PRE- AND POST-PROCESSING

TOOLS FOR NEXT-GENERATION SEQUENCING

DE NOVO ASSEMBLIES

by

Sari S. Khaleel

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Science in Bioinformatics and Computational Biology

Spring 2012

PRE- AND POST-PROCESSING

TOOLS FOR NEXT-GENERATION SEQUENCING

***DE NOVO* ASSEMBLIES**

by

Sari S. Khaleel

Approved: _____
Cathy H. Wu, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: _____
Errol Lloyd, Ph.D.
Chair of the Department of Computer and Information Sciences

Approved: _____
Babatunde A. Ogunnaike, Ph.D.
Interim Dean of the College of Engineering

Approved: _____
Charles G. Riordan, Ph.D.
Vice Provost for Graduate and Professional Education

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Cathy Wu; and my committee members Dr. Chuming Chen, Dr. Shawn Polson, and Dr. Eric Wommack for the guidance, encouragement, and patience that they have provided me with over the past two years. I would also like to thank my friends, Phil Perry and Michael Turchiano, and my colleagues, Dan Nasko and Xia Bi who have supported me and enriched my life.

This dissertation was made possible by funding from the National Science Foundation grant # MCB-0731916 (to K.E. Wommack and S.C. Cary), the Gordon and Betty Moore Foundation's Marine Microbiology Initiative to (K.E. Wommack and S.W. Polson); and the U.S Department of Energy grant # DE-FOA-0000368 (to C.H. Wu).

I would like to acknowledge Craig Cary of the University of Waikato, New Zealand for his cooperation with my metagenome analysis projects, and Miguel Pignatelli from the Wellcome Trust Genome Campus, EBI for his assistance with the development of my tools and expanding my knowledge of bioinformatics programming.

This dissertation is dedicated to my parents, Safaa and Wiaam Khaleel for their love, support, and guidance throughout my life, and to my brother, Waseem Khaleel, for being my brother and best friend.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

High-throughput Next-Generation Sequencing (NGS) technologies have revolutionized and accelerated genomic analyses. However, their shorter read length and higher error rates in comparison to classical Sanger sequencing have hindered downstream analyses such as *de novo* genome assembly. Therefore, there is an urgent need to develop tools for quality control and pre-processing of NGS short-read data and perform systematical assessments of their impact on downstream analyses

Many *de novo* assembly projects that use NGS include a pre-processing step where low quality reads and sequence artifacts are cleaned from NGS reads using unpublished in-house scripts. Although some open-source and commercial trimming scripts or pre-processing tools are available, a simple and comprehensive open-source toolkit with major trimming algorithms is currently lacking. Furthermore, most of the aforementioned assembly projects assess assembly by contiguity alone without evaluating the correctness of the assembled contigs.

The problem of misassembly is complicated further in metagenomic assemblies by contig chimerism, which is caused by the co-assembly of reads from two or more genomes at regions of sequence similarity. Recently developed metagenomic assemblers attempts to solve this problem during assembly, but they are trained on short, high-coverage Illumina reads and not the long, low-coverage reads of 454, the preferred platform for metagenome sequencing. Contig chimerism affects downstream metagenome analyses, such as binning and gene prediction.

Presented are methods and tools for pre- and post-assembly processing of NGS data. ngsShoRT (next-generation-sequencing Short Reads Trimmer), a tool written in Perl to implement many of the commonly used algorithms in trimming literature as well as methods developed by our group, was developed for pre-processing of NGS reads. ngsShORT was tested on Illumina paired-end (PE) reads of the *Caenorhabditis elegans genome*, and its trimming methods were compared by the improvement in assembly contiguity as well as accuracy with BLAST. A particular problem in trimming NGS reads is the identification of adaptor sequences in NGS reads that may not be detected by regular text-search algorithms. This problem can be managed by the identification and removal of high frequency K-mers in reads using kmerFreq, our adaptor sequence detection and trimming tool. kmerFreq was tested on 454 reads of a viral metagenome specimen and the resulting improvement in assembly contiguity and assembler performance was evaluated.

Finally, an analysis pipeline for contig chimerism in metagenomic assemblies of simulated 454 reads is presented and tested on a simulated bacterial metagenome. Our hypothesis is that chimeric contigs can be identified in a metagenomic assembly by the presence of unique coverage and polymorphism attributes that distinguish them from non-chimeric contigs. To train this post-assembly approach, a chimeric contig simulation and analysis pipeline was developed to study contig chimerism in assemblies of simulated metagenomes. The pipeline was used to simulate a bacterial metagenome and analyze and compare chimeric and non-chimeric contigs in its assembly. The results of this analysis may provide insight into coverage and polymorphism patterns in chimeric contigs, and may be useful for the detection of chimeric contigs in a real metagenome assembly.

Chapter 1

INTRODUCTION

The emergence of Next-Generation Sequencing (NGS) technologies as a cost-effective, high throughput alternative to classical Sanger technology has greatly benefitted biological research. Popular commercially available NGS platforms include Illumina (Genome Analyzer I, II and HiSeq), Roche (454 GS FLX Pyrosequencer) and ABI (SOLiD). Currently, the two major NGS platforms are Illumina and 454 [52, 60].

Unfortunately, all NGS platforms have major drawbacks in comparison to classical Sanger sequencing, including shorter read lengths, higher base-call error rates, and novel platform-specific artifacts. These sequencing errors result in poor quality assemblies that have shorter lengths and higher error rates when aligned to reference sequences [7]. In addition, the high throughput and short length of NGS reads (especially Illumina and ABI-SOLiD) make classical Overlap-Layout-Consensus (OLC) assembly algorithms obsolete. Instead, short high-throughput NGS reads are usually assembled using the de Bruijn Graph (DBG) approach, in which reads are decomposed into K-mers that in turn become the nodes of the DBG [37, 45]. The DBG approach has several drawbacks that include sensitivity to sequencing errors, as miscalled bases produce erroneous K-mers that result in increasing the DBG complexity, which can lead to longer runtime, more memory overhead, and more importantly, poor assemblies [7].

Consequently, many assembly projects include a pre-processing step where NGS data is “quality-trimmed” using a combination of trimming methods. Although

some open-source and commercial trimming scripts or pre-processing tools are available, a simple and comprehensive open-source toolkit that includes all the major trimming and contaminant detection methods is currently lacking.

Chapter 2 covers the problem of pre-processing NGS reads and their subsequent assembly using De Bruijn Graph assemblers. General problems of the Illumina and 454 platforms are reviewed first, followed by a discussion of the DBG assembly algorithm and its popular assemblers and a review of trimming algorithms used in literature as well as commonly used trimming packages/scripts for pre-processing NGS reads. Chapter 2 ends with presenting *ngsShoRT*, an NGS reads trimming tool that implements most of the commonly used sequence-trimming methods in NGS literature as well as methods created by our group. *ngsShORT* methods are applied to Illumina GA II paired-end (PE) reads of the *Caenorhabditis elegans* genome, which are subsequently assembled using several popular DBG assemblers. Individual trimming methods of *ngsShoRT* are compared by measuring their resulting improvements on assembly contiguity and correctness with BLAST, and then the combination of trimming methods that produces the greatest improvement in both measures is determined.

Chapter 3 covers the problem of detection and removal of sequence artifacts, a major step in NGS trimming. Most trimming tools search for a user-supplied list of known platform sequence artifacts in reads using direct text-search. These methods often do not allow for mismatches and cannot adapt to the possibility of primer/adaptor sequence fragmentation. The alternative approach proposed in this chapter for primer/adaptor trimming from NGS reads is done in two steps: The first step is the detection overrepresented K-mers in the 5'- and 3'-ends of reads sampled

from the dataset using an approximate hash table (a hash that uses approximate matching for key lookups), as these K-mers may represent adaptor/primer sequences; the second step is to trim out these over-represented sequences from the dataset reads.

This approach is implemented in *kmerFreq*, a tool developed in Perl (with multiprocessing in order to manage high throughput data) as a pre-processing step for NGS reads. *kmerFreq* was tested on 454 reads of a viral metagenome specimen. To evaluate *kmerFreq*'s detection function, its detected sequenced were aligned against NCBI's nr/nt database using BLASTN. To evaluate the impact of removing these sequences, the subsequent improvement in *de novo* assembly of these reads was measured for assemblies done using the OLC assemblers Phrap [15] and Newbler.

The problem of misassembly is more complicated in metagenomic assemblies due to the lack of a reference sequence to use for evaluating assemblies [28]. This complicates downstream analyses, such as binning and gene prediction. In addition, metagenome assembly suffers from contig chimerism, reads from different taxonomic groups co-assembling into the same contig [28, 42, 50].

Recently developed metagenome-specific assemblers are based upon DBG assembly and are trained only on high-coverage short Illumina reads, and are not reliable for assembly of low-coverage reads from the 454 platform, the more popular platform for metagenome assembly. Therefore, Newbler, 454's native OLC assembler, remains the main assembler for 454 reads although it was not designed for metagenome assembly.

Chapter 4 discusses the problem of contig chimerism. Our hypothesis is that base coverage and polymorphism information can be used to differentiate chimeric from non-chimeric contigs. To test this hypothesis, a chimeric contig analysis pipeline

that generates simulated NGS reads of artificial metagenomes and analyzes the coverage and polymorphism patterns of non-chimeric and chimeric contigs assembled from these reads is presented. The goal of the pipeline to find a set of coverage and polymorphism features that can be used to identify potentially chimeric contigs in assemblies of real metagenomes with unknown source species. Chapter 4 concludes by presenting the findings from running our analysis pipeline on a simulated bacterial metagenome and discussing differences between chimeric and non-chimeric contigs and examine the chimeric regions of these contigs at which reads from different species co-assemble. Finally, the conclusions of this dissertation are summarized in chapter 5.

Chapter 2

PRE-PROCESSING OF NEXT GENERATION SEQUENCING READS AND DE-NOVO GENOME ASSEMBLY

The disadvantages of NGS include higher error rates and shorter read lengths in comparison to classical Sanger sequencing. In addition, the high throughput of these short reads has led to using the De Bruijn Graph (DBG) approach for their assembly using recently developed and not well-optimized algorithms. The higher error rate of NGS data and the limitations of DBG assembly algorithms have led to emphasis on pre-processing NGS reads using some trimming methods that usually include the removal of reads with uncalled 'N' bases and adaptor sequences as well as some form of quality filtering. However, most assembly projects perform this pre-processing step using unpublished in-house scripts, and currently popular and freely available trimming tools do not implement all popular trimming algorithms and lack essential features for managing NGS reads and their platform-specific errors.

This chapter begins by discussing NGS platforms, specifically second-generation, cyclic array sequencing platforms, and focuses on the most common of these platforms, Illumina and 454, discussing their error profiles. Next, the De Bruijn Graph (DBG) assembly method and its drawbacks are presented. The chapter concludes by presenting ngsShoRT, a powerful tool written in Perl to manage Single or Paired-end NGS data in the popular FastQ file format as well as Illumina's native qseq format, with special trimming methods for Illumina reads. ngsShoRT allows users to trim and filter reads using varying combinations of popular trimming methods

in literature as well as methods developed by our group. ngsShoRT is tested on Illumina GA II datasets and the resulting improvements in assembly contiguity, correctness, and assembler performance are evaluated in order to determine the best combination of ngsShoRT methods and parameters for NGS read trimming.

2.1 Next-Generation Sequencing Technology

Until recently, the overwhelming majority of DNA sequence production has relied on some version of the Sanger biochemistry [32, 52]. The need for a cheaper, faster and more flexible alternative to Sanger sequencing has driven the development of new, next-generation technologies [52]. The cheap cost, high throughput and high coverage offered by NGS make the technology available to researchers in many old and newly emerging fields of life sciences, such as genomics, transcriptome studies, metagenomics, etc. In addition to its lower cost, major advantages of NGS over Sanger sequencing include parallelization (over 100 reactions versus 96 or 384-channel capillary systems), and the removal of cloning, the main bottleneck of Sanger Sequencing. Cloning is not fully automated, is expensive, and since the target vector is bacterial chromosomes, cloning is sometimes biased against certain DNA sequences that are not easily clonable. In contrast, NGS uses PCR to generated amplicons from fixed DNA segments [52].

Shendure *et al.* [52] classified alternative strategies for DNA sequencing into several categories: (i) microelectrophoretic methods, (ii) sequencing by hybridization, (iii) real-time observation of single molecules, and (iv) cyclic-array sequencing. Cyclic-array sequencing is currently the most common of these technologies and most of the major, commercially available NGS platforms are implementations of cyclic array. Cyclic array is also known as “second generation sequencing” or sequencing-

by-synthesis, that is, serial extension of primed templates [32, 52]. The most common and commercialized implementations of NGS include 454 Genome Sequencers (Roche Applied Science; Basel), Illumina (Genome Analyzer I, II, and HiSeq). IIllumina; Sang Diego), the SOLiD platform (Applied Biosystems; Foster City, CA, USA), and the HeliScope Single Molecule Sequencer technology (Helicos; Cambridge, MA, USA) [52]. In this dissertation, the term "NGS" will be used to refer specifically to sequencing-by-synthesis platforms.

2.2 Error Profiles of Illumina and 454 Platforms

Currently, the two major NGS platforms are Illumina and 454. Illumina offers high throughput, short (~100 bp with Illumina GA II, ~150 bp with HiSeq) reads at relatively low cost, making it the more popular platform for genome re-sequencing (sequencing a genome with a known reference sequence to detect small mutations, SNPs, etc), *de novo* genome and transcriptome assembly, seq-based studies (RNA-seq, ChIP-seq), and quantitative analysis based on the number of sequence segments [37, 52]. 454, on the other hand, offers lower throughput, long (~400 bp with GS FLX Titanium) reads, making it the popular platform for metagenomics and other fields where long read length is essential [52, 60].

Several studies have been done to analyze the error profiles of NGS platforms [18, 20, 37, 55], and their evaluations of the Illumina and 454 error profiles can be summarized under the following metrics:

1. Base-call quality and alignability of reads to reference: On average, 55% of Illumina GA reads passed quality filters, of which approximately 77% aligned to the reference sequence. In contrast, approximately 95% of 454 reads uniquely aligned to the target sequence [18].

2. Coverage variation: An often-described property of Illumina and 454 profiles is coverage variation [18, 20, 37, 55], which has been attributed to the inherent bias of polymerase chain reaction (PCR) amplification during sample preparation [18], or formation of secondary structures in single-stranded DNA (ssDNA) [56]. AT-rich repetitive sequences showed lower coverage [18], while GC-rich region had higher error rates [37]. Harismendy *et al.* [18], noticed an over-representation of amplicon end sequences: These regions, which represented only 2.3% of their targeted sequence, accounted for up to 56% of sequenced base pairs from Illumina GA technology in contrast to only 5% of 454 bases. This difference in platform performance was attributed to library preparation process.

   For 454, the emulsion PCR (ePCR) reaction seems to have a tendency for generation of duplicate reads (multiple reads from a single template), which occur when amplified DNA attaches to empty beads during ePCR, or when the optical signal during sequencing 'bleeds' into the space of an adjacent empty well [14].

3. Error rates: Illumina sequencers result in more substitution-type miscalls than indel-type miscalls, while 454 sequencers result in more indel-type miscalls than substitution-type miscalls [22]. The high frequency of indel miscalls are caused by the difficulty of correlating the intensity of light produced by the ePCR when the polymerase runs through a homopolymer with the actual number of nucleotide positions.

2.3 *De novo* Genome Assembly Using the De Bruijn Graph Approach

The long read length (~1000 bp) and low throughput of ABI-Sanger reads allowed them to be assembled using the classical Overlap-Layout-Consensus (OLC) approach. In the Overlap phase, overlap discovery involves all-against-all, pair-wise read comparison. Overlap candidates must share K-mers that are used as alignment seeds. Next, an approximate read layout is constructed from manipulation of the overlap graph. In the consensus phase, progressive pair-wise alignments are used compute an optimal multiple sequence alignment [11, 35]. Newbler, the native assembler of the 454 platform, implements the OLC algorithm [35].

Unlike ABI-Sanger and 454, NGS technologies like Illumina and ABI-SOLiD generate far shorter reads with far higher coverages (30 to even 100X coverages for Illumina and SOLiD in comparison to the 8X coverage typical of Sanger sequencing projects). Furthermore, the shorter read length of NGS reads requires many more reads to generate the same level of coverage as that of Sanger. This results in very large datasets of short read sequences that not only make the 'read-centric' OLC method computationally unfeasible, but it also makes it seemingly impossible to find heuristics to resolve the large number of overlaps between short reads [11]. However, pioneering work by Pevzner and colleagues in the late 1980s and Idury and Waterman in the mid-1990s introduced a different framework for handling assemblies. This framework utilizes the de Bruijn Graph (DBG) data structure instead of the overlap graph [11, 35, 43]. The DBG consists of very small, fixed-length subsequences known as K-mers (K is usually 19 or higher), and was originally developed for combinatorial mathematics [11].

In the DBG approach, the reads are decomposed into K-mers that in turn become the nodes of a DBG. This allows compressing the massive read dataset by

converting it into a K-mer frequency table where, regardless of the number of a K-mer's occurrences in the read dataset, it is represented only once in the table. A directed edge between the DBG nodes indicates that the K-mers on those nodes occur consecutively in one or more reads. Given perfect data, i.e., error-free K-mers providing full coverage and spanning every repeat, the K-mer graph would be a de Bruijn graph and it would contain an Eulerian path that traverses each edge exactly once, which is equivalent to the genome sequence, making assembly a by-product of the graph construction [11, 35]. Figure 2.1 shows the basic approach for constructing Overlap and de Bruijn graphs from the same set of reads.

