877	500	4167	642	0	0	0.598817	0.0037119 5	0.0128878	21424	2304721
kmer:15 -exp_cov 7	1000	500	5424	834	0	0	0.670457	0.0047914 5	0.0158779	31129	2763257
kmer:15 -exp_cov 9	1175	500	9833	1121	559	0	0.720799	0.005485	0.015345	31642	3159891
kmer:15 -exp_cov 11	1311	500	9833	1326	628	0	0.74122	0.0058620 9	0.0155385	33191	3376780
kmer:15 -exp_cov 13	1401	500	9833	1510	669	0	0.750695	0.0059780 9	0.017854	42092	3481043
kmer:15 -exp_cov 15	1441	500	9833	1543	687	0	0.754511	0.0061555	0.0198706	49314	3524163
kmer:15 -exp_cov 17	1457	500	11455	1575	696	0	0.755969	0.0061666 5	0.0200156	50084	3544063
kmer:15 -exp_cov 19	1459	500	11455	1583	711	0	0.758022	0.0063448 9	0.0199259	49321	3559244
kmer:15 -exp_cov 21	1462	500	11455	1595	715	0	0.759152	0.0065099 3	0.0200786	49321	3561941
kmer:15 -exp_cov 23	1463	500	11455	1606	717	0	0.759556	0.0064530 7	0.0200134	49321	3564352
kmer:17 -exp_cov 3	887	500	4117	582	0	0	0.557884	0.0030960 5	0.0036400 1	1158	2121091
kmer:17 -exp_cov 5	905	500	4117	674	0	0	0.616183	0.0037304 6	0.0071816	8350	2402117
kmer:17 -exp_cov 7	1038	500	4683	896	0	0	0.683068	0.0044363 7	0.0112802	19647	2838359
kmer:17 -exp_cov 9	1199	500	8419	1158	567	0	0.722684	0.0049751 9	0.0116263	21517	3197464
kmer:17 -exp_cov 11	1317	500	9194	1363	629	0	0.73997	0.0052200 6	0.0109884	19726	3382146
kmer:17 -exp_cov 13	1392	500	9458	1476	656	0	0.747357	0.0054399 6	0.01116	20143	3482193
kmer:17 -exp_cov 15	1421	500	9458	1514	667	0	0.749665	0.0055467	0.0129999	26561	3517394

								2			
kmer:17 -exp_cov 17	1429	500	9458	1527	674	0	0.752174	0.0055891 7	0.0132373	27373	3531649
kmer:17 -exp_cov 19	1435	500	9458	1543	685	0	0.753917	0.0056466 9	0.0135878	28514	3541895
kmer:17 -exp_cov 21	1437	500	9458	1553	692	0	0.754096	0.0056735	0.0136108	28514	3543493
kmer:17 -exp_cov 23	1436	500	9458	1553	704	0	0.754891	0.0056932	0.0136723	28693	3546861
kmer:19 -exp_cov 3	904	500	4435	597	0	0	0.565362	0.0031103 5	0.0043214 8	2630	2162138
kmer:19 -exp_cov 5	922	500	4435	697	0	0	0.626778	0.0035687	0.0070887 9	8757	2470088
kmer:19 -exp_cov 7	1059	500	6619	928	0	0	0.68909	0.0043223 8	0.0079959 4	10741	2900487
kmer:19 -exp_cov 9	1219	500	8421	1173	575	0	0.722497	0.0048890 6	0.0080581 6	10313	3228021
kmer:19 -exp_cov 11	1328	500	8421	1365	623	0	0.735934	0.0052144	0.0084164 2	10957	3393104
kmer:19 -exp_cov 13	1384	500	8421	1427	654	0	0.742657	0.0053368 9	0.0092588 7	13706	3462314
kmer:19 -exp_cov 15	1406	500	8421	1447	666	0	0.745817	0.0054013 9	0.0094870 5	14414	3494473
kmer:19 -exp_cov 17	1414	500	8421	1470	673	0	0.746845	0.0054339 7	0.0094751 3	14302	3505543
kmer:19 -exp_cov 19	1416	500	8421	1478	681	0	0.746898	0.0055981 7	0.0094445 3	13627	3509364
kmer:19 -exp_cov 21	1420	500	8421	1480	684	0	0.747429	0.0055955	0.0094392 7	13627	3511753
kmer:19 -exp_cov 23	1420	500	8421	1482	687	0	0.748159	0.0056211 2	0.0094622 6	13627	3514070
kmer:21 -exp_cov 3	900	500	4612	602	0	0	0.572872	0.0032238 4	0.0046615 6	3144	2176599
kmer:21 -exp_cov 5	922	500	4612	694	0	0	0.62871	0.0035212 8	0.0064733 5	7315	2461887
kmer:21 -exp_cov 7	1053	500	6618	916	0	0	0.688573	0.0042665 4	0.0099083 5	16533	2901411
kmer:21 -exp_cov 9	1204	500	6761	1145	562	0	0.71965	0.0047255 8	0.0094051 7	15170	3211246
kmer:21 -exp_cov 11	1305	500	6761	1312	605	0	0.730192	0.0049248	0.0093954	15151	3357233

