ABSTRACT

Title of dissertation: NOVEL METHODS FOR COMPARING AND EVALUATING SINGLE AND METAGENOMIC ASSEMBLIES

Christopher Michael Hill, Doctor of Philosophy, 2015

Dissertation directed by: Professor Mihai Pop
Department of Computer Science

The current revolution in genomics has been made possible by software tools called genome assemblers, which stitch together DNA fragments "read" by sequencing machines into complete or nearly complete genome sequences. Despite decades of research in this field and the development of dozens of genome assemblers, assessing and comparing the quality of assembled genome sequences still heavily relies on the availability of independently determined standards, such as manually curated genome sequences, or independently produced mapping data. The focus of this work is to develop reference-free computational methods to accurately compare and evaluate genome assemblies.

We introduce a reference-free likelihood-based measure of assembly quality which allows for an objective comparison of multiple assemblies generated from the same set of reads. We define the quality of a sequence produced by an assembler as the conditional probability of observing the sequenced reads from the assembled sequence. A key property of our metric is that the true genome sequence maximizes

the score, unlike other commonly used metrics.

Despite the unresolved challenges of single genome assembly, the decreasing costs of sequencing technology has led to a sharp increase in metagenomics projects over the past decade. These projects allow us to better understand the diversity and function of microbial communities found in the environment, including the ocean, Arctic regions, other living organisms, and the human body. We extend our likelihood-based framework and show that we can accurately compare assemblies of these complex bacterial communities.

After an assembly has been produced, it is not an easy task determining what parts of the underlying genome are missing, what parts are mistakes, and what parts are due to experimental artifacts from the sequencing machine. Here we introduce VALET, the first reference-free pipeline that flags regions in metagenomic assemblies that are statistically inconsistent with the data generation process. VALET detects mis-assemblies in publicly available datasets and highlights the current shortcomings in available metagenomic assemblers.

By providing the computational methods for researchers to accurately evaluate their assemblies, we decrease the chance of incorrect biological conclusions and misguided future studies.

NOVEL METHODS FOR COMPARING AND EVALUATING SINGLE AND METAGENOMIC ASSEMBLIES

by

Christopher Michael Hill

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Advisory Committee:
Professor Mihai Pop, Chair/Advisor
Professor Atif Memon
Professor Héctor Corrada Bravo
Professor Michael Cummings
Professor Stephen Mount, Dean's Representative

Preface

The algorithms, software, and results in this dissertation have either been published in peer-reviewed journals and conferences or are currently under preparation for submission. At the time of this writing, Chapters 2, 3, 4, and 6 have already been published or submitted and are reformatted here. Chapter 5 is under preparation for submission. I am indebted to my co-authors on these projects - their dedication and knowledge in the areas of computer science, statistics, and biology have resulted in much stronger scientific papers.

Chapter 2:

- Mohammadreza Ghodsi*, Christopher M. Hill*, Irina Astrovskaya, Henry Lin, Dan D. Sommer, Sergey Koren, and Mihai Pop. *De novo likelihood-based measures for comparing genome assemblies*. BMC research notes 6, no. 1 (2013): 334.

My contributions to this work include: (1) aiding in the development of the underlying theory, specifically to the development of the theory incorporating paired-read information, (2) implementation of the alignment-based and sampling methods, (3) producing all of the results, and (4) aiding in drafting the manuscript.

The authors would like to thank Héctor Corrada Bravo and Bo Liu for their advice on the sampling procedure and associated statistics, Todd Treangen for advice on accessing the GAGE data, and the other members of the Pop lab for valuable discussions on all aspects of our work. This work was supported in part by the National Science Foundation (grants IIS-0812111, IIS-1117247 to MP), and by the

National Institutes of Health (grant R01-HG-004885 to MP).

Chapter 3:

- Christopher M. Hill, Irina Astrovskaya, Heng Huang, Sergey Koren, Atif Memon, Todd J. Treangen, and Mihai Pop. *De novo likelihood-based measures for comparing metagenomic assemblies.* In Bioinformatics and Biomedicine (BIBM), 2013 IEEE International Conference on, pp. 94-98. IEEE, 2013.

My contributions to this work include: (1) developing the theory to handle the addition of organismal abundances, (2) drafting the manuscript, and (3) producing the majority of the results.

The authors would like to thank the members of the Pop lab for valuable discussions on all aspects of our work. This work was supported in part by the NIH, grant R01-AI-100947 to MP, and the NSF, grant IIS-1117247 to MP.

- Koren, Sergey, Todd J. Treangen, Christopher M. Hill, Mihai Pop, and Adam M. Phillippy. *Automated ensemble assembly and validation of microbial genomes.* BMC bioinformatics 15, no. 1 (2014): 126.

My contributions to this work include: (1) the development of the LAP software used by MetAMOS.

The authors would like to thank Magoc et al. and Comas et al. who submitted the raw data that was used in this study. We thank Lex Nederbragt and an anonymous reviewer for detailed comments on the manuscript and iMetAMOS software, usability, and documentation. MP and CMH were supported by NIH grant

R01-AI-100947and the NSF grant IIS-1117247.