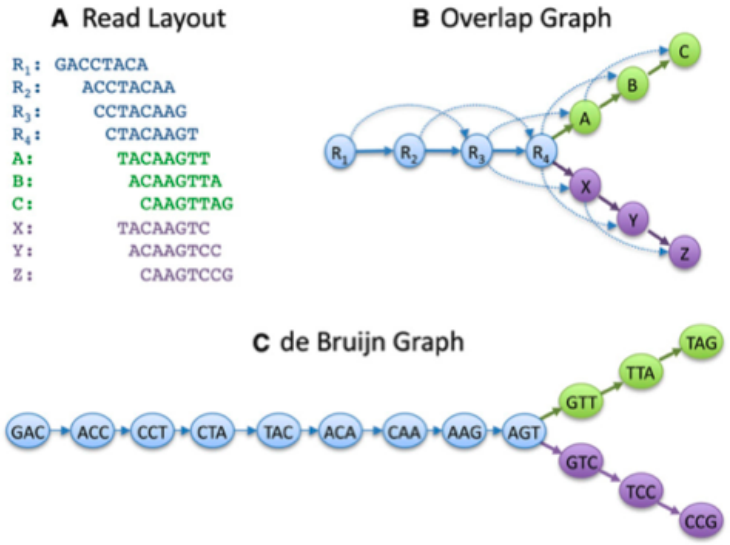


Figure 2.1 Differences between Overlap and De Bruijn Graph Construction. Based on the set of 10 8-bp reads (A), we can build an overlap graph (B) in which each read is a node, and overlaps >5 bp are indicated by directed edges. Transitive overlaps, which are implied by other longer overlaps, are shown as dotted edges. In a de Bruin graph (C), a node is created for every K-mer in all the reads; here the K-mer size is 3. Edges are drawn between every pair of successive K-mers in a read, where the K-mers overlap by k -1 bases. In both approaches, repeat sequences create a fork in the graph. Note here we have only considered the forward orientation of each sequence to simplify the figure. *Adapted from [35]*.

Sequencing errors complicate K-mer graphs because a single miscall will result in creating a new K-mer sequence that adds an additional branch within the de Bruijn Graph. However, many errors are easily recognized by their structure in the graph. For example, errors at the end of a read usually create K-mers that occur only once, and therefore form dead-end ‘‘tips’’ in the graph. Errors in the middle of a read create alternate paths called ‘‘bubbles’’ that terminate at the same node (Figure 2.2). DBG assemblers search for these localized graph structures in an error correction phase and remove the error nodes and other low coverage nodes [11]. Mate-pair information can be used to resolve ambiguity, using the coverage at each node to identify repeats, and by searching for unique paths through the graph consistent with the mate pairs [11, 35 63]. Popular freely available DBG assemblers include Velvet [63], ALLPATHS [4], ABySS [54], and SOAPdenovo [31].

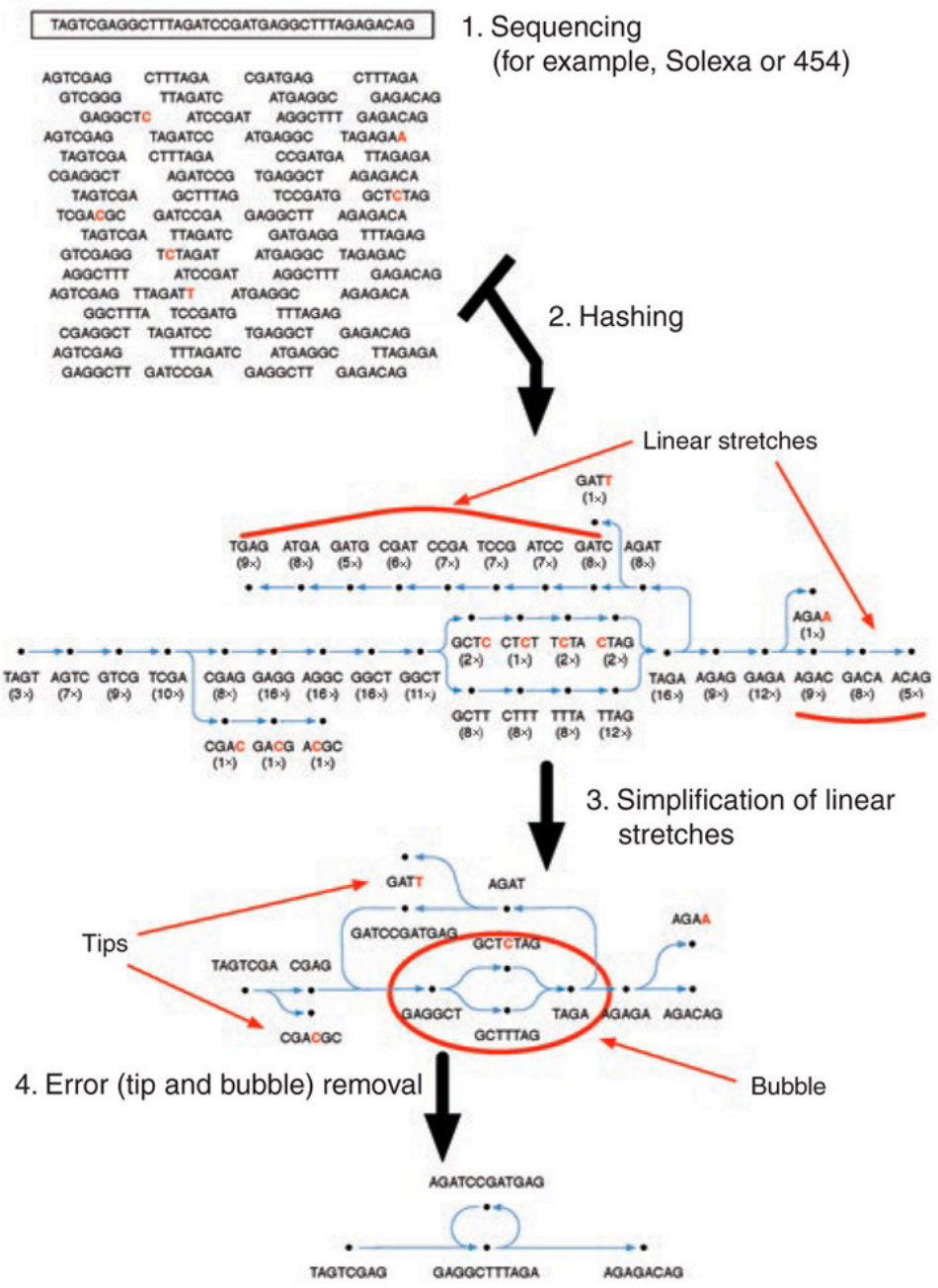

TAGTCGAGGCTTTAGATCCGATGAGGCTTTAGAGACAG
1. Sequencing
(for example, Solexa or 454)
2. Hashing
Linear stretches
3. Simplification of linear stretches
Tips
Bubble
4. Error (tip and bubble) removal
AGATCCGATGAG
TAGTCGAG
GAGGCTTTAGA
AGAGACAG

Figure 2.2 Construction and Resolution of the DBG graph of a DNA sequence. The sequence at the top represents the polymorphism-free genome sequence, which is then sampled using shotgun sequencing with 7-bp reads (step 1). Some of the reads have errors (red). In step 2, the K-mers in the reads (4-mers in this example) are collected into nodes and the coverage at each node is recorded. There are continuous linear stretches within the graph, and the sequencing errors create distinctive, low-coverage features through out the graph. In step 3, the graph is simplified to combine nodes that are associated with the continuous linear stretches into single, larger nodes of various k-mer sizes. In step 4, error correction removes the tips and bubbles that result from sequencing errors and creates a final graph structure that accurately and completely describes in the original genome sequence. *From [11]*.

By breaking reads into shorter K-mers and processing these K-mers instead of reads, the K-mer graph construction discards long-range continuity information in the reads that are longer than their constitutive K-mers [35]. Repeats longer than K lead to tangled K-mer graphs that complicate the assembly problem, and complete read information can be essential for resolving long repeat or ambiguous regions that cannot be resolved using the K-mer graph or mate pair information [11, 35]. Furthermore, unlike the overlap graph, the DBG is not read coherent, i.e., there may be paths (resulting from K-mer alignment) through the graph that form a sequence that is not supported by the underlying reads. For example, if the same K-mer occurs in the middle of two reads, but the reads do not otherwise overlap, the corresponding DBG for those reads contains a branching node instead of two separate paths [11]. Another potential drawback of the DBG approach is that graph construction can require an enormous amount of computer resources [35]. Unlike conventional overlap computations, which can be partitioned into multiple jobs with distinct batches of reads (overlaps can be discovered in parallel with a matrix partition), the construction

and analysis of a DBG is not easily parallelized [35]. As a result, DBG assemblers such as Velvet [63] and ALLPATHS [4], which have been used successfully on bacterial genomes, would require several terabytes of RAM for assembly of the human genome, which is far more memory than is available on most computers. Currently, ABySS [54], SOAP [31] and CLCBio [5] are the only DBG assemblers known to be capable of assembling a mammalian genome [35]. All three assemblers use some form of parallelization to divide the memory load, as well as early correction of the DBG to remove erroneous K-mers that would otherwise add more complexity to the graph, thus increasing memory usage [5, 11, 31, 35, 54]. Finally, current DBG assemblers do not include base-call quality in graph construction, and do not produce quality score information for assembled contig bases.

For these reasons, many sequencing projects (especially metagenome projects where read length matters for better homology detection [60]) prefer the relatively longer 454 reads (~400 bp with Titanium FLX) over Illumina reads (100 bp with GA II, ~150 bp with HiSeq). Instead of DBG assembly, 454 reads are assembled using Newbler which implements OLC instead of DBG, uses read quality score and flow information in OLC construction, and produces quality scores for assembled contig bases, along with complete read-to-contig alignment information [35].

2.4 Review of Current Tools for Pre-processing NGS Reads

Due to the problems of NGS, assembly projects that use NGS data include a pre-processing step where NGS data is "quality-trimmed." This section reviews the pre-processing steps used by several recent assembly projects [1, 8, 9, 22, 34, 40, 54, 61, 64] that used Illumina and 454 data, as well as popular NGS-trimming software [5, 7, 10, 21, 16, 26, 30, 56, 41, 49]. Trimming steps that were used by all assembly

projects and provided by most trimming software include removing reads with uncalled 'N' bases, and the detection and removal of adaptor sequences using direct text-search algorithms to search for user-specified adaptor sequences (usually the platform's known PCR primers, linkers and adaptors) in the reads set. To our knowledge, only CLC's commercial genomics workbench allows for the modification of the text-search algorithms used for adaptor sequence detection.

An additional step done by a few assembly projects [1, 17] is to trim a number of bases from the 3'-end of all reads because of low base-call quality at the 3'-ends of reads [7, 20, 37]. Multiple trimmed datasets are generated by trimming different arbitrary numbers of bases from the 3'-end of reads, and are assembled at different K-mer lengths with a DBG assembler like Velvet. The chosen dataset (and thus the number of 3' bases that was removed from all of its reads) is the one with highest contiguity assembly. This method is inefficient because it relies on selecting an arbitrary set of numbers, treats all reads as if they all have the same quality scores at their 3'-ends, and requires assembly of multiple datasets using different arbitrary K-mer lengths, which is computationally expensive. Finally, the correctness of these trimmed datasets is not evaluated by comparison to a reference sequence.

A less crude approach involves a sliding window algorithm that tries to extract a substring of read bases whose first and last base quality scores exceed a specified cutoff. This "dynamic trimming" step [7, 16, 26] is still arbitrary when it comes to determining the quality score cutoff and sliding window length, and can result in high-quality reads that are still skipped during DBG assembly because they were trimmed to lengths shorter than the K-value (K-mer length) used by the DBG assembler, which requires all reads to be at least longer than K bases [63].

An additional complication of trimming is the special case of mate pairs. As we mentioned earlier, mate pairs are important for repeat resolution and scaffolding, especially for DBG assemblers where long reads are broken down into short K-mers that are usually shorter than repeat regions. Most NGS platforms, especially Illumina, produce PE or mate pair reads. Mate pairs are usually listed in two separate files: the forward direction reads are in a file suffixed with _1, and their reverse direction mate reads are in a file suffixed with _2 *in the same order* of reads in the _1 file. During PE assembly, assemblers take one read from each mate file and use them only if the header of the read from _1 matches the read from _2. Moreover, a pair will still be skipped if either or both of its reads are shorter than the K-mer length used by the DBG assembler.

Given the importance of mate pairs and the important ordering and length conditions for using their information, it was surprising to find that most of the available trimming tools do not trim mate pair files as a unit. Instead, they trim each mate file separately, which can result in removing some reads from the _1 file, while their mates in the _2 file are not removed (and vice versa), thus ruining the mate pair ordering for the two files. To our knowledge, the only freely available platforms that manage mate pairs are Btrim [26] (in a secondary step following separate mate file trimming), and NGS QC [41]. Except for NGS QC, none of the available toolkits implement multiprocessing to manage large NGS read datasets, especially the short reads of the high coverage Illumina and ABI-SOLiD platforms.

Despite the popularity of the Illumina platform, none of the software tools that we reviewed offer Illumina-platform specific trimming methods or manage the direct qseq-format output of Illumina, which included “Failed_Chastity” filter information

that is very useful for trimming and can greatly improve assembly [22] but is lost during conversion to FastQ or FastA + qual files prior to trimming. Another unique feature of Illumina qseq format that is usually lost during quality score conversion from Illumina's ASCII-to-Phred mapping to the standard Sanger mapping is Illumina's 'B' quality score, which in Illumina jargon means 'unknown quality score.' None of the platforms reviewed managed this case or try to trim 'B'-scored bases, and only one assembly project [12] managed it using unpublished in-house scripts that process raw qseq files.

Finally, we could not find a toolkit that included all popular trimming methods and was used by many assembly projects. Instead, most of the assembly projects that were reviewed [1, 8, 9, 22, 34, 40, 54, 61, 64] do *not* use these open-source toolkits and instead perform trimming using their own unpublished in-house scripts, with arbitrary parameters that are tailored to their NGS data.

2.5 ngsShoRT

ngsShoRT (NGS Short Read Trimmer) is a tool written in Perl 5.6 to trim Single or Paired-end reads in the popular FastQ read format or Illumina's native qseq format using all of the popular trimming methods used in NGS assembly projects and tools, as well as methods developed by our group. ngsShoRT uses multiprocessing to manage high throughput data and reduce running time. Another unique feature of ngsShoRT is that it was designed to manage Paired-End (PE) reads using PE-specific modules. ngsShoRT can be easily incorporated as a pre-processing step for most NGS transcriptome and genome assembly pipelines to improve the contiguity and correctness of assembly as well as reducing the memory usage for the assembly process.

The following methods are implemented by ngsShoRT:

(1) 5adpt[mp,list,approx_match_modifiers,search_depth,action], which detects (at a match percentage *mp* and up to a depth of *search_depth*) 5'-adaptor/primer sequences loaded from *list* (which is defaulted to Illumina library primers and adaptor sequences) and removes them from reads. *5adpt* allows users to implement approximate matching using the Levenshtein edit distance implementation in CPAN's String::Approx module [19]. This module allows approximate matching using a simple percentage cutoff, or using detailed modifiers: the number of allowed insertions, substitutions, and deletions. This feature, accessible through the *approx_match_modifiers* option, allows *5adpt* to be modified to adapt to specific platform features and error profiles. For example, one should expect more indels over substitutions for 454 reads, and expect the opposite for Illumina reads.

After an adapter/primer/linker sequence (or fragment) is matched and trimmed out of the read, *action* allows users to specify what to do with the read: it can be removed completely (*action*=kill-read, kr) or trimmed to the base 5' to the detected artifact string (*action*=kill-after, ka).

(2) *nperc[p]* and (3) *ncutoff[n]*, which filter out reads with uncalled 'N' bases whose percentage or number are >= *p* or *n*, respectively. *ncutoff[n =1]* can be used to replicate the commonly used pre-processing step of trimming out reads with any 'N' bases.

(4) *nsplit[l],* which detects strings of uncalled 'N' bases whose length is >= *l*, removes them from the read, and then splits the read around the detected 'N'-string into two

smaller daughter reads. This method was developed by our group to remove N-bases from reads without having to remove the entire read and lose its information.

nperc, *ncutoff* and *nsplit* are important for removing 'N' bases because they usually have low quality scores, and DBG assemblers discard reads with such bases [54], or simply convert them to an arbitrarily-chosen nucleotide like 'A' [63].

(5) *3end[x]* and (6) *5end[y],* which trim x and y bases from the 3' and 5' ends of all reads.

(7) *TERA[avg],* is a method that I developed as an alternative to *3end*. Unlike *3end*, *TERA* trims the 3'-end of each read differently depending on it base-call quality scores. Starting at the 3'-end, the running average quality score (RAQS) for each base is calculated until it exceeds a cutoff, *avg*, at a base X; all bases 3' to X are then discarded. A good read with high quality (above *avg*) bases at its 3'-end will not be trimmed by *TERA*, while a low quality read may be trimmed more than other reads.

(8) *Mott[ml]*, is a quality-window extraction algorithm (i.e., it can trim both the 5' and 3' ends of a read). Starting at the 3'-end of a read, it counts the running sum of (ml - P_{error}) values, RSMLP, for each base in the read (P_{error} of a base = 10-Q/10), it extracts the string from the first base with RSMLP > 0 to the base with the highest RSMLP. *Mott* method was adapted from the CLC Bio Genomics Workbench [5]. *Mott* is also similar to the "QRL" algorithm used by [8].