								9	3		
kmer:21 -exp_cov 13	1343	500	6957	1353	615	0	0.734025	0.00515165	0.0105271	18559	3416189
kmer:21 -exp_cov 15	1358	500	6957	1376	625	0	0.736349	0.00520235	0.0105293	18529	3441712
kmer:21 -exp_cov 17	1360	500	6957	1378	628	0	0.737648	0.00531539	0.0113236	20952	3447727
kmer:21 -exp_cov 19	1363	500	6957	1384	630	0	0.737548	0.00534237	0.0113472	20952	3449594
kmer:21 -exp_cov 21	1365	500	6957	1384	630	0	0.737382	0.00535159	0.0113557	20952	3450004
kmer:21 -exp_cov 23	1365	500	6957	1385	630	0	0.737334	0.00534842	0.0115607	21676	3448869
kmer:23 -exp_cov 3	893	500	5419	594	0	0	0.565854	0.00291522	0.0050449	4625	2160732
kmer:23 -exp_cov 5	920	500	5419	694	0	0	0.62378	0.00340794	0.0063883 4	7389	2463362
kmer:23 -exp_cov 7	1057	500	6619	913	0	0	0.682054	0.00408132	0.0092250 8	15028	2894650
kmer:23 -exp_cov 9	1201	500	6619	1120	529	0	0.709706	0.00442108	0.0088618 3	14262	3183158
kmer:23 -exp_cov 11	1269	500	6836	1213	562	0	0.719478	0.00462502	0.0095607 7	16423	3295550
kmer:23 -exp_cov 13	1300	500	6836	1254	574	0	0.722176	0.00482885	0.0096893 4	16423	3346137
kmer:23 -exp_cov 15	1312	500	6836	1273	584	0	0.724761	0.00503737	0.0096675 2	15756	3370013
kmer:23 -exp_cov 17	1315	500	6836	1275	585	0	0.724817	0.00508824	0.0099265 4	16497	3375822
kmer:23 -exp_cov 19	1316	500	6836	1275	584	0	0.724463	0.00505368	0.0104287	18322	3373188
kmer:23 -exp_cov 21	1316	500	6836	1274	585	0	0.725036	0.00508967	0.0116469	22371	3371930
kmer:23 -exp_cov 23	1316	500	6836	1274	583	0	0.72446	0.00509293	0.011652	22371	3370947
kmer:25 -exp_cov 3	897	500	5421	591	0	0	0.565139	0.00295546	0.0038929	2044	2171915
kmer:25 -exp_cov 5	929	500	5421	692	0	0	0.622104	0.00339364	0.0061431 7	6860	2479641