Chapter 4:

- Christopher M. Hill, Sergey Koren, Daniel Sommer, Bryan Dzung Ta, Atif Memon. and Mihai Pop. *De novo genome assembly regression testing.* Under revision.

My contributions to this work include: (1) generating the software used by the assembler regression pipeline, (2) evaluating SOAPdenovo2's version history, (3) evaluating the effect of read permutation and multiple threads on assembly quality, and (4) drafting the manuscript.

The authors would like to thank members of the Memon and Pop labs for their support and valuable discussions. This work was supported by NIH grant R01-AI-100947 to MP.

Chapter 5:

- Christopher M. Hill, Jonathan Gluck, Atif Memon, and Mihai Pop. *VALET: a de novo pipeline for finding metagenomic mis-assemblies.* In preparation.

My contributions to this work include: (1) aiding in the development and theory of the overall pipeline, (2) developing the tool used for finding highly variable coverage regions, (3) producing all results, and (4) drafting the manuscript.

Chapter 6:

- Christopher M. Hill, Carl H. Albach, Sebastian G. Angel, and Mihai Pop. *K-mulus: Strategies for BLAST in the Cloud.* In Parallel Processing and Applied

Mathematics, pp. 237-246. Springer Berlin Heidelberg, 2014.

My contributions to this work include: (1) implementing the query segmentation, database segmentation, and hybrid approaches, (2) producing the majority of the results, and (3) drafting the manuscript.

The authors would like to thank Mohammadreza Ghodsi for advice on clustering, Daniel Sommer for advice on Hadoop, Lee Mendelowitz for manuscript feedback, Katherine Fenstermacher for the name K-mulus, and the other members of the Pop lab for valuable discussions on all aspects of our work.

- Christopher M. Hill, Andras Szolek, Mohamed El Hadidi, and Michael Cummings. *Lossy compression of DNA sequencing quality data.* Under review.

My contributions to this work include: (1) developing the overall pipeline, (2) implementing the regression, profile, and binning approaches, and (3) aiding in the drafting of the manuscript.

The authors would like to thank the 2014 Bioinformatics Exchange for Students and Teachers (BEST) Summer School, funded by the offices of the Dean of The Graduate School at University of Maryland and the Rektor of University of Tübingen, where this research was initiated.

Dedication

To those who inspired it and will not read it.

Acknowledgments

I have been truly fortunate to know and work with many outstanding people throughout my career, and I am grateful to all of you.

First and foremost, I would like to thank my advisor Mihai Pop. It has been an absolute privilege to be under your guidance for the past 8 years. I hope that you will continue to watch over my academic career and be proud. I am eternally grateful for your advice, support, and encouragement and hope to have the opportunity to pay it forward someday.

I would also like to thank all of my other committee members. I will always enjoy the [often off-topic] discussions with Atif Memon who has introduced me to the world of software testing. I am indebted to Héctor Corrada Bravo for increasing the focus of statistics within the Center for Bioinformatics and Computational Biology. I wish to thank Michael Cummings for his mentorship and allowing me to work abroad with him. I have built up collaborations and friendships that were made possible thanks to him. I am also very grateful to Stephen Mount for always being able to answer any biological question I come up with during our social hours.

I owe a great deal of thanks to countless past and present faculty and graduate students at the University of Maryland. In particular, the majority of this dissertation was only made possible with the help of MohammadReza Ghodsi, Irina Astrovskaya, and Henry Lin. In addition, I owe a great deal of thanks to Sergey Koren who has been my mentor and oracle for all things related to assembly. Ted Gibbons has been a close friend and would always listen to me rant about the

seemingly endless negative results and lack of progress. I would also like to thank everyone (past and present) from the labs of Mihai Pop, Héctor Corrada Bravo, Sridhar Hannenhalli, and Steven Salzberg.

Finally, I must thank my friends, family, and loved ones who have supported me unconditionally throughout this adventure.

It is impossible to remember everyone, so I apologize to anyone who may have been left out. If I forgot you, let me know and I will happily buy you a drink.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Genome assembly

The genome sequence of an organism is the blueprint for building that organism. It is a key resource that allows researchers to better understand the organism's function and evolution. Initially published in 2001, the human genome has undergone dozens of revisions over the years [1]. Researchers fill in gaps, and correct mistakes in the sequence. It is not an easy task determining what parts of the genome are missing, what parts are mistakes, and what parts are due to experimental artifacts from the sequencing machine. Obtaining the genome of any organism is difficult as modern sequencing technologies can only "read" small stretches (under a few thousand of basepairs/characters in length) of the genome (called *reads*). During the later years of the human genome project, the proposal that these tiny reads could be pieced together to reconstruct the human genome (3.2 billion basepairs) was the subject of vigorous scientific debate [2, 3]. The development of algorithms and computational tools called *genome assemblers* able to reconstruct near-complete genome sequences from the reads produced by sequencing machines played a pivotal role in the modern genomic revolution.