(9) *LQR[lqs, p],* which reads with over *p*% of bases whose quality score is under *lqs*. It is similar to algorithms used in [5, 16, 22].

(10) *qseq0* and (11) *qseqB[n,mode,action]* are methods specifically designed for Illumina reads. *qseq0* removes qseq reads that did not pass the "Failed_Chastity" filter, which was shown to greatly improve assembly contiguity and correctness (Illumina technote, 2010). *qseqB* trims out reads with more 'B'-scored bases than *n*. Illumina's 'B' score means 'unknown quality score' and bases with such scores are usually trimmed out along with bases 3' to them [8, 12]. Unlike *qseq0, qseqB* can be used with Illumina reads in the FastQ format after switching their ASCII-to-Phred score mapping from Sanger back to Illumina using *switch_scores* (see below).

At *mode="local," qseqB[n]* will remove reads with >= n 'B'-scored bases. At *mode="local," qseqB[n]* will search for a string of consecutive 'B'-scored bases no shorter than n. If such a string is detected, the read can be removed completely (*action*=kill-read, kr) or trimmed to the base 5' to the detected 'B'-scored string (*action*=kill-after, ka). A limited implementation of *qseqB* with *mode*=local and *action*=ka is used by [12].

(12) *switch_scoring* is not a trimming feature, but it allows switching the ASCII-to-Phred mapping of base-call quality scores between Illumina and Sanger mapping, which restores original Illumina-scoring for FastQ-formatted Illumina reads downloaded from NCBI's short read archive, including the aforementioned 'B'-scoring of bases that can then be trimmed using *qseqB*.

DBG assemblers will skip a PE read pair in paired-end assembly mode if either of its reads is shorter than the K-mer length used for assembly. To avoid trimming reads to lengths smaller than the K-mer size used for assembly, ngsShoRT enforces a global minimum read length limit variable, *min_rl*, on *TERA, 3end, 5end*, and *Mott*

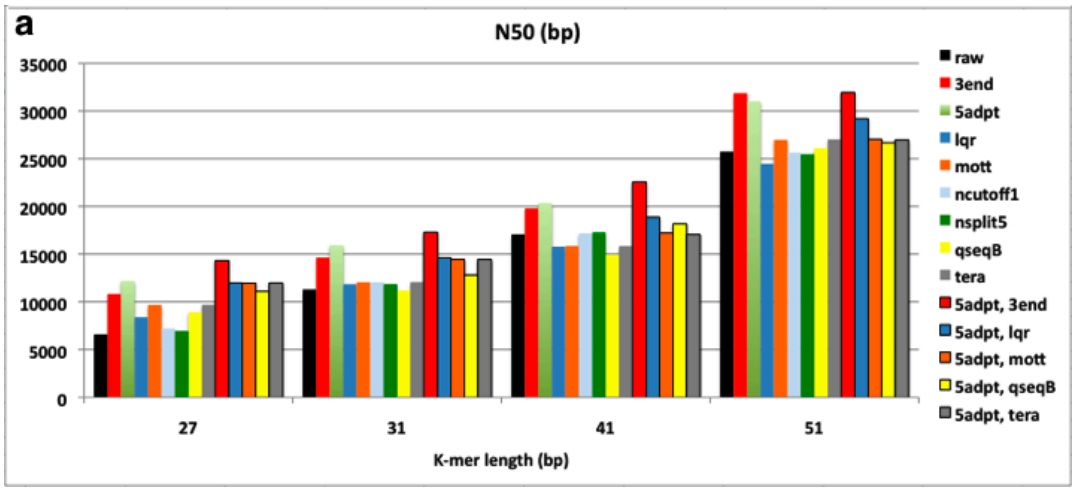
methods to stop trimming when a read's length = *min_rl*. So, if the highest K-mer length used for assembly was 41, we set *min_rl* to 42 bp.

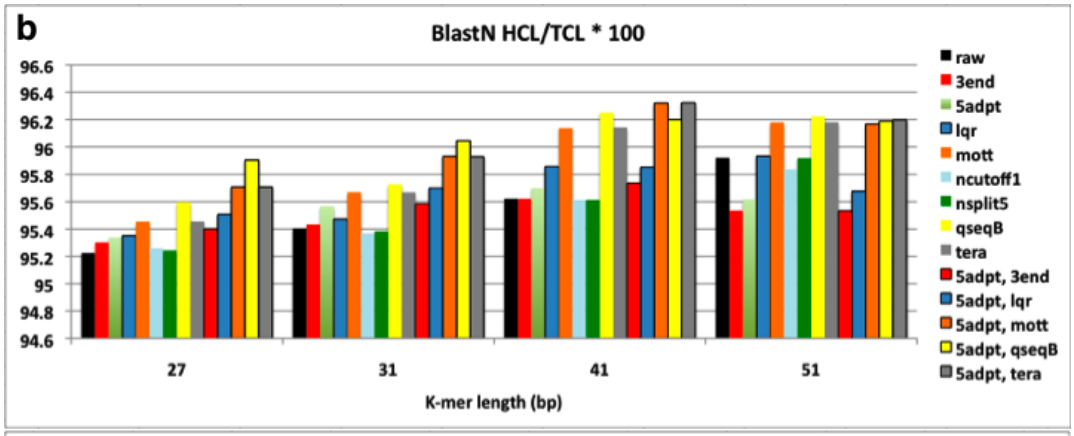
Another special case of PE read trimming is "widowed" mates: if a read pair had one trimmed-out read and another intact read, ngsShoRT saves this "widowed" read in a separate single reads file that is co-assembled with the PE reads file. This approach was suggested by Daniel Zerbino (personal communication, 2011) and is used by [8] and [41].

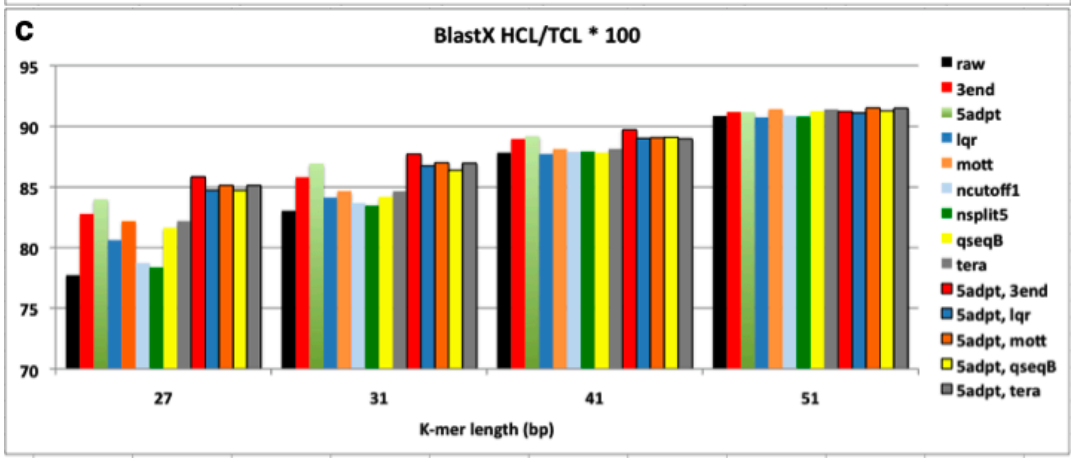
ngsShoRT was designed in object-oriented Perl where the main object is a READ object. Every time ngsShoRT parses the next (qseq, FastQ, FastA) read from its input reads file, a READ object is created for every read to hold its attributes (most importantly, the header, the sequence, and the quality score strings of the source read, and the "Failed_Chastity" filter of qseq-format reads). Trimming methods act on the READ object's components and are independent of the reads file's original format, which makes it easy to implement additional trimming methods or have different output formats in ngsShoRT since the design is format-independent.

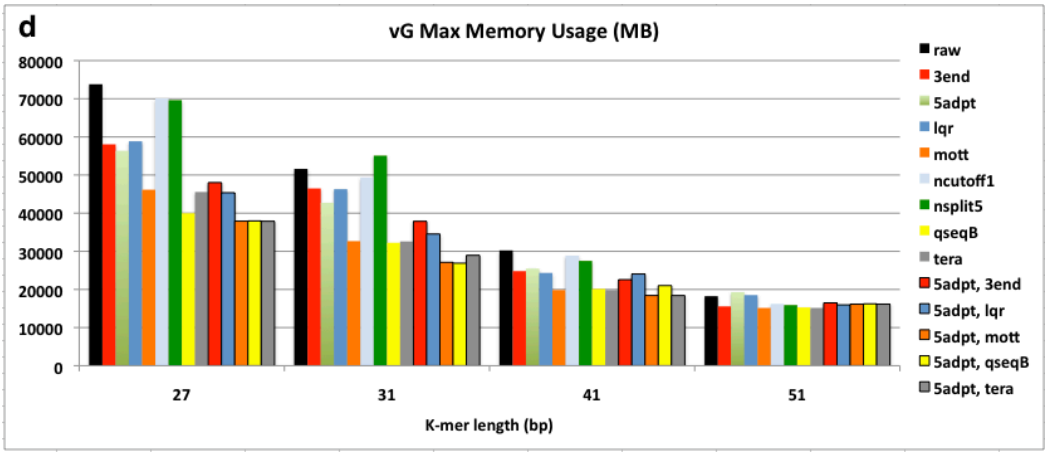
ngsShoRT was tested on a publically available (from NCBI Short Read Archive, Accession Number: SRX026594) *C. elegans* genome, sequenced with Illumina Genome Analyzer II, consisting of 3.3 million PE 100-bp reads (insert size: 356) which totaled to about 6.8 billion base pairs. This dataset was chosen because it was relatively complex (multicellullar eukaryote with ~100 Mbp genome, consisting of 7 large chromosomes), had an annotated reference genome and transcriptome, and used a manageable memory overhead when assembled with Velvet (maximum RAM usage was ~70 GB at K = 23, minimum was ~20 GB with K=41 on most of our machines).

A trimmed reads dataset was produced for each method by running ngsShoRT to implement this method on the raw *C. elegans* genome dataset. The *min_rl* was set to 52 bp, which was greater than all K-mer lengths used for assembly. Assembly was done using the popular DBG assembler Velvet (version 1.1.04) at different K-mer lengths (27, 31, 41, and 51 bp). ngsShoRT and Velvet were run on X86_64 Fedora Core 14 server with 256G RAM and 64 CPU/Core (2GHz). ngsShoRT was run using 60 threads, and the average runtime was 20 minutes per trimming job with a maximum RAM usage of ~200 MB for the total of the 60 threads. Assembly correctness was evaluated by aligning contigs against the *C. elegans* reference genome and complete proteome database using BLASTN and BLASTX (coverage i.d. = 90, e-value = 0.00001), respectively. Figure 2.3 shows N50 Contig Length (C.L.), BLAST correctness, and assembler performance results for these assemblies.




b
BlastN HCL/TCL * 100
96.6
96.4
96.2
96
95.8
95.6
95.4
95.2
95
94.8
94.6
27
31
41
51
K-mer length (bp)
raw
3end
5adpt
lqr
mott
ncutoff1
nsplit5
qseqB
tera
5adpt, 3end
5adpt, lqr
5adpt, mott
5adpt, qseqB
5adpt, tera


c
BlastX HCL/TCL * 100
95
90
85
80
75
70
27
31
41
51
K-mer length (bp)
raw
3end
5adpt
lqr
mott
ncutoff1
nsplit5
qseqB
tera
5adpt, 3end
5adpt, lqr
5adpt, mott
5adpt, qseqB
5adpt, tera


d
vG Max Memory Usage (MB)
80000
70000
60000
50000
40000
30000
20000
10000
0
27
31
41
51
K-mer length (bp)
raw
3end
5adpt
lqr
mott
ncutoff1
nsplit5
qseqB
tera
5adpt, 3end
5adpt, lqr
5adpt, mott
5adpt, qseqB
5adpt, tera

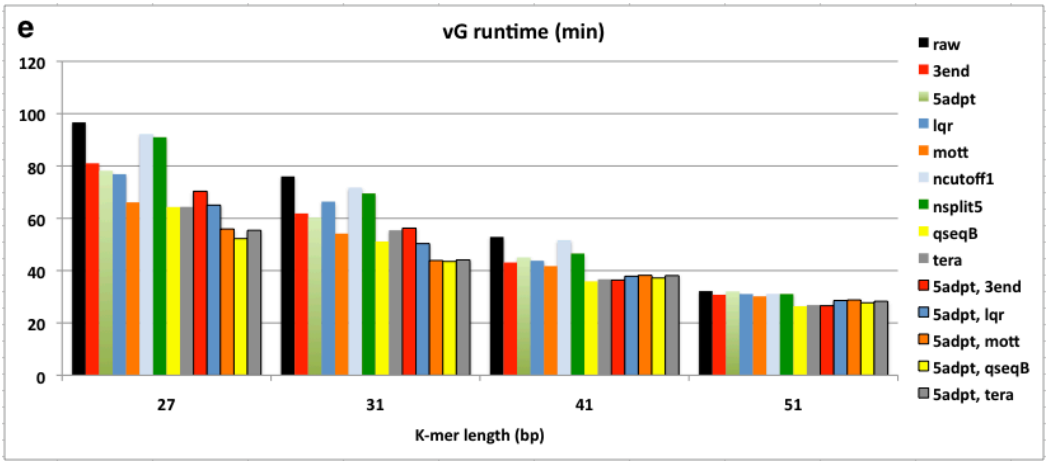


Figure 2.3 Assembly of ngsShoRT-trimmed datasets using Velvet. Each trimmed dataset is named after the sequence of ngsShoRT trimming algorithms used to create it from raw reads. For (a), Contiguity is measured as the N50 contig length (C.L.). For (b) and (c), Correctness is measured as the ratio of length of contigs (Hit Contig Length, HCL) aligning to the *C. elegans* reference genome and the complete proteome database using BLASTN and BLASTX, respectively. Contiguity and Correctness were measured for contigs with C.L. >= 200 bp. In (d) and (e), Velvet performance is measured as the maximum RAM usage and runtime of the DBG building and manipulation step (velvetG) of Velvet, respectively. 5adpt = *5adpt[mp =100, list=Illumina primers and adapters, search_depth = full_read_length, action=ka]*, nsplit5 = *nsplit[n ≥5],* ncutoff1 = *ncutoff[n=1]* = remove reads with any 'N' bases, tera = *TERA[avg =2],* lqr = *LQR[lqs =3, p ≥70%],* mott= *Mott[ml = 0.6],* 3end = *3end[x=7]*. For *3end*, x was set to 7 to compare it to TERA*[avg=2],* which removed about 7% of bases.

Overall, Velvet assemblies of from trimmed datasets had higher N50 contig length (C.L.) and BLAST correctness than raw dataset assemblies at all K-mer lengths, with the N50 increasing with higher K-mer lengths (Figure 2.3.a-c). VelvetG RAM usage and runtime were also considerably reduced, especially for low K-mer length assemblies where quality-trimming methods (*TERA, lqr, qseqB*) reduced them to 50% of the raw dataset's values. The high sensitivity smaller K-mer length

assemblies to quality is a unique feature of the DBG algorithm: a miscall in a small K-mer assembly can generate many more potential branches within the DBG than a miscall in a longer K-mer assembly would. So, the choice of K is dataset-dependent, and is a trade-off between specificity and sensitivity: shorter K-mers are more sensitive to base-call errors, while longer K-mers allow more specificity (i.e. less spurious overlaps) but lower coverage [63]. The higher sensitivity to base-call errors of smaller K-mer (23, 27 bp) assemblies makes quality-based trimming much more important than simple removal of reads with uncalled 'N' bases or low quality bases.

When comparing individual methods against each other, the following trends are noticed from Figure 2.3:

1. The improvement in contiguity, correctness and assembler performance was higher with methods that trimmed low quality bases and artifacts from reads (*5adpt-ka, 3end, TERA, Mott, qseqB-local-ka*) rather than filtering entire reads (*lqr, ncutoff*). This is interesting, since most assembly projects examined in section 2.4 used methods that filter entire reads and did not use methods that trim bases from reads.
2. *qseqB* greatly improved assembly. To our knowledge, ngsShoRT is the first tool to implement *qseqB* on Illumina reads in FastQ format by restoring the original Illumina ASCII-to-Phred scoring using *switch_scores*. Instead, a methods similar to *qseqB[mode=loca, action=ka]* was used for raw qseq-formatted Illumina reads by [12].
3. Removing all reads with 'N' bases (using *ncutoff*), a standard trimming step for assembly projects, and splitting reads around strings of 5 or more 'N'-bases using *nsplit* reads did not seem to improve contiguity nor correctness and

actually worsened Velvet's performance. This observation may be limited to our dataset, which had a low percentage of reads with uncalled bases (about 3%).