kmer:25 -exp_cov 7	1066	500	5812	913	0	0	0.678177	0.0039584 7	0.0069514 7	8788	2915773
kmer:25 -exp_cov 9	1184	500	5857	1078	510	0	0.702114	0.0042486 6	0.0076775 8	10908	3156758
kmer:25 -exp_cov 11	1241	500	6836	1142	526	0	0.707634	0.0043944 9	0.0076843 9	10790	3254530
kmer:25 -exp_cov 13	1263	500	6836	1164	534	0	0.709234	0.0044599 8	0.0088466 9	14529	3282748
kmer:25 -exp_cov 15	1268	500	6836	1168	538	0	0.710232	0.0046293 1	0.0087183 7	13587	3293796
kmer:25 -exp_cov 17	1269	500	6836	1174	540	0	0.71132	0.0046209	0.0087018 1	13587	3300437
kmer:25 -exp_cov 19	1270	500	6836	1175	543	0	0.711776	0.0047224 5	0.0088008 9	13587	3302101
kmer:25 -exp_cov 21	1269	500	6836	1175	544	0	0.712095	0.0047403 1	0.0088176 4	13587	3302948
kmer:25 -exp_cov 23	1270	500	6836	1177	544	0	0.711984	0.0047530 9	0.0088319 7	13587	3301642

Table 4A – Evaluation of homology-guided assembly on real world data with LOCAS (utilizing left-over reads).

-Lt 15 -St 4 -P kmer 13 -K(Kmerg) 21

Parameter Settings	**N50**
kmer:21 -Llo(Lm) 19 -Slo(Sm) 0 -DR 19 150	1514
kmer:21 -Llo(Lm) 19 -Slo(Sm) 0 -DR 21 150	1514
kmer:21 -Llo(Lm) 19 -Slo(Sm) 0 -DR 17 300	1514
kmer:21 -Llo(Lm) 19 -Slo(Sm) 0 -DR 17 500	1513
kmer:21 -Llo(Lm) 21 -Slo(Sm) 0 -DR 19 150	1514
kmer:21 -Llo(Lm) 21 -Slo(Sm) 0 -DR 21 150	1514
kmer:21 -Llo(Lm) 21 -Slo(Sm) 0 -DR 17 300	1514
kmer:21 -Llo(Lm) 21 -Slo(Sm) 0 -DR 17 500	1513
kmer:21 -Llo(Lm) 23 -Slo(Sm) 0 -DR 19 150	1514
kmer:21 -Llo(Lm) 23 -Slo(Sm) 0 -DR 21 150	1514
kmer:21 -Llo(Lm) 23 -Slo(Sm) 0 -DR 17 300	1514
kmer:21 -Llo(Lm) 23 -Slo(Sm) 0 -DR 17 500	1513

kmer:21 -Llo(Lm) 19 -Slo(Sm) 0 -DR 17 150	1514
kmer:21 -Llo(Lm) 21 -Slo(Sm) 0 -DR 17 150	1514
kmer:21 -Llo(Lm) 23 -Slo(Sm) 0 -DR 17 150	1514

Table 4B – Evaluation of homology-guided assembly on real world data with VELVET (utilizing left-over reads).

-ins_length 200 -ins_length_sd 20 -scaffolding no

Parameter Settings	**N50**
kmer:19 -exp_cov 27	1435
kmer:21 -exp_cov 17	1379
kmer:21 -exp_cov 27	1379
kmer:23 -exp_cov 17	1285
kmer:23 -exp_cov 27	1293
kmer:21 -exp_cov 7	935
kmer:23 -exp_cov 7	935
kmer:19 -exp_cov 17	1427
kmer:19 -exp_cov 27	1435
kmer:19 -exp_cov 7	901
kmer:19 -exp_cov auto	1110
kmer:21 -exp_cov 17	1379
kmer:21 -exp_cov 7	936
kmer:21 -exp_cov auto	1196
kmer:23 -exp_cov auto	1160