4. Not surprisingly, removing Illumina sequencing artifacts using *5adpt* improved all assembly measures, even with the simple parameters that we used for experiment [*match*=100% with no approximate matching or edits allowed in the sequence] in order to compare performance to the generally used method for adapter trimming in literature, which do not use approximate matching.
5. *TERA* greatly improved all four measures even with the low cutoff of 2, which was chosen in order to remove 'B'-scored bases (which would be equivalent to a score of 2 on Illumina's ASCII-Phred mapping) as well as lower-quality bases from read ends. In comparison to *qseqB*, *TERA*-trimmed read assemblies performed a little better in terms of contiguity, and much better in terms of BLAST correctness. So, *TERA* essentially removed regions similar to these removed by *qseqB* <u>*and*</u> other low quality bases from reads. This supports our belief that base-trimming, quality score-guided methods (*qseqB* does not really calculate quality scores, it simply searches for 'B' characters) outperform read-filtering or even base-trimming methods that do not use quality scores.
6. *TERA* was also compared to *3end[x]* by removing a similar number of bases. *TERA[avg=2]* removed ~7% of bases non-uniformly: a low-quality read may be trimmed down to *min_rl*, while a high-quality read may not be trimmed at all. In contrast, *3end* trims reads uniformly (the same *x* bases for all reads) regardless of each read's overall quality. *x* was set to 7 for *3end*, which resulted in removing exactly 7% of bases from our 100 bp reads. *TERA*-

trimmed read assemblies, with their higher average quality scores, outperformed *3end*-trimmed assemblies in assembler performance (Figure 2.3.d,e), as they result in less erroneous K-mers and thus less complicated DBGs. *TERA* also outperformed *3end* BLAST correctness (Figure 2.3.b,c).

In terms of contiguity (Figure 2.3), *TERA* performed similarly to *3end*, but for higher K-mer length assemblies (31, 41, 51 bp), *3end* outperformed *TERA*. Again, this can be explained by the balance of sensitivity and specificity coming from K: smaller K-mers assemblies are more sensitive to sequencing errors and thus prefer read correctness over length, while longer K-mers assemblies prefer read length (for more alignments) over read correctness and are less sensitive to sequencing errors. This is what is seen when comparing the N50 C.L. of *TERA*-trimmed datasets that have a non-uniform read length pool (read lengths between 100 bp and 52, the *min_rl*), to the uniform 93 bp lower quality read pool of *3end[x=7]*.

Next, I tried to find the best combination of trimming methods to improve assembly results. Only methods that improved our evaluation measures considerably were used (therefore, *ncutoff* and *nsplit* we discarded), and we always started by removing artifacts from reads using *5adpt*, followed by trimmin bases/reads using a quality-trimming method (*lqr, mott, TERA, qseqB)* or *3end*, which was used for comparison with *TERA*. Not surprisingly, using *5adpt* greatly improved all measures, making *[5adpt, 3end]* the best combination in terms of contiguity and BLASTX correctness, and *[5adpt, TERA]* the best combination in terms of BLASTN correctness and reducing Velvet's RAM usage and runtime. It's important to note that our choice of *x*=7 bp for *3end*, was based on the percentage of bases removed by *TERA* at *avg=2*,

rather than comparing trimming with different *x* values then selecting the one that resulted in the highest contiguity [1, 17].

2.6 Conclusions

This chapter presented ngsShoRT, a tool that implements several short read trimming methods adapted from assembly literature as well as methods that we created to trim single and paired-end NGS short reads in the FastQ and raw Illumina qseq sequence formats. Several combinations of ngsShoRT methods were tested on publicly available Illumina GA II eukaryotic genome data and showed that trimming improved the contiguity, correctness and performance of assemblies. I recommend the combination of 5'-adaptor-trimming and trimming of low quality bases from the 3'-ends of reads using our novel trimming method, *TERA*. ngsShoRT is written in the platform-independent programming language Perl and uses multiprocessing to manage high throughput read sets. ngsShoRT can be incorporated as a pre-processing step for genome and transcriptome sequencing pipelines to clean up NGS data to improve assembly contiguity and correctness as well as assembler performance. Further directions for ngsShoRT testing may include evaluating trimmed read assembly performance with other popular DBG assemblers (SOAPdenovo, CLC Genomics, etc), with different genome or simulated read datasets; and comparing the performance (runtime, memory usage, and improvement in assembly) of ngsShoRT to other popular NGS trimming tools.

Chapter 3

DETECTION AND REMOVAL OF EXOGENOUS SEQUENCE ARTIFACTS FROM NGS READS

The removal of primer/adaptor contamination is major step in trimming of NGS reads. The approach for trimming adaptor/primer sequences in literature is to use direct text-search algorithms to look for a user-specified set of sequences in the read set and remove any matching reads instead of trying to trim only the adaptor sequence. This approach requires *a priori* knowledge of platform-specific artifact sequences, does not usually allow for mismatches or the possibility of primer/adaptor sequence fragmentation. An alternative approach to adaptor/primer trimming that tries to manage these problems is to search for high frequency K-mers in the dataset, but is mostly used for detection and removal of tag sequences instead of sequence contaminants [41, 49].

This chapter begins with a discussion of currently used approaches for adaptor/primer trimming in literature. Next, an alternative approach that tries to detect exogenous sequences in NGS reads without prior knowledge of platform-specific artifacts is presented. This approach is implemented in kmerFreq, a tool written in Perl to detect and remove adapter sequences from NGS reads. kmerFreq is tested on 454 reads from viral metagenomes and its performance is evaluated by examining the sequences that it detected in these reads and the effect of trimming these sequences from reads on subsequent assembly with Newbler and Phrap.

2.1 Removing Primer/Adaptor Contamination in Literature

The general approach for removal of primer/adaptor sequences is to search for sequences from a user-specified list of known artifacts (such as Illumina and 454 primer/adaptor libraries and/or user-specified sequences) in the read dataset [1, 5, 8, 16, 26, 41, and others]. These tools use direct text-search algorithms that do not allow for mismatches or indels within the target sequence. A variation of this approach is to align all reads against the UniVec [58] database using BLAST, discarding any reads that match [10, 56]. This approach is computationally inefficient, especially for high throughout datasets, and does not account for adaptor/primer fragmentation either.

Even with the approximate matching function of ngsShoRT (*5adpt,* see section 2.5), it is difficult to predict if and how adaptor/primer sequences will be fragmented in the data, or adapt to the possibility of finding novel, dataset-specific artifacts. Therefore, an alternative method for adaptor or tag detection implements searching for over-represented sequences by obtaining a K-mer frequency table for the dataset reads. K-mers sequences are loaded a hash table and examining high frequency K-mers. This is implemented by [41] and [49], but it allows for only one mismatch per K-mer sequence during the process of K-mer table construction. Additionally, the extracted, non-tag, over-represented sequences were not tested to see if they actually reflected possible vector/adaptor/primer contamination, and their influence on subsequent assembly contiguity and assembler performance was assessed.

2.2 The kmerFreq Algorithm

My proposed approach to adaptor/primer detection is a variation of the aforementioned K-mer frequency approach and is implemented in kmerFreq, a tool written in Perl with multiprocessing functionality to manage high throughput reads.

kmerFreq's main function is to search for over-represented K-mers in a read sample using approximate matching with the Levenshtein edit distance implementation in CPAN's String::Approx module [19]. An optional additional functionality is to trim these overrepresented K-mers and/or user-specified adaptor sequences from the read dataset using an implementation of ngsShoRT's 5' adaptor trimming algorithm, *5adpt* (section 2.5). kmerFreq's algorithm is described below, and was tested on 454 reads datasets for viral metagenome specimens.

From both directions in a read, kmerFreq searches for over-represented K-mers in overlapping windows of size K. For each window, a local hash table of K-mers that uses approximate K-mer matching (allowing a certain mismatch percentage as well as specific values for allowed insertions, substitutions and deletions) is built at every iteration. K-mers in the window are approx-matched against this window's local hash table as well as a global K-mer table to allow for sliding of adaptor/primer sequences. K-mer table building continues up to a user-specified depth within the read. K-mers whose frequency exceeds a user-specified percentage cutoff are printed for every window (Left/Right inSpan K-mer table file) and for the global K-mer hash (Left/Right global K-mer table file).

In the second step of kmerFreq, these sequences are trimmed from reads (along with an optional adaptor/primer/linker sequence list provided by the user). Approximate matching is done for these sequences against all of the dataset reads within a user-specified matching range (the default range is the first 100 bases from the 5' and 3' ends). If a target sequence is detected within the first 100 5' bases, it is removed along with all bases 5' to it. So, if the 8 bp sequence **AGGACCGT** was found to start at 5'-33 in the read, the 5' bases 1 through 41 are trimmed from the read. The

same action is done from the 3'-end for sequences detected in the first 100 3'-bases. If multiple hits are found on the same read, trimming is done from the deepest hit.

2.3 Testing kmerFreq on 454 Reads of a Real Viral Metagenome

We tested the K-mer detection and trimming functionalities of kmerFreq on four Chesapeake Bay viral metagenome 454 datasets [Polson and Wommack, unpublished]. For each dataset, reads were clipped and converted from sff to FastA + FastA.qual files using Newbler's sffinfo function, which is the default trimming tool for 454 reads that trims out suspected primer and low quality sequences. Newbler clipped 13 5'-bases (all of which had the sequence **tcagaccgaatac**) and, on average, 84 3'-bases from all reads.

Next, kmerFreq was run on these clipped reads to search for over-represented 8-mers (exceeding a minimum percentage of 2% in sampled reads) in the read sample (10% of reads, picked at random) within the first 100 5' and 3' bases at a match percentage of 100%. Any detected over-represented left and right global 8-mers were then used by the trimming functionality of kmerFreq to trim all reads without using a list of known 454 primer/adaptor sequences. kmerFreq's trimming functionality removed, on average, about 2.4% of bases in each clipped reads dataset.

Interestingly, in all four datasets, kmerFreq detected the same high frequency overlapping global 8-mers within the first 30 5'-bases of reads, finding them in ~18% of reads. High frequency global and local 8-mers were also found in the 3'-ends of ~3% of reads. We aligned the left global 8-mer sequences using CLUSTALW (http://www.ebi.ac.uk/Tools/msa/clustalw2/), then used the alignment information to manually merge the sequences into fewer representative sequences:

>Left1

AGGACCGTTATAGTTAGG
>Left2
TCGCTACCTTAGGACCGTT
>Left3
CGCTACTTAGGACC
>Right1
TGGCAATATCAATC
>Right2
ATCCTGGCAAT
>Right3
GCGATGGAATCCTGG

Next, these sequences were aligned against the NR/NT database using BLASTN. Figure 3.1 shows top hits (with E-value < 5) for these sequences, which included several known cloning vectors. To evaluate the effect of trimming these sequences from the datasets, clipped reads (reads clipped using sffinfo) and trimmed-clipped reads (reads clipped using sffinfo, then trimmed using kmerFreq) that were generated from the same raw dataset were assembled using Newbler and Phrap [15].

During assembly of clipped reads, Newbler printed a warning about finding (using direct text match) "suspected 5' primer **TCGCTACCTTAGGACCGTTATAGTTA**" and the "suspected 3' primer **CGCTACCTTAGGACCGTTATAGTTA**" in ~17% and ~3% of reads, respectively. This warning was not printed for the trimmed-clipped reads. Interestingly, the merged Left(1-3) and Right(1-2) sequences were substrings of these suspected primers. So, Newbler's pre-assembly warning confirms that (1) kmerFreq's detection function reported valid hits because they are substrings of known primer sequences, and (2) kmerFreq's trimming function successfully removed all of these sequences (and their fragments and variant sequences, if any) from the trimmed dataset. This indicates that Newbler's sffinfo function left some untrimmed primer sequences (within the first 30 5' and 3'-bases of reads, constituting about ~2.4% of all bases) in the clipped reads, and that these sequences were trimmed out by kmerFreq.

Newbler assembly of trimmed-clipped reads had higher N50 contig length (972 bp), maximum contig length (13,400 bp) and total assembled bases (2,065,295 bp) values than those of clipped reads (N50: 952 bp, maximum contig length: 11,917 bp, total bases: 2,034,316). Assembly of trimmed-clipped reads took only 7 minutes, while assembly of clipped reads took 42 minutes on the same machine (Dell PowerEdge R415 server, 2 AMD Opteron 4238 six-core processors at 3.60 GHz (total of 12 cores), 128 GB of memory). Considering the fact that Newbler warned about primer sequences only in the clipped reads set, this suggests that the 6X increase in runtime for these reads (relative to trimmed-clipped reads) was caused by these primer sequences, further confirming that kmerFreq detected and removed the correct sequences.

Sequences producing significant alignments:

Accession	Description	Max score	Total score	Query coverage	E value	Max ident
BC085130.1	Mus musculus lipoma HMGIC fusion partner-like protein 4,	36.2	36.2	100%	1.0	100%
BC043103.1	Mus musculus serine/threonine kinase 36 (fused homolog,	36.2	36.2	100%	1.0	100%
BC060612.1	Mus musculus SRY-box containing gene 17, mRNA (cDNA	36.2	36.2	100%	1.0	100%
U34922.1	Cloning vector pI-LINK, with multiple cloning site cut by fo	36.2	36.2	100%	1.0	100%
HQ398620.1	Cloning vector pSI-DGB2, complete sequence	34.2	34.2	94%	4.1	100%
HQ398619.1	Cloning vector pSI-DCP2, complete sequence	34.2	34.2	94%	4.1	100%
HQ398617.1	Cloning vector pSI-DST2, complete sequence	34.2	34.2	94%	4.1	100%
HQ398614.1	Cloning vector pSI-DAL2, complete sequence	34.2	34.2	94%	4.1	100%
CP003011.1	Thielavia terrestris NRRL 8126 chromosome 3, complete s	34.2	34.2	94%	4.1	100%
FJ804477.1	Cloning vector TKVBL-w+, complete sequence	34.2	34.2	94%	4.1	100%
FJ804476.1	Retrofitting vector TKVBL-GVB, complete sequence	34.2	34.2	94%	4.1	100%
FJ804475.1	Expression vector plac-Cre, complete sequence	34.2	34.2	94%	4.1	100%
EU049865.1	Cloning vector pSAT5.nosP.RNAi, complete sequence	34.2	66.4	94%	4.1	100%
AY818368.1	Cloning vector pSAT5-EGFP-C1, complete sequence	34.2	66.4	94%	4.1	100%
AF128862.1	Cloning vector pHIND2.2, complete sequence	34.2	34.2	94%	4.1	100%

Figure 3.1 BLASTN matches in nr/nt for Left1 and Left, the merged sequences of high-frequency left 8-mers detected by kmerFreq. BLASTN was run using word length = 7, and all other parameters were set to their default values. Only hits with E-value < 5 are shown in this figure.

For Phrap [15] assemblies, the most remarkable effect of trimming was on the assembly runtime and memory usage: trimmed-clipped reads were assembled within 5 minutes and with a maximum RAM usage of 572 MB. In contrast, clipped reads assembly required 9 hrs (108X of the trimmed-clipped reads' assembly time) and a maximum RAM usage of 4.6 GB (8X of the trimmed-clipped reads' assembly memory usage). This difference is similar to what was seen with Newbler assemblies, which suggests that this 60X difference in runtime was caused by the primer sequences that were not trimmed out of the clipped reads and thus caused many faulty overlaps that complicated Phrap's assembly. Additionally, trimmed-clipped read assemblies had much higher N50 and Maximum contig length (767 bp and 28,214 bp, respectively) than clipped-read assemblies (540 bp and 16,919 bp, respectively).

Finally, kmerFreq was tested on artificial datasets, created by inserting artificial "adaptor" sequences into a subset of reads taken from one of the trimmed-clipped datasets from the previous experiments. Sequences of 6-11 bases that had very low frequency in the test dataset (< 0.01% of reads) were created and inserted them at random positions within the first 100 5' and 3' bases in 70% of reads (selecting reads at random). kmerFreq was run on the test dataset to detect over-represented (occurring in >= 5% of reads) 8-mers, using 100% match (no approximate matching) for building the K-mer frequency table. kmerFreq detected only the artificial sequences (or their fragments if they were longer than 8 bp) in the test dataset (data not shown).

In conclusion, I have created a tool that can accurately detect (with/without approximate matching) and remove primer/adaptor sequences and their fragments in NGS reads without prior knowledge of the platform and its expected artifact sequences. kmerFreq allows users to modify many parameters for K-mer search and

trimming, and to specify target sequences to be trimmed along with over-represented K-mers. kmerFreq was tested on newbler-trimmed 454 reads for real viral metagenome datasets, and showed that it detected substrings of known 5' and 3' primers and removed them successfully (Newbler did not detect the aforementioned primers in trimmed-clipped datasets) from the reads resulting in much faster and higher contiguity assemblies with Newbler and Phrap. Further directions for development of kmerFreq can include testing on other platforms (Illumina GA I, II, and HiSeq; 454 FLX+, etc) as well as automating the K-mer search to try different K-mer lengths for target dataset and determine the "best" set of K-mer sequences to use for trimming. It would also be interesting to study the relation between K-mer length and detected sequences in different platforms.

Chapter 4

CHIMERIC CONTIGS IN METAGENOME ASSEMBLIES

The field of metagenomics aims to examine the genetic diversity encoded by microbial life in organisms inhabiting a common environment by directly studying the genomic material without culturing [52]. Due to the reliance of metagenomics on sequencing, development of NGS technologies has provoked a profound impact in this field and has put metagenomic experiments within the range of many microbiological laboratories in terms of budget, time and work [27].

A major step in metagenomics projects is *de novo* assembly, which differs from single-genome *de novo* assembly in three main aspects: one, a "correct" solution or reference sequence for assembly evaluation are usually lacking in metagenomics; two, single-genome assembly algorithms are not well suited for metagenome assembly because they assume uniform read coverage depth and utilize coverage depth information to resolve repeats, remove errors, and construct contigs and scaffolds. Coverage depth is non-uniform in metagenomes, and high/low coverage regions are not necessarily repeats or errors; three, reads from different species can co-assemble into a single contig, a problem known as contig chimerism. Contig chimerism reduces the quality of assembly and hinders downstream analyses that rely on it, such as binning (association of sequences to taxonomic groups) and gene annotation, which assume that contigs consist of reads from the same species.

This chapter begins with a discussion of *de novo* metagenome assembly and its problems in general and contig chimerism in particular. Contig chimerism is very

difficult to resolve (especially within strains of the same species), and most approaches to managing this problem work at the pre-assembly (by binning/clustering reads by taxonomy, sequence, etc) or assembly (metagenome-specific assembly/scaffolding software) levels with limited success. I propose a post-assembly approach that tries to identify chimeric contigs in a metagenome assembly, and to break such contigs back to their original reads and re-assemble these reads into non-chimeric contigs. Our hypothesis is that chimeric contigs are different from non-chimeric contigs in certain features, such as coverage distribution and polymorphism (SNPs) levels, and that such features can be used to identify potentially chimeric contigs in an assembly without necessarily knowing the source species of their reads. Our hypothesis will be tested by studying chimeric and non-chimeric contig populations reads in assemblies of simulated metagenomes using a chimeric contig analysis pipeline presented in this chapter. The chapter ends with a discussion of analysis results on a simulated bacterial metagenome, the future directions for improving the pipeline and how it may be applied for detection of chimeric contigs in real metagenome assemblies.

4.1 Introduction to Metagenomics and Metagenome *De Novo* Assembly

Metagenomics can be defined as the application of shotgun sequencing of DNA samples obtained directly from environmental specimens or from living beings without culturing [28, 45]. In metagenomic studies that rely on NGS, the full genomic content of the communities is sequenced to obtain the bacterial composition and functional repertoire present in the environment of interest at the DNA or DNA-RNA level. This is a great improvement over the classic 16S rRNA surveys that were useful for taxonomic classification of bacterial and eukaryotic species [28, 52].

The workflow of a typical metagenome project is shown in Figure 4.1. The two major NGS platforms used in the DNA sequencing step of this workflow are 454 and Illumina [28, 33, 45]. However, The longer length of 454 reads (400-500 bp with GS FLX Titanium versus ~150 bp with Illumina HiSeq), shorter runtime (PE analysis requires only 1 day versus 10 instrument days with Illumina) and their lower single read error rate relative to Illumina makes it more the more favorable platform for metagenome assembly [57]. The longer read length of 454 is the most important of these advantages because it reduces the possibility of misassembly and improves downstream analyses, especially for low-abundance species and functional annotation [57, 60]. However, 454 sequencing technology has its drawbacks, which include a high amount of artificially duplicated reads and a bias towards indel errors (see section 2.2). Failure to remove duplicate reads can affect downstream analyses, such as differential assessment of gene fractions between two metagenomic communities [14, 57]. Indel errors, on the other hand, result in reading frameshifts if protein-coding sequences are called on a single read, which may be corrected during assembly as the consensus sequence is determined from multiple reads [57].

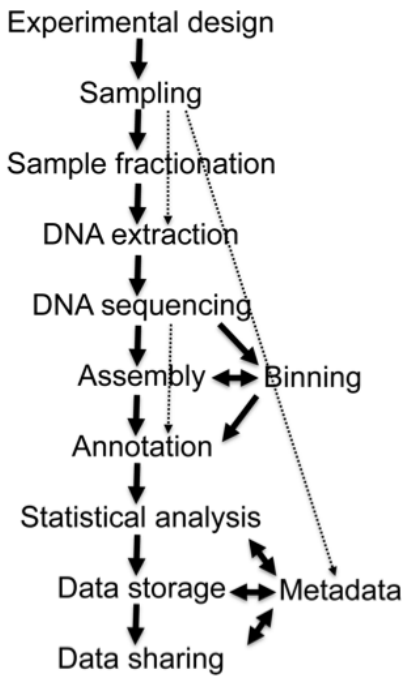


Figure 4.1 Flow diagram of a typical meta-genome project. Dashed arrows indicate steps that can be omitted. *From [57].*

An important attribute of metagenomes built from viral/microbial communities that can affect sequencing and assembly design is community complexity, which defined by Kunin *et al.* [28] as the function of the number of species in the community (richness) and their relative abundance (evenness). A community with more species that are closer to equal abundance is more complex than a community with less species that have unequal abundance [28]. Community complexity affects downstream analyses in general, and assembly in particular [28, 33, 45]. In literature, communities are commonly divided into low, moderate, and high-complexity communities [33]. Low-complexity (LC) communities are dominated by a single near-clonal population flanked by low-abundance populations, as in bioreactor communities. Assembly of LC datasets may result in a near-complete draft assembly of the dominant population [28, 33]. Moderately complex communities (MC) have more than one dominant population, also flanked by low-abundance ones, as has been observed in an acid mine drainage biofilm [33]. High-complexity communities lack a dominant species, such as agricultural soil communities, and result typically in minimal assembly without large genomic fragments of any component population.

The variation of organism abundance in metagenomes built from viral/microbial communities results in nonuniform read depth over the sequence and makes *de novo* assembly algorithms that assume clonal genomes less suitable for metagenomics. Most OLC and DBG assemblers (see section 2.3) were designed for single genomes, where the major challenge is repeat resolution [11, 27, 28, 35, 57, 63]. Both OLC and DBG assemblers use coverage depth information to identify unitigs or paths that appear to represent repetitive segments of a genome and keep them out of initial assembly [27, 35]. These coverage-based approaches work well with single

genomes, but can lead to false positives in metagenomic datasets. Coverage-based methods can classify abundant organisms as repeats, preventing the assembly of exactly those segments of the community that should be easily assembled [27, 57]. In contrast, three algorithms will classify low-coverage sequences derived from low-abundance organisms as sequencing errors, leading to false negatives in assembly. Due to these problems of metagenomic DBG assemblers, most 454 reads are still assembled using Newbler, the native OLC assembler of 454 technology, despite the fact that it is also not trained for metagenome assemblies.

The consequence of failing to separate similar read sequences that belong to different species or strains during assembly is the coassembly of these reads into the same contig, resulting in contig chimerism [27, 28, 33, 45]. Coassembly is more likely with reads from closely related genomes where the sequence similarity is higher, but has been found between reads originating from phylogenetically distant taxa, with conserved genes serving as the focal point for misassembly [28]. In addition to reducing assembly quality, contig chimerism hinders downstream analyses, such as binning and functional annotation (Figure 4.1). Binning, the process of associating sequence data (reads, contigs, scaffolds) with contributing species (or higher- level taxonomic groups) benefits from assembly because assembly should result in assembling reads from the same source species into the same contigs.

Several solutions have been proposed for reducing contig chimerism: pre-assembly approaches attempt to bin or cluster similar reads prior to assembly [45, 57], a process that is computationally expensive with high throughput short reads. Assembly approaches are done by modifying assembly algorithms (e.g., Celera and TIGR) or development of metagenomic assemblers that were discussed above, and

post-assembly pipelines try to assess contigs by aligning them to reference sequences and calculate the percent identity, which is not a reliable solution since a fundamental problem of metagenomics is that the "correct" or "reference" sequences are not usually available [28].

This chapter presents a post-processing approach for resolving chimeric contigs. This method will distinguish chimeric from non-chimeric contigs in an assembly by identifying potential chimeric regions in a contig by their coverage, polymorphism, and logistical attributes, which are expected to be significantly different from their contig and analogous non-chimeric regions on that contig. These features will be analyzed in assemblies of NGS reads of simulated metagenomes using our chimeric contig analysis pipeline, which is presented in the following section.

4.2 The Chimeric Contig Simulation and Analysis Pipeline

The basic design of the pipeline is shown in Figure 4.2. createMetagenomes (steps 1-4) is used to create raw FastA reads for an artificial metagenome with a custom selection of organisms and coverages raised to a user-specified sequencing coverage, X. Next, createMetagenomes converts the raw reads generated into Illumina FastQ or 454 sff files using ngsfy (step 5), a program designed by Miguel Pignatelli [45] to simulate NGS platform reads (Single- or Paired-end), with or without sequencing errors.

The next step in the pipeline is the assembly of reads using an assembler that provides read-to-contig tracking information, such as Celera, Newbler, SSAKE, and Velvet [45]. Since the CM-generated test dataset consisted of 454 sff reads, Newbler (step 6) was used for assembly, which stores read-to-contig tracking information in its 454Contigs.ace file. This information was used by contigAnalyzer (step 7) to generate

the detailed contig reports (listed in section 4.4) for every contig. contigAnalyzer separates contig reports into two main categories: chimeric and non-chimeric. It was necessary to identify a detection threshold for our algorithm such that chimeric contigs passing this cutoff are believed to be detectable based on specific attributes in coverage and polymorphism caused by the co-assembly of reads from different species. Chimeric contigs with coverages above this cutoff are referred to as "significantly chimeric" contigs. For the initial test run, this threshold was represented by a coverage cutoff: chimeric contigs with coverages above this cutoff were detectable. This coverage cutoff and the detection threshold can be subsequently "tuned" for our algorithm with more test runs on simulated and real data.

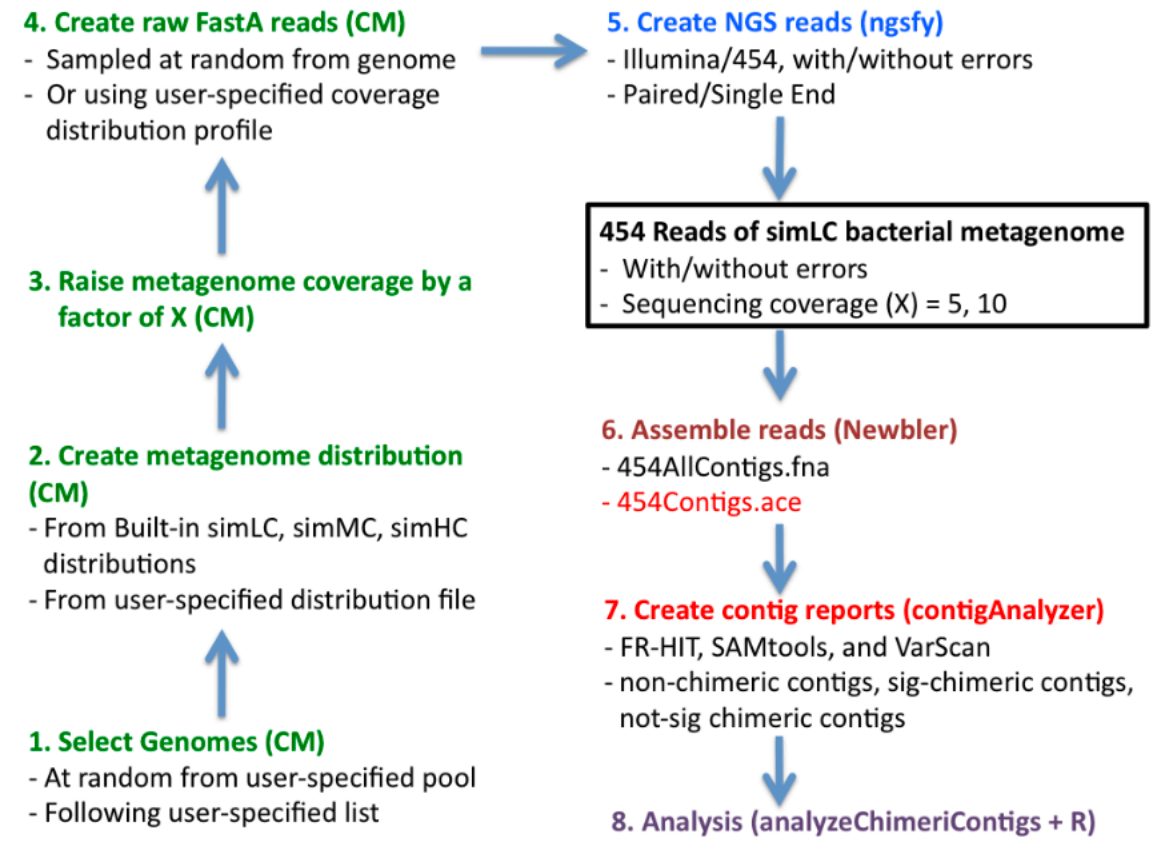


Figure 4.2 The chimeric contig simulation and analysis pipeline.

Therefore, chimeric contigs are divided into two subcategories: significantly chimeric (sig-chimeric) and not-significantly chimeric (not-sig-chimeric). A contig is "significantly chimeric" if it contains a chimeric region that passes certain cutoffs (discussed in section 4.4), the most important of which is the aforementioned coverage cutoff. contigAnalyzer divides a chimeric contig by its source species into "slices," so that each slice has different source(s) from its adjacent slices. A slice is chimeric if it has multiple sources, and is sig-chimeric (significantly chimeric) if it passes the coverage cutoffs discussed in section 4.4. Our hypothesis is that sig-chimeric slices are different in coverage, polymorphism, and other features from other non-chimeric slices in their parent contig, and that the presence of sig-chimeric slices can be used to differentiate chimeric from non-chimeric contigs.

The contig reports of non-chimeric, sig-chimeric, and not-sig-chimeric contigs are analyzed and contrasted by analyzeChimericContigs (step 8), a script that generates multiple large (60+ column) information tables for contigs and all slices within contigs that can be loaded into R for statistical analysis. The current version (1.0) of analyzeChimericContigs produces the following reports:

1. *Taxonomy report:* for every species used in the simulated reads set, analyze-ChimericContigs reports its contribution in all, chimeric, and sig-chimeric contigs. In addition, it reports (and prints a taxonomy tree of) all organisms whose reads co-assembled with reads from this species.

2. *Non-, sig-chimeric, and Not-sig-chimeric contig reports*: these reports include information (length, taxa, LCA, coverage statistics, polymorphism, number of slices and sig-chimeric slices, etc) for every contig in each group.
3. *Sig-slice report*: this report includes detailed information for each sig-slice in the assembly, as well as the sig-slice's parent contig, and analogous non-chimeric slice(s) on the same contig. This report is important because it allows to test our hypothesis: that sig-slices have distinguishing coverage, polymorphism, and statistical features that differentiate them from their parent contig and other non-chimeric regions on the parent contig.

4.3 Generation of Simulated Metagenomes: createMetagenomes

Since the pipeline relies on tracking reads and their source species to contigs, it needs a metagenome simulation tool that (1) can create simulated reads with source species information, (2) allows us to manipulate individual genome coverage as well as overall metagenome sequencing coverage, (3) produces platform-specific read formats that included quality score information, such as FastQ/qseq for Illumina and sff for 454, and (4) offers pre-built artificial metagenome coverage distributions commonly used in literature. Metasim [47], the popular metagenome simulation software, satisfies the first and second requirements but fails the third and fourth requirements, as it produces only flat FastA files without quality score information and users have to design metagenomes from scratch.

Therefore, createMetagenomes (CM) a metagenome-simulation pipeline designed in cooperation with Miguel Pignatelli (Wellcome Trust Genome Campus, EBI) was created for the chimeric contig analysis pipeline. CM was written in Perl

with a detailed usage manual and supports all of the above requirements. The steps of the CM pipeline are shown in Figure 4.2 (steps 1-5) and are as follows:

Step 1 *Selection of genomes*. CM uses the genome FastA files provided by the user (such as NCBI's viral or bacterial genome repository) to generate a "genome pool." Genomes are then picked from this pool at random or following a user-specified list. In the random pick case, the number of genomes to pick follows the number of entries in the coverage distribution used in Step 2.

Step 2 *Creation of Metagenome Distribution*. CM offers the user three pre-built coverage distributions that are based on the artificial metagenomes designed by Mavromatis *et al.* [33], which consist of three different complexity metagenomes (simLC, simMC, simHC). These datasets have been used by about 20 metagenomic benchmarking studies since their publication in 2007 [33, 45]. Users can select and edit any of these distributions, or create a custom distribution.

Step 3 *Raise the sequencing coverage by X-fold*. X is an integer value (1 by default), and the coverage values from Step 2 are simply multiplied by X to simulate raising the sequencing coverage of the metagenome specimen. This is useful for studying effect of varying sequencing on metagenome assembly and chimeric contig formation [45].

Step 4 *Create raw FastA reads*. By default, reads of user-specified length range (or using the defaults of 100-100 bp for Illumina GA II and 400-400 bp for 454 Titanium) are picked from genomes, and each genome is "sequenced" by a number of reads that corresponds to its target coverage (the product of the coverage specified in Step 2 and the X value in Step 3). So, the number of

reads per genome = *(cov · g)/r*, where *cov* = the genome's target coverage, *g* = genome length, and *r* = minimum read length (specified in the length range).

By default, reads are picked at random from the genome sequence (with a special case for circular bacterial genome sequences) with no region-specific bias, i.e., read coverage is uniform over the genome sequence. However, the user can simulate region-specific bias (such as known amplicon sequences – see section 2.2) using a distribution profile option that forces sampling to be higher at certain regions than others. The reads are generated as FastA sequences, and the read header includes the organism's gi, taxid, scientific name (loaded from NCBI's taxonomy files), read length, and the read's location in the genome.

Step 5 *Convert FastA reads to NGS reads*. The final step of the CM pipeline is the conversion of the raw FastA reads to Illumina or 454 sff reads (with/without platform errors and mate pair features), and is done by ngsfy, a tool created by Miguel Pignatelli to simulate Illumina and 454 reads in [45].

CM serves as a user-friendly interface for ngsfy that allows multiprocessing to speed up simulations of large metagenomes. This step can be run on flat FastA reads independently of the previous steps, which is useful for two reasons: one, the FastA reads used by ngsfy can be generated from CM (steps 1-4) or any other simulation software that produces FastA files, like Metasim [47]; two, the same raw FastA reads can be subsequently used for several runs of Step 5 to generate reads for different platforms or with different error profiles and study the effect of changing the platform and its errors on the behavior of the same original read sequence.

4.4 Analysis of Read-to-Contig Alignment Information: contigAnalyzer

Our analysis pipeline needed a tool that can generate per-base coverage, polymorphism, and taxonomy reports for contig assemblies from read-to-contig tracking files. I did not find such a tool in literature, and thus created contigAnalyzer, a tool written in Perl 5.6 that utilizes object-oriented programming, multiple bioinformatics applications, and BioPerl modules to produce detailed analysis reports for the three contig populations of our pipeline's simulated metagenome assemblies: non-chimeric, sig-chimeric, and not-sig-chimeric contigs. The ultimate purpose of our analysis pipeline is to train contigAnalyzer and make it capable of identifying potentially chimeric contigs in real metagenomic assemblies (where read sources are unknown) based on a set of distinguishing coverage and polymorphism features deduced from our analysis pipeline.

The workflow of contigAnalyzer is shown in Figure 4.3. For every contig, its read-to-contig tracking information (which provided by several assemblers including Celera, Velvet, SSAKE, Newbler, etc) is used to generate a CONTIG object, which consists of individual BASE and READ objects. This object-oriented design makes contigAnalyzer independent of the assembler's format for read-to-contig tracking information (Newbler's ace file format, Velvet's and Celera's amos files, etc) and also facilitates designing methods that manipulate CONTIGs and their corresponding READs. READ sequences are re-aligned to their CONTIG using FR-HIT [39] (step 3), whose output can also be used to generate pileup files using SAMtools (step 4), which can then be used by VarScan [25] to predict SNPs (step 5).

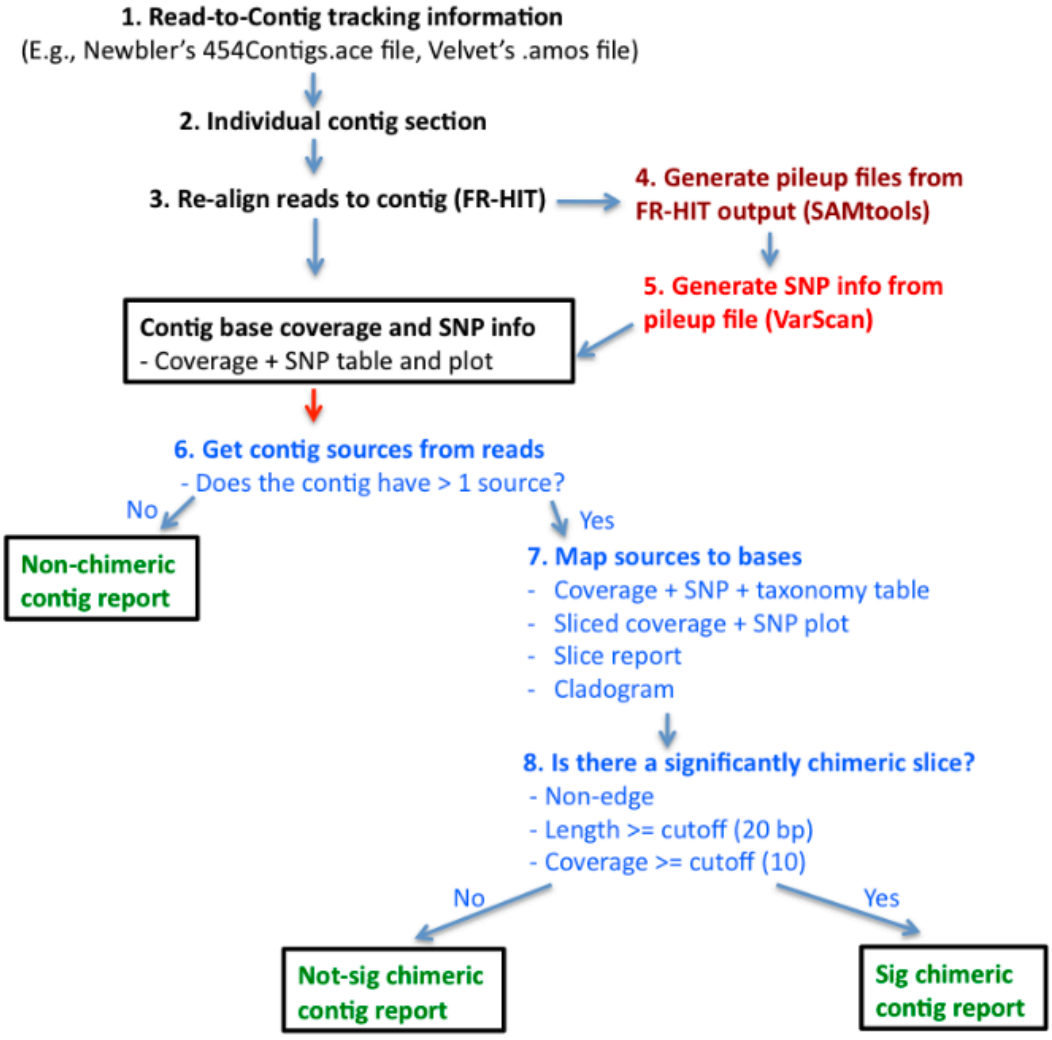


Figure 4.3 The contigAnalyzer pipeline.

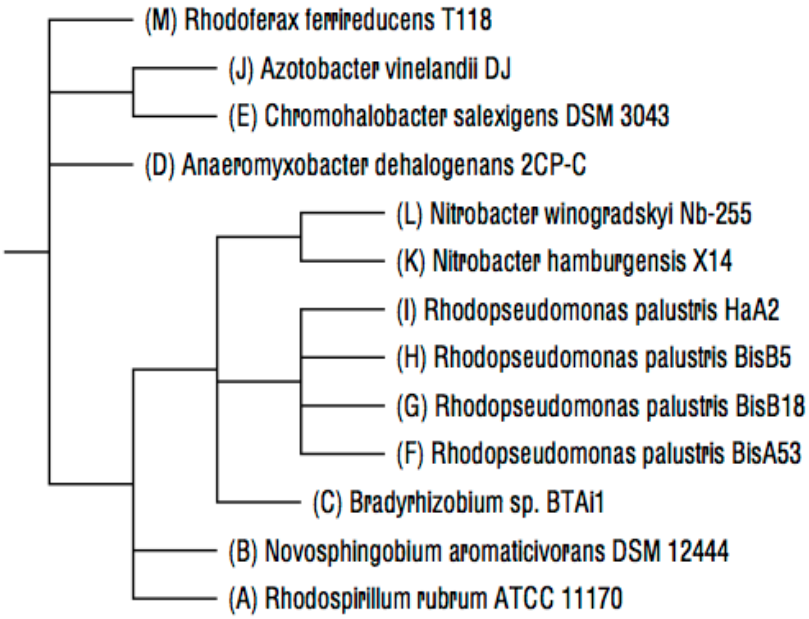


Figure 4.4 Cladogram for a significantly chimeric contig. This cladogram was generated by contigAnalyzer (step 7) for the largest contig in the X5 assembly. The LCA for this tree is the Phylum Proteobacteria.

Steps 1 through 5 produce read coverage and SNP information for every base in the contig as well as the contig's coverage statistics, reported in detailed information tables and as a png coverage plot. Next, steps 6 through 8 (blue) are done for chimeric contig analysis in assemblies of reads generated by createMetagenomes (CM). CM-generated reads include their source taxonomy information (NCBI taxid) in their headers. Therefore, contigAnalyzer can determine if a contig is chimeric by examining the sources of reads covering its bases, separating chimeric (multiple source) from non-chimeric contigs (single source) in step 6.

In step 7, the taxonomy of chimeric contigs is analyzed using a special module that utilizes NCBI's Taxonomy Browser tool [36], which builds a taxonomy tree for the contig's sources. This tree is then loaded into BioPerl's **`Tree`**, **`Taxonomy`**, and **`cladogram`** modules [3] to predict the lowest common ancestor (LCA) of the contig's

sources and generate a cladogram of these sources, such as the cladogram shown in Figure 4.4. Next, contigAnalyer breaks the contig into several "slices," where each slice is a substring of the contig's bases that differs from adjacent slices in its source species. So, if a contig consists of reads from sources A and B so that reads from A and B co-assemble only in the middle of the contig, then the contig consists of three slices: the slice from A, the slice from AB, and the slice from B. Slice AB is the chimeric slice within the contig, and is "significantly chimeric" (sig-chimeric) if it passes certain cutoffs: (1) it must be longer than a cutoff length (default is 20 bp), (2) its overall read coverage is no less than a coverage cutoff (default is 10 reads), and (3) this read coverage must be divided somewhat evenly between species: each species in the slice must contribute no less than a cutoff number of reads to the coverage (default is 3). An example of a contig region with sig-chimeric, not-sig-chimeric, and non-chimeric slices is shown in Figure 4.5.

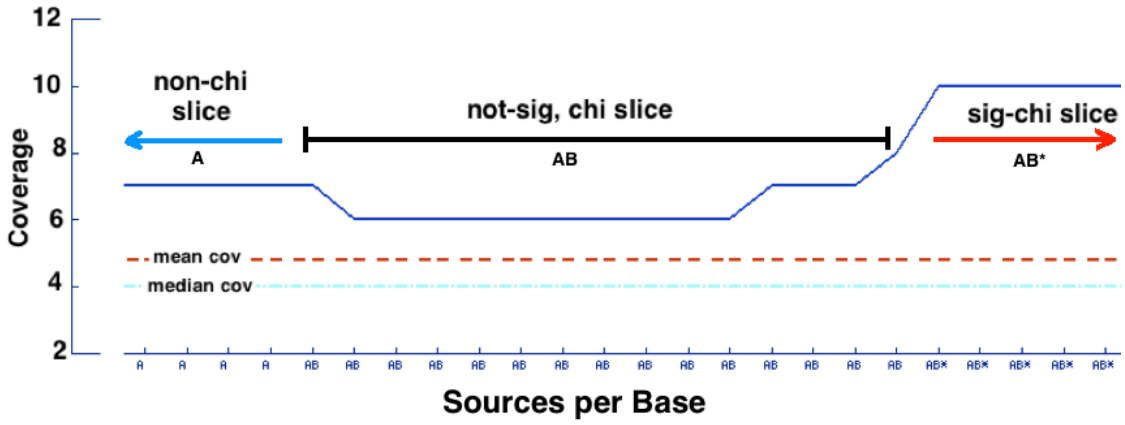


Figure 4.5 Types of slices in a chimeric contig. This is a section of the sliced coverage plot generated by contigAnalyzer (step 7) for a contig in the X5 assembly. The X-axis shows the sources of each bases (represented by characters: A= *Xylella fastidiosa 9a5c*, B = *Xylella fastidiosa M12*), and the Y-axis shows the coverage for every base. A non-chi (non-chimeric) slice has one source, while a not-sig (not significant) slice is chimeric but not "significant" for analysis because it did not satisfy the coverage and/or length cutoffs (10, 20, respectively). Base sources of the sig-chi slice are suffixed with an asterisk. The parent contig of these slices is labeled "sig-chimeric" because it had at least one sig-chi slice. Not-sig-chimeric contigs have non-chi and not-sig chi slices but no sig-chi slices.

The location of slices on contigs is normalized by dividing each contig into 100 intervals that will be referred to as "cents," and identifying each slice's location by its cent span. This is useful for comparing the location of sig-chimeric slices on different contigs. Chimeric contigs that contain at least one non-edge (not in the first or last N cents, with N = 5 by default) sig-chimeric slice are labeled "sig-chimeric," and "not-sig-chimeric" otherwise (step 8). Edge slices are discarded because contig edges tend to have low and irregular coverage, which may affect analysis of sig-chimeric slices. So, in summary, the sig-chimeric contig population consists of contigs with at least one, non-edge sig-chimeric slice whose length, coverage, and per-species coverage satisfy user-specified cutoffs (20 bp, 10, and 3 are the default values, respectively).

4.5 Analysis Results for Simulated Bacterial Metagenome Assemblies

The analysis pipeline was tested on the artificial bacterial metagenomes (simLC, simMC, simHC) designed by Mavromatis et al. [33] to facilitate benchmarking of metagenomic data processing methods. Since their publication, these datasets have been simulated by about 20 metagenomic benchmarking studies,

including a study by Pignatelli and Moya [45] that analyzed the assemblies of simulated short (100 bp) Illumina and long (400 bp) 454 reads of the simLC, simMC, simHC, and simHC_hc (simHC with a sequencing coverage of 10) metagenomes. This pipeline was tested on the simLC dataset as a baseline, since it was reported to have a lower occurrence of chimeric contigs than simMC and simHC assemblies by [45].

The species list and coverage distribution of the simLC dataset for steps 1 and 2 of the pipeline, then created two different sequencing coverage read sets in Step 3 with X = 5 and 10 (because X =1 resulted in only two sig-chimeric contigs in our initial assemblies), creating the X5 and X10 flat FastA datasets in step 4. Then, for each dataset, Step 5 was run to generate 400 bp sff reads of 454 Titanium with and without errors (error parameters were set to ngsfy's default values). Therefore, four sff read datasets were generated: X5 and X10 (no errors), X5_e and X10_e (errors). The sequencing coverage of the X5 dataset is comparable to a plate of 454 reads. Each dataset was then assembled using Newbler (step 6 of the analysis pipeline, see Figure 4.2).

Table 4.1 provides information about each dataset's reads and its assembly statistics. Not surprisingly, the no-error datasets (X5 and X10) had a higher N50 than error datasets (X5_error and X10_error). However, the X10 sets had a smaller N50 and maximum contig length than the X5 dataset: X5 had the largest maximum contig length (455,147 bp), which was a significantly chimeric contig (LCA : the Proteobacteria, a Phylum), as shown in Figure 4.4. As expected, the main contributing species to the longest contig in all dataset assemblies was the dominant species (*Rhodopseudomonas palustris HaA2*: 5,331,656 bp, coverage = 25.95X), with different species aligning to its sequences at different regions of the contig.

Table 4.1 Simulated 454 dataset and their Newbler *de novo* assembly statistics**.** Contig statistics were derived from each assembly's 454AllContigs.fna, excluding contigs shorter than 400 bp (read length). C.L. = contig length.

Data-set	# Reads	# Bases	Min. Cov.	Max. Cov.	# Contigs	N50 C.L.	Max C.L.	Total Assembly length
X5	1,080,987	432,394,800	0.35	25.95	8,324	4,776	455,147	21,463,012
X5_e	1,080,987	432,394,800	0.35	25.95	8,320	4,649	581,484	21,263,753
X10	2,161,920	864,768,000	0.7	51.9	21,798	3,552	276,717	39,410,064
X10_e	2,161,920	864,768,000	0.7	51.9	22,698	3,028	314,555	39,271,810

Table 4.2 Analysis pipeline results for the simulated test dataset assemblies. The information shown in this table was derived from contig reports generated by analyzeChimericContigs (see section 2.4). *chi contigs* = chimeric contigs, *sig-chi contigs* = significantly chimeric contigs, *sig-chi contigs with LCA > spp* = significantly chimeric contigs where the rank of the LCA reported by NCBI's Taxonomy Browser was higher than "species" (genus and above), whereas other sig-chimeric contigs were formed by different species strains of the same organism.

Dataset	# Contigs	# chi contigs (% of contigs)	# sig-chi contigs (% of chi contigs)	# sig-chi-contigs with LCA > spp (% of sig-chi contigs)
X5	8,324	1,448 (17.4%)	29 (2.0%)	13 (44.8%)
X5_e	8,320	1,325 (15.9%)	25 (1.9%)	7 (28.0%)
X10	21,798	3,033 (13.9%)	349 (11.5%)	61 (17.5%)
X10_e	22,698	2,057 (9.1%)	293 (14.2%)	49 (16.7%)

Next, contig chimerism in these assemblies was assessed using contigAnalyzer and analyzeChimericContigs (steps 7 and 8 of the analysis pipeline). In general, increasing sequencing coverage resulted in reducing the percentage of chimeric contigs (Table 4.2), which may be due to improved accuracy of assembly with higher coverage. The addition of sequencing errors also resulted in lowering the percentage of chimeric contigs probably because these errors reduced the sequence similarity

among reads from different species. These trends were also observed with the fraction of chimeric contigs that were significantly chimeric (Table 4.2).

Interestingly, the ratio of sig-chimeric contigs that had an LCA above species was reduced two fold with the increase in sequencing coverage. Analysis of the taxonomic composition of these chimeric contigs showed that the abundance of a species did not seem to be directly related to its contribution in chimeric contigs. For example, the dominant organism, *Rhodopseudomonas palustris HaA* contributed less to chimeric contigs than lower-abundance organisms such as *Xylella fastidiosa* (in the X10 and X5 assemblies) and *Burkholderia* species (in the X10 assembly). Another interesting observation was that the five different species of *Burkholderia* formed more chimeric contigs than the four strains of the same species of *Rhodopseudomonas* in the X10 dataset, which may suggest that these *Burkholderia* species have more conserved sequences among them than the four strains of *Rhodopseudomonas palustris*, or that the higher coverage of *Rhodopseudomonas palustris* resulted in a more accurate assembly (Table 4.3). This conclusion is supported by the difference in the taxonomic composition of chimeric contigs between the X10 and the X5 species: in the higher coverage X10 datasets, the highest chimera-contributing organisms were mostly closely related species or strains. In contrast, the highest chimera-contributing organisms in the lower coverage X5 dataset consisted of more distant species, and the X5/X5_error assemblies had a higher percentage of chimeric contigs forming by distant species than X10/X10_error assemblies (Tables 4.2 and 4.3).

In summary, sequencing coverage was the main factor affecting chimeric contig formation: higher coverage results in a smaller percentage of chimeric contigs (Table 4.2), most of which were formed by reads of closely related species and strains

(Table 4.3). So, a possible explanation of the higher N50 and maximum contig lengths of the X5/X5_error compared to the X10/X10_error assemblies is that they had a higher level of contig chimerism and a higher likelihood of co-assembly of reads from distant species (Tables 4.2 and 4.3). To confirm this, the slicing report of the largest contig (contig000001) of each assembly were compared as follows: contig000001 (length: 455,147 bp) of the X5 dataset consisted of 525 distinct slices and was formed by reads from 13 different organisms whose LCA was the Phylum Proteobacteria. In contrast, contig000001 of the X10 dataset consisted of only 340 slices formed by reads from 7 different organisms whose LCA was the Class Alphaproteobacteria.

Table 4.3 **Taxonomy reports for simulated metagenome assemblies.** This table shows a subset of the Taxonomy report file produced by our pipeline for Newbler assemblies of the X10, X10_error, and X5 datasets, showing the top 18 organisms contributing to sig-chimeric contigs. Organisms from the same species/genus are highlighted using the same color. Note that the "*percentage of (Not-) significantly chimeric contigs*" equals the "percentage of (Not-) significantly chimeric contigs with an LCA rank higher than species" + "percentage of (Not-) significantly chimeric contigs formed by strains of the same species."

cov	Species	%all	%non chi	%sig Chi	%sig Chi (LCA > spp)	%not Sig Chi	%not Sig Chi (LCA > spp)
			X10				
4.4	Xylella_fastidiosa_9a5c	3.92	1.14	61.46	1.19	4.95	0.19
2	Xylella_fastidiosa_M12	3.52	0.6	61.46	1.19	4.81	0.05
1.1	Burkholderia_cenocepacia_HI2424	9.07	0.09	14.03	1.78	28.08	1.68
1.1	Rhodopseudomonas_palustris_BisA53	2.74	0.94	13.64	11.46	5.7	2.31
1.2	Rhodopseudomonas_palustris_BisB5	2.01	0.95	13.64	11.26	3.3	1.8
1.3	Burkholderia_383	8.97	0.5	13.44	1.78	26.93	1.7
1.3	Rhodopseudomonas_palustris_BisB18	3.4	1.88	13.24	10.87	5.82	2.69
51.9	Rhodopseudomonas_palustris_HaA2	0.68	0.02	13.24	10.87	1.01	0.74
1.2	Burkholderia_ambifaria_MC40_6	8.6	0.36	13.24	1.98	26.04	1.73
1.1	Burkholderia_vietnamiensis_G4	7.79	0.51	12.85	1.78	23.12	1.59
1.1	Burkholderia_cenocepacia_AU_1054	7.67	0.1	12.65	1.78	23.61	1.34
1.2	Nitrobacter_winogradskyi_Nb_255	2.33	0.86	7.71	7.71	5.05	1.85
0.9	Burkholderia_xenovorans_LB400	3.26	0.65	6.92	1.98	8.6	1.35
1.3	Shewanella_MR_7	10.42	0.47	4.35	0.4	32.48	0.26
1.1	Shewanella_oneidensis	7.46	1.17	3.75	0.4	21.39	0.29
1.3	Shewanella_ANA_3	10.37	0.59	3.56	0.4	32.14	0.29
1.1	Shewanella_baltica_OS155	4.68	1.05	3.16	0.4	12.65	0.21
			X10 error				
4.4	Xylella_fastidiosa_9a5c	5.48	1.14	68.47	1.18	7.93	0.25
2	Xylella_fastidiosa_M12	4.86	0.31	68.47	1.18	7.74	0.06
1.1	Burkholderia_cenocepacia_HI2424	10.19	0.08	14.35	1.65	34.98	2.23
1.3	Burkholderia_383	10.13	0.31	14.12	1.88	34.19	2.08
1.2	Burkholderia_ambifaria_MC40_6	9.6	0.18	13.65	1.65	32.68	2.23
1.1	Burkholderia_cenocepacia_AU_1054	8.84	0.09	13.18	1.65	30.2	1.82
1.1	Burkholderia_vietnamiensis_G4	8.9	0.3	12.94	1.65	29.92	2.01
1.2	Rhodopseudomonas_palustris_BisB5	1.51	0.8	9.65	8.94	2.2	1.13
51.9	Rhodopseudomonas_palustris_HaA2	0.6	0.01	9.65	8.94	0.85	0.5
1.1	Rhodopseudomonas_palustris_BisA53	2.28	0.94	9.41	8.94	4.69	2.2
1.3	Rhodopseudomonas_palustris_BisB18	2.93	1.77	9.18	8.94	5	2.45
1.2	Nitrobacter_winogradskyi_Nb_255	1.7	0.75	7.53	7.53	3.3	1.23
0.9	Burkholderia_xenovorans_LB400	3.6	0.51	6.82	2.35	10.92	1.51
1	Shewanella_W3-18-1	3.08	0.44	2.82	0.71	9.72	0.09
1.1	Rhodospirillum_rubrum_ATCC_11170	0.85	0.82	2.59	2.59	0.69	0.69
1.1	Shewanella_oneidensis	5.5	0.55	2.35	0.24	18.34	0.19
1.3	Shewanella_ANA_3	8.38	0.36	2.12	0	29.32	0.22
			X5				
2.2	Xylella_fastidiosa_9a5c	13.57	2.29	36.84	1.75	28.53	0.56
1	Xylella_fastidiosa_M12	12.3	0.32	36.84	1.75	28.19	0.22
0.55	Rhodopseudomonas_palustris_BisA53	2.69	0.32	35.09	33.33	5.35	4.83
5.55	Bradyrhizobium_BTAi1	26.94	36.94	33.33	33.33	13.21	13.21
0.6	Rhodopseudomonas_palustris_BisB5	1.79	0.16	33.33	33.33	3.47	3.09
26	Rhodopseudomonas_palustris_HaA2	0.96	0.07	33.33	33.33	1.61	1.52
0.65	Rhodopseudomonas_palustris_BisB18	2.81	0.48	33.33	31.58	5.45	4.86
0.6	Nitrobacter_winogradskyi_Nb_255	2.11	0.3	21.05	21.05	4.24	3.65
0.55	Azotobacter_vinelandii_DJ	0.9	0.66	12.28	12.28	1.02	1.02
0.45	Pseudomonas_fluorescens_Pf0_1	0.82	0.11	12.28	12.28	1.58	1.05
0.55	Rhodospirillum_rubrum_ATCC_11170	0.39	0.11	12.28	12.28	0.56	0.56
0.6	Novosphingobium_aromaticivorans_D!	0.48	0.18	10.53	10.53	0.71	0.71
0.4	Rhodobacter_sphaeroides_KD131	0.34	0	10.53	10.53	0.62	0.62
0.45	Chromohalobacter_salexigens_DSM_3	0.21	0.05	7.02	7.02	0.31	0.31
0.6	Polaromonas_JS666	0.78	0.57	7.02	7.02	0.96	0.96
0.55	Pseudomonas_syringae_phaseolicola	0.82	0.41	7.02	7.02	1.27	0.87
0.55	Rhodoferax_ferrireducens_T118	0.36	0.2	7.02	7.02	0.46	0.46

Sig-chimeric slices were analyzed and compared to analogous non-chimeric slices (non-edge slices with similar length) on the same contig. In all of our datasets, sig-chimeric slices were generally found at the middle of their contig (Figure 4.6) and had significantly higher mean coverage than their parent contig and analogous non-chimeric slices (Figure 4.7). No significant differences were found in terms of SNP percentage or quality scores between these regions (data not shown). However, these results confirm our hypothesis: that (significantly) chimeric contigs differ from non-chimeric contigs in the presence of high coverage sig-chimeric regions resulting from the co-assembly of reads from different organisms at regions of sequence similarity.

Next, the sequences of sig-chimeric slices formed from co-assembly of distant species (LCA rank above genus) were analyzed to understand why their sampled reads would co-assemble at these slices. To predict the function of these sig-chimeric slice sequences, they were aligned against NCBI's nr/nt database using BLASTN, and the results for some of these slices are shown in Table 4.4. As expected, these slice sequences matched to regions of important and highly preserved functions such as DNA replication (DNA gyrase), protein synthesis (Elongation factor Tu, 16S and 23S ribosomal RNA), and biosynthesis pathways (3-dehydroquinate synthase, 1-deoxy-D-xylulose-5-phosphate synthase). This agrees with the observations of [28] and [45], and suggests that conserved functions can be used a hotspots for detection of chimeras. Additionally, this maybe be extended to all ORFs: contigs can be searched for ORFs of conserved functions that can be used as markers of potential chimeras.

Table 4.4 Possible functions of chimeric regions on contigs with an LCA rank above genus. Several sig-slices from the Sig-slice report generated by analyzeChimericContigs (see section 4.4) were aligned against the nr/nt database using BLASTN (with the Entrez query field set to "bacteria"). The table shows the dataset, slice ID (parent contig ID and slice location on contig in 1/100ths of contig length), the LCA (and its rank), and the expected function of this region. The expected function of these slices was deduced from the BLAST report by choosing the most common feature of the subject sequences. Note that contig01335 was almost entirely chimeric (some slices are not shown in this table) and aligned to the rRNA-16S ribosomal RNA sequence in members of the family Pasteurellaceae.

Data-set	slice ID	LCA	Expected function of sequence
X5	contig00071_52-53	Bacteria (SuperKingdom)	3-dehydroquinate synthase (shikimate pathway)
X5	contig00005_75-75	Bradyrhizobiaceae (Family)	1-deoxy-D-xylulose-5-phosphate synthase, a protein of Bradyrhizobiaceae bacteria
X5	contig02868_62-71	Bacteria (SuperKingdom)	rRNA-23S ribosomal RNA
X5	contig04219_49-56	Proteobacteria (Phylum)	rRNA-23S ribosomal RNA
X10	contig00007_94-94	Alphaproteobacteria (Class)	Elongation factor Tu (EF-Tu)
X10	contig00320_59-61	Bacteria (SuperKingdom)	DNA gyrase subunit B/DNA topoisomerase IV subunit B
X10	contig01335_24-33*	Pasteurellaceae (Family)	rRNA-16S ribosomal RNA
X10	contig00147_50-51	Proteobacteria (Phylum)	polyribonucleotide nucleotidyltransferase protein
X10	contig20615_6-35	Bradyrhizobiaceae (Family)	DNA-directed RNA polymerase subunit alpha
X10	contig11891_6-24	Gammaproteobacteria (Class)	rRNA-23S ribosomal RNA
X10	contig20786_87-94	Actinomycetales (Order)	16S ribosomal RNA gene

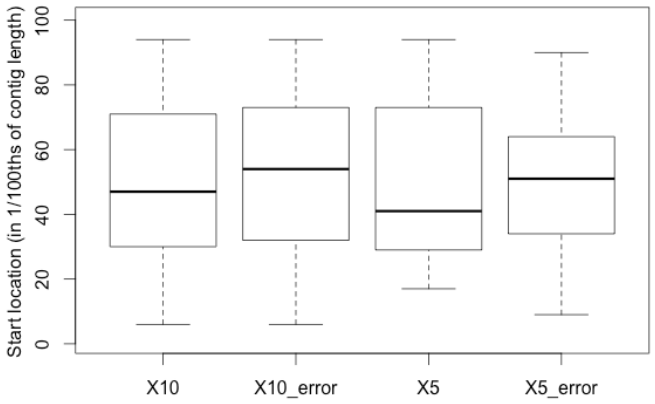


Figure 4.6 Location of sig-chimeric slices on contig.

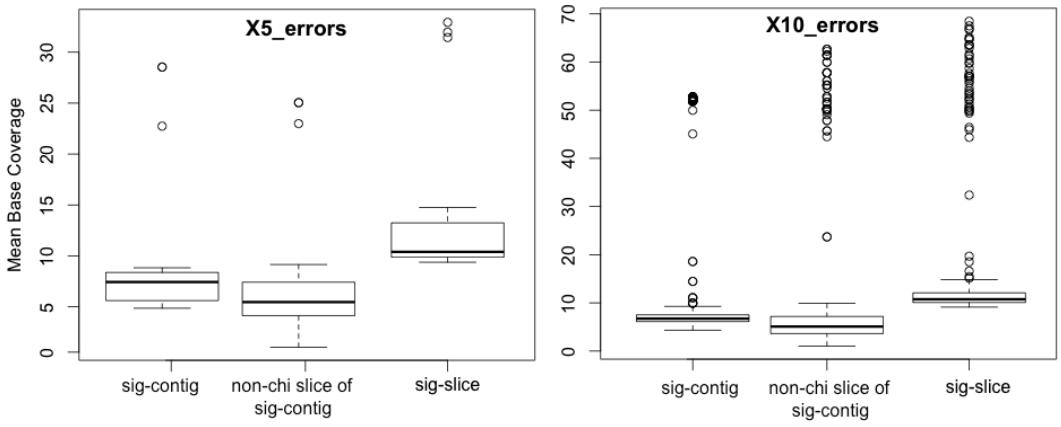


Figure 4.7 Mean base coverage of significantly chimeric contigs (sig-contig) and their significantly chimeric (sig-) and non-chimeric (non-chi) slices.

4.6 Conclusion

This chapter presented a complete chimeric contig simulation and analysis pipeline, starting from the creation of NGS (Illumina or 454) reads for simulated metagenomes and ending at the analysis of chimeric versus non-chimeric contig populations in terms of coverage, polymorphism, quality scores, and taxonomic distribution. The pipeline can be used to create and analyze any artificial metagenome (viral, bacterial, etc) and with any assembler that can provide read-to-contig tracking information in order to trace reads (and their source species) to their contigs. The goal of the pipeline is to analyze chimeric contigs and identify distinguishing attributes (coverage, polymorphism, location on contig, etc) of chimeric "slices" (the regions at which reads from different species co-assemble) in these contigs. I believe that we can use these distinguishing attributes to identify potentially chimeric contigs in a real metagenome assembly where read sources are unknown.

The training dataset for this pipeline was derived from the artificial bacterial metagenome designed by Mavromatis *et al.* [33], and was used to generate four simLC

datasets that differed in sequencing coverage and/or the presence or absence of sequencing errors: X5, X5_error, X10, and X10_error. Significantly chimeric slices (non-edge, chimeric co-assembly regions on chimeric contigs whose length was >= 20 bp and were covered by >= 10 reads) were analyzed and compared to their parent contig and analogous slices (similar length and analogous location on contig) on the parent contig. These sig-chi-slices had a significant difference from their parent contig and analogous slices in coverage levels and general localized at the middle of their contig. Sig-chi-slices with an LCA higher than species generally belonged to regions of conserved function among distant species.

The main factor affecting contig chimerism in Newbler assemblies of these datasets was sequencing coverage: a two fold increase in coverage (X5 to X10) increased chimeric contigs over 10 fold and changed the taxonomic distribution of species in contigs. In contrast, sequencing errors slightly affected contig chimerism, resulting in less chimeric contigs in datasets with errors. Considering that the X5 is comparable to the typical 454 sequencing unit used by metagenome researchers, it is interesting that such coverage with a low complexity community is not enough to overcome contig chimerism, while the X10 coverage does overcome it.

The majority of chimeric contigs were formed by the co-assembly of different strains of the same species, which was expected since strains of different species have the highest sequence similarity. Still, a few chimeric contigs with an LCA rank as high as kingdom and super-kingdom were found and analyzed, and their sequences were found to correspond to highly-conserved functional regions (DNA replication, translation, and biosynthesis pathways). At fixed sequencing coverage, sequencing errors significantly reduced the ratio of such higher-LCA contigs, which may explain

why the X10/X10_error assemblies had lower contiguity than the X5/X5_error assemblies. Finally, analysis of significantly chimeric slices generally localized at the middle of their contigs, and had significantly higher base-coverages (means of base-coverages) than their parent contigs and analogous non-chimeric slices on their parent contigs.

These results confirm our hypothesis, which is that chimeric contigs can be distinguished from non-chimeric contigs by the presence of small regions of significantly higher coverage that correspond to the region of co-assembly of reads from different species. However, our results may be dataset-dependent and our detection threshold for chimeras may require further tuning. Therefore, future directions for our analysis of simulated metagenomes must include varying the complexity as well as the composition of our tested metagenomes, such as using simLC, simMC, and simHC distributions of different viral or bacterial metagenomes. These variations can be easily simulated and analyzed using our pipeline. While the test runs focused on 454 reads and Newbler, they can be used to study and compare contig chimerism with Illumina and/or new metagenomic assemblers, such as Meta-velvet [38] and Genovo [29].

The final goal for the chimeric contig simulation and analysis pipeline is to identify set of coverage and polymorphism features that can be used to identify chimeric contigs in a real metagenome assembly in order to break these contigs into their source reads and re-assemble these reads with the non-chimeric contigs. These features will be then incorporated into contigAnalyzer and validate its performance by testing it on a real metagenome assembly. Without a priori knowledge of the community complexity or the number of sources in a contig, contigAnalyzer will

identify “possible chimeric contigs” based on coverage, polymorphism, and local features concluded from analyses of simulated datasets. Next, these contigs will be managed by simply breaking them back to their reads, then reassembling these reads more stringently. At worst, the reads will re-assemble into the same “potentially chimeric” contigs, and at best they will assemble into multiple smaller contigs that represent same or closer species. Validation of detection will be done by examining the contigs labeled as “potentially chimeric” (along with their source reads) by sequence analysis, binning, or gene/ORF/conserved function analysis to see whether they are really chimeric or not. Evaluation of management will be done by assessing the difference in downstream analyses (functional binning and gene prediction) results between managed and unmanaged “potentially chimeric” contigs.

Chapter 5

CONCLUSION

The purpose of this thesis was to develop tools for improving *de novo* assembly of NGS reads through pre-processing of NGS reads prior to single and metagenome *de novo* assembly, and management of chimeric contigs after *de novo* metagenome assembly. The analysis of these tools' performance provided important insights into problems with NGS assembly that may have been neglected by the majority of recent assembly projects and the how managing these problems can significantly improve assembly contiguity, correctness, and assembler performance.

In ngsShoRT, I implemented popular trimming methods (as well as methods developed by our group) and analyzed their resulting improvement in terms of assembly contiguity, correctness, and assembler performance for Illumina reads. I showed that higher contiguity does not always correspond to higher correctness or better assembler performance. This is interesting, since most assembly projects rely on contiguity alone to assess trimming method performance. In kmerFreq, I implemented an algorithm method for detection and removal of sequence artifacts in NGS reads without *a priori* knowledge of their platform's error profile. I compared trimming of 454 reads by kmerFreq to trimming by the 454-specific Newbler platform. kmerFreq resulted in better assembly contiguity and assembler performance, which is interesting since most projects that use 454 reads rely solely on Newbler for trimming 454 sequencing artifacts. For both ngsShoRT and kmerFreq, I showed that a better method for managing reads with adaptor or low quality bases is to trim these regions out of the

reads rather than filtering out the reads, while most assembly projects rely on read-filtering as the major trimming step. In addition, both tools significantly reduced the computational resources required for assembly of trimmed reads. Computational resources are the major bottleneck for most NGS assembly projects, as assembly algorithms require exponentially larger resources to assemble erroneous reads.

In my work with *de novo* metagenome assembly, I focused on contig chimerism, a problem that hinders the quality of metagenomic *de novo* assembly as well as downstream analyses including binning and gene prediction. To study contig chimerism, I designed a chimeric contig simulation and analysis pipeline to analyze chimeric contigs and the regions of co-assembly of reads from different species. My analysis showed that sequencing coverage was the most important factor affecting the occurrence of contig chimerism in *de novo* assemblies, and suggested that the current level of sequencing coverage of 454 (GS FLX Titanium), the popular platform for metagenomics, may result in a significant level of contig chimerism that was probably not managed post-assembly. The analysis also showed that chimeric contig reads co-assemble into relatively high coverage sequences that correspond to regions of conserved function among different species, which raises questions about the reliability of binning and gene prediction analyses done for assembled contigs in recent metagenomic studies that relied on NGS reads. Finally, the results of this analysis can be used to determine a set of methods that can be used to predict chimeric contigs in a real metagenome assembly where read sources are unknown, and to manage these contigs in order to reduce their level of chimerism, thus improving the quality of *de novo* assembly as well as binning and gene prediction.

REFERENCES

1. R Atherton, B McComish, L Shepherd, L Berry, N Albert, and P Lockhart. Whole genome sequencing of enriched chloroplast DNA using the Illumina GAII platform. *Plant Methods*, 6(1):22, 2010.

2. S Balzer, K Malde, A Lanzén, A Sharma, and I Jonassen. Characteristics of 454 pyrosequencing data—enabling realistic simulation with flowsim. *Bioinformatics*, 26(18):i420-i425, 2010.

3. Bioperl Tree and Taxonomy modules. A HOWTO page for their usage is available online at http://www.bioperl.org/wiki/HOWTO:Trees

4. 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):810-820, 2008.

5. CLC Bio, CLC Bio Genomics Workbench User Manual. Available online at http://www.clcbio.com/files/usermanuals/CLC_Genomics_Workbench_User_Manual.pdf

6. P A Cock, C J Fields, N Goto, M L Heuer, and P M Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research*, 38(6):1767-1771, 2010.

7. M Cox, D Peterson, and P Biggs. SolexaQA: At-a-glance quality assessment of Illumina second-generation sequencing data. *BMC Bioinformatics*, 11(1):485, 2010.

8. S DiGuistini, N Liao, D Platt, G Robertson, M Seidel, S Chan, T Docking, I Birol, R Holt, M Hirst, E Mardis, M Marra, R Hamelin, J Bohlmann, C Breuil, and S Jones. De novo genome sequence assembly of a filamentous fungus using Sanger, 454 and Illumina sequence data. *Genome Biology*, 10(9):R94, 2009.

9. D Earl, K Bradnam, A Darling, D Lin, J Fass, H K Yu, V Buffalo, D R Zerbino, M Diekhans, N Nguyen, P N Ariyaratne, W Sung, Z Ning, M Haimel, J T Simpson, N A Fonseca, I Birol, T R Docking, I Y Ho, D S Rokhsar, R Chikhi, D Lavenier, G Chapuis, D Naquin, N Maillet, M C Schatz, D R Kelley, A M Phillippy, S Koren, S Yang, W Wu, W Chou, A Srivastava, T I Shaw, J G Ruby, P Skewes-Cox, M Betegon, M T Dimon, V Solovyev, I Seledtsov, P Kosarev, D Vorobyev, R Ramirez-Gonzalez, R Leggett, D MacLean, F Xia, R Luo, Z Li, Y Xie, B Liu, S Gnerre, I MacCallum, D Przybylski, F J Ribeiro, S Yin, T Sharpe, G Hall, P J Kersey, R Durbin, S D Jackman, J A Chapman, X Huang, J L DeRisi, M Caccamo, Y Li, D B Jaffe, R E Green, D Haussler, I Korf, and B Paten. Assemblathon 1: A competitive assessment of de novo short read assembly methods. *Genome Research*, 21(12):2224-2241, 2011.

10. J Falgueras, A Lara, N Fernandez-Pozo, F Canton, G Perez-Trabado, and M G Claros. SeqTrim: a high-throughput pipeline for pre-processing any type of sequence read. *BMC Bioinformatics*, 11(1):38, 2010.

11. P Flicek and E Birney. Sense from sequence reads: methods for alignment and assembly. *Nat Meth*, 6(11s):S6-S12, 2009.

12. T I Garcia, Y Shen, J Catchen, A Amores, M Schartl, J Postlethwait, and R B Walter. Effects of short read quality and quantity on a de novo vertebrate transcriptome assembly. *Comparative Biochemistry and Physiology Part C: Toxicology & Pharmacology*, 155(1):95-101, 2012.

13. S D Goldberg, J Johnson, D Busam, T Feldblyum, S Ferriera, R Friedman, A Halpern, H Khouri, S A Kravitz, F M Lauro, K Li, Y Rogers, R Strausberg, G Sutton, L Tallon, T Thomas, E Venter, M Frazier, and J C Venter. A Sanger/pyrosequencing hybrid approach for the generation of high-quality draft assemblies of marine microbial genomes. *Proceedings of the National Academy of Sciences*, 103(30):11240-11245, 2006.

14. V Gomez-Alvarez, T K Teal, and T M Schmidt. Systematic artifacts in metagenomes from complex microbial communities. *ISME J*, 3(11):1314-1317, 2009.

15. Green P. Phrap, version 1.090518. http://phrap.org., 2011.

16. Hannon Lab (unpublished), FASTX-Toolkit. http://hannonlab.cshl.edu/fastx_toolkit/

17. S Haridas, C Breuill, J Bohlmann, and T Hsiang. A biologist's guide to de novo genome assembly using next-generation sequence data: A test with fungal genomes. *Journal of Microbiological Methods*, 86(3):368-375, 2011.

18. O Harismendy, P Ng, R Strausberg, X Wang, T Stockwell, K Beeson, N Schork, S Murray, E Topol, S Levy, and K Frazer. Evaluation of next generation sequencing platforms for population targeted sequencing studies. *Genome Biology*, 10(3):R32, 2009.

19. Hietaniemi J. String-Approx, version 3.26. http://search.cpan.org/~jhi/String-Approx-3.26/Approx.pm, 2011.

20. S Hoffmann, C Otto, S Kurtz, C M Sharma, P Khaitovich, J Vogel, P F Stadler, and J Hackermüller. Fast Mapping of Short Sequences with Mismatches, Insertions and Deletions Using Index Structures. *PLoS Comput Biol*, 5(9):e1000502, 2009.

21. X Huang. An Improved Sequence Assembly Program. *Genomics*, 33(1):21-31, 1996.

22. Illumina. De Novo Genome Assembly Using Illumina Reads. Technical Note: Sequencing, 2010.

23. W Kao, A H Chan, and Y S Song. ECHO: A reference-free short-read error correction algorithm. *Genome Research*, 21(7):1181-1192, 2011.

24. D Kelley, M Schatz, and S Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome Biology*, 11(11):R116, 2010.

25. D C Koboldt, K Chen, T Wylie, D E Larson, M D McLellan, E R Mardis, G M Weinstock, R K Wilson, and L Ding. VarScan: variant detection in massively parallel sequencing of individual and pooled samples. *Bioinformatics*, 25(17):2283-2285, 2009.

26. Y Kong. Btrim: A fast, lightweight adapter and quality trimming program for next-generation sequencing technologies. *Genomics*, 98(2):152-153, 2011.

27. S Koren, T J Treangen, and M Pop. Bambus 2: scaffolding metagenomes. *Bioinformatics*, 27(21):2964-2971, 2011.

28. V Kunin, A Copeland, A Lapidus, K Mavromatis, and P Hugenholtz. A Bioinformatician's Guide to Metagenomics. *Microbiology and Molecular Biology Reviews*, 72(4):557-578, December 2008.

29. J Laserson, V Jojic, and D Koller. Genovo: <i>De Novo</i> Assembly for Metagenomes. In Bonnie Berger, editors, *Research in Computational Molecular Biology*, volume 6044 of *Lecture Notes in Computer Science*, pages 341-356. Springer Berlin / Heidelberg, 2010.

30. T Lassmann, Y Hayashizaki, and C O Daub. TagDust—a program to eliminate artifacts from next generation sequencing data. *Bioinformatics*, 25(21):2839-2840, 2009.

31. R Li, Y Li, K Kristiansen, and J Wang. SOAP: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713-714, 2008.

32. E R Mardis. The impact of next-generation sequencing technology on genetics. *Trends in Genetics*, 24(3):133-141, 2008.

33. K Mavromatis, N Ivanova, K Barry, H Shapiro, E Goltsman, A C McHardy, I Rigoutsos, A Salamov, F Korzeniewski, M Land, A Lapidus, I Grigoriev, P Richardson, P Hugenholtz, and N C Kyrpides. Use of simulated data sets to evaluate the fidelity of metagenomic processing methods. *Nat Meth*, 4(6):495-500, 2007.

34. E Meyer, G Aglyamova, S Wang, J Buchanan-Carter, D Abrego, J Colbourne, B Willis, and M Matz. Sequencing and de novo analysis of a coral larval transcriptome using 454 GSFlx. *BMC Genomics*, 10(1):219, 2009.

35. J R Miller, S Koren, and G Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315-327, 2010.

36. NCBI's Taxonomy Browser (http://www.ncbi.nlm.nih.gov/Taxonomy/CommonTree/wwwcmt.cgi)

37. K Nakamura, T Oshima, T Morimoto, S Ikeda, H Yoshikawa, Y Shiwa, S Ishikawa, M C Linak, A Hirai, H Takahashi, M Altaf-Ul-Amin, N Ogasawara, and S Kanaya. Sequence-specific error profile of Illumina sequencers. *Nucleic Acids Research*, 39(13):e90, 2011.

38. T Namiki, T Hachiya, H Tanaka, and Y Sakakibara. MetaVelvet : An extension of Velvet assembler to de novo metagenome assembly from short sequence reads, ACM Conference on Bioinformatics, *Computational Biology and Biomedicine*, 2011.

39. B Niu, Z Zhu, L Fu, S Wu, and W Li. FR-HIT, a very fast program to recruit metagenomic reads to homologous reference genomes. *Bioinformatics*, 27(12):1704-1705, 2011.

40. R Pandey, V Nolte, and C Schlotterer. CANGS: a user-friendly utility for processing and analyzing 454 GS-FLX data in biodiversity studies. *BMC Research Notes*, 3(1):3, 2010.

41. R , Patel and M , Jain. NGS QC Toolkit: A Toolkit for Quality Control of Next Generation Sequencing Data. *PLoS ONE*, 7(2):e30619, 2012.

42. Y Peng, H M Leung, S M Yiu, and F L Chin. Meta-IDBA: a de Novo assembler for metagenomic data. *Bioinformatics*, 27(13):i94-i101, 2011.

43. P A Pevzner, H Tang, and M S Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748-9753, 2001.

44. M Pignatelli and A Moya. Evaluating the Fidelity of De Novo Short Read Metagenomic Assembly Using Simulated Data. *PLoS ONE*, 6(5):e19984, 2011.

45. Pignatelli M. ngsfy (unpublished). Uncompiled version available for download at https://github.com/emepyc/NGSfy.

46. M Rho, H Tang, and Y Ye. FragGeneScan: predicting genes in short and error-prone reads. *Nucleic Acids Research,* 38(20):e191, 2010.

47. 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):e3373, 2008.

48. L Salmela and J Schröder. Correcting errors in short reads by multiple alignments. *Bioinformatics*, 27(11):1455-1461, 2011.

49. R Schmieder, Y Lim, F Rohwer, and R Edwards. TagCleaner: Identification and removal of tag sequences from genomic and metagenomic datasets. *BMC Bioinformatics*, 11(1):341, 2010.

50. T Schoenfeld, M Liles, K E Wommack, S W Polson, R Godiska, and D Mead. Functional viral metagenomics and the next generation of molecular tools. *Trends in Microbiology*, 18(1):20-29, 2010.

51. J Schröder, J Bailey, T Conway, and J Zobel. Reference-Free Validation of Short Read Data. *PLoS ONE*, 5(9):e12681, 2010.

52. J Shendure and H Ji. Next-generation DNA sequencing. *Nat Biotech*, 26(10):1135-1145, 2008.

53. V Shulaev, D J Sargent, R N Crowhurst, T C Mockler, O Folkerts, A L Delcher, P Jaiswal, K Mockaitis, A Liston, S P Mane, P Burns, T M Davis, J P Slovin, N Bassil, R P Hellens, C Evans, T Harkins, C Kodira, B Desany, O R Crasta, R V Jensen, A C Allan, T P Michael, J C Setubal, J Celton, D G Rees, K P Williams, S H Holt, J R Rojas, M Chatterjee, B Liu, H Silva, L Meisel, A Adato, S A Filichkin, M Troggio, R Viola, T Ashman, H Wang, P Dharmawardhana, J Elser, R Raja, H D Priest, D W Bryant, S E Fox, S A Givan, L J Wilhelm, S Naithani, A Christoffels, D Y Salama, J Carter, E L Girona, A Zdepski, W Wang, R A Kerstetter, W Schwab, S S Korban, J Davik, A Monfort, B Denoyes-Rothan, P Arus, R Mittler, B Flinn, A Aharoni, J L Bennetzen, S L Salzberg, A W Dickerman, R Velasco, M Borodovsky, R E Veilleux, and K M Folta. The genome of woodland strawberry (Fragaria vesca). *Nat Genet*, 43(2):109-116, 2011.

54. J T Simpson, K Wong, S D Jackman, J E Schein, S J Jones, and I Birol. ABySS: A parallel assembler for short read sequence data. *Genome Research*, 19(6):1117-1123, 2009.

55. A Stein, T E Takasuka, and C K Collings. Are nucleosome positions in vivo primarily determined by histone-DNA sequence preferences?. *Nucleic Acids Research*, 38(3):709-719, 2010.

56. The Gene Index Project. SeqClean. Available online at http://compbio.dfci.harvard.edu/tgi/software/

57. T Thomas, J Gilbert, and F Meyer. Metagenomics - a guide from sampling to data analysis. *Microbial Informatics and Experimentation*, 2(1):3, 2012.

58. UniVec Database. Available online for download at ftp://ftp.ncbi.nih.gov/pub/UniVec/

59. D Willner, R V Thurber, and F Rohwer. Metagenomic signatures of 86 microbial and viral metagenomes. *Environmental Microbiology*, 11(7):1752-1766, 2009.

60. K E Wommack, J Bhavsar, and J Ravel. Metagenomics: Read Length Matters. *Applied and Environmental Microbiology*, 74(5):1453-1463, March 1, 2008.

61. T Woyke, G Xie, A Copeland, J M González, C Han, H Kiss, J H Saw, P Senin, C Yang, S Chatterji, J Cheng, J A Eisen, M E Sieracki, and R Stepanauskas. Assembling the Marine Metagenome, One Cell at a Time. *PLoS ONE*, 4(4):e5299, 2009.

62. D R Zerbino and E Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821-829, 2008.

63. Zerbino D. Velvet Manual (version 1.1, unpublished). Available online at http://bioweb2.pasteur.fr/docs/velvet/Manual.pdf, 2008.

64. Q Zhao, Y Wang, Y Kong, D Luo, X Li, and P Hao. Optimizing de novo transcriptome assembly from short-read RNA-Seq data: a comparative study. *BMC Bioinformatics*, 12(Suppl 14):S2, 2011.