Despite tremendous advances made over the past 30 years in both sequencing

technologies and assembly algorithms, genome assembly remains a highly difficult computational problem. In all but the simplest cases, genome assemblers cannot fully and correctly reconstruct an organism's genome. Instead, the output of an assembler consists of a set of contiguous sequence fragments (*contigs*), which can be further ordered and oriented into *scaffolds*, representing the relative placement of the contigs, with possible intervening gaps, along the genome.

1.1.1 Computational challenges of assembly

The genome assembly problem is often formulated as either a Hamiltonian or an Eulerian path problem depending on how the reads and overlaps between reads are represented [4]. In the overlap-layout-concensus (OLC) paradigm, reads are represented as nodes in the graph with edges connecting reads that overlap. The assembler seeks to reconstruct a path through the graph that contains all nodes, i.e., a Hamiltonian path. In de Bruijn graph-based assemblers, complete reads are not necessarily represented as nodes in the graph. Instead, the reads are broken up into overlapping strings of length k. Each k-length substring (k-mer) is represented as an edge in the graph connecting the nodes corresponding to the $k-1$-length prefix and suffix of the k-mer. In this case, the assembler seeks to reconstruct a path in the graph that uses all edges, i.e., an Eulerian path.

Theoretical analyses of the assembly problem have shown that assembly is NP-hard [5,6], i.e., finding the correct optimal solution may require an exhaustive search of an exponential number of possible solutions. The presence of repeated DNA

segments (*repeats*) exacerbates the difficulty of genome assembly. Repeats longer than the length of the sequenced reads lead to ambiguity in the reconstruction of the genome – many different genomes can be built from the same set of reads [7, 8].

As a result, practical implementations of assembly algorithms (such as ABySS [9], Velvet [10], SOAPdenovo [11], etc.) are forced to make tradeoffs between correctness, speed, and memory. Although in most cases, it is common for the output of assemblers to either contains errors, or be fragmented, or both.

Ideally, in a genome project, the assembly would be followed by the scrupulous manual curation of the assembled sequence to correct the hundreds to thousands of errors [12], and fill in the gaps between the assembled contigs [13]. Despite the value of fully completed and verified genome sequences [14], the substantial effort and associated cost necessary to conduct a finishing experiment to its conclusion can only be justified for a few high-priority genomes (such as reference strains or model organisms). The majority of the genomes sequenced today are automatically reconstructed in a "draft" state. Despite the fact that valuable biological conclusions can be derived from draft sequences [15], these genomes are of uncertain quality [16], possibly impacting the conclusions of analyses and experiments that rely on their primary sequence.

1.1.2 Assessing the quality of an assembly

Assessing the quality of the sequence output by an assembler is of critical importance, not just to inform downstream analyses, but also to allow researchers to

choose from among a rapidly increasing collection of genome assemblers. Currently, there are two ways to evaluate assemblies: reference-based and *de novo* evaluation. When a reference genome is available, an assembly's quality can be estimated based on the percentage of its true genome reconstruction, number of incorrect bases, structural errors, and additional biologically relevant information, such as the percent of genes reconstructed. *De novo* evaluation relies on assessing an assembly's quality based on the sequencing data alone. *De novo* metrics include global "sanity checks" (such as gene density, expected to be high in bacterial genomes, measured, for example, through the fraction of the assembled sequence that can be recognized by PFAM profiles [17]) and internal consistency measures [18] that evaluate the placement of reads and mate-pairs along the assembled sequence.

Despite incremental improvements in the performance of genome assemblers, **none** of the software tools available today outperforms the rest in all assembly tasks. As highlighted by recent high profile assembly bake-offs [19,20], different assemblers "win the race" depending on the specific characteristics of the sequencing data, the structure of the genome being assembled, or the specific needs of the downstream analysis process. Furthermore, these competitions have highlighted the inherent difficulty of assessing the quality of an assembly - all assemblers attempt to find a trade-off between contiguity (the size of the contigs generated) and accuracy of the resulting sequence. Even with the availability of a gold standard, evaluating this trade-off is difficult. In most practical settings, a reference genome sequence is not available, and the validation process must rely on other, often costly, sources of information, such as independently derived data from mapping experiments [21], or

from transcriptome sequencing [22]. Most commonly, validation relies on *de novo* approaches based on the sequencing data alone. The validation approaches outlined above can highlight a number of inconsistencies or errors in the assembled sequence, information valuable as a guide for further validation and refinement experiments, but difficult to use in a comparative setting where the goal is to compare the quality of multiple assemblies of a same dataset. For example, given a reference genome sequence, it is unclear how to weigh single nucleotide differences and short indels against much larger structural errors (e.g., translocation or large scale copy-number changes) [19] when comparing different assemblies. Furthermore, while recent advances in visualization techniques, such as the FRCurve of Narzisi et al. [23, 24], have made it easier for scientists to appropriately visualize the overall tradeoff between assembly contiguity and correctness, there exist no established approaches that allow one to appropriately weigh the relative importance of the multitude of assembly quality measures, many of which provide redundant information [24].

1.2 Contributions of this dissertation

In Chapter 2, we present our LAP framework, an objective and holistic approach for evaluating and comparing the quality of assemblies derived from a same dataset. Our approach defines the quality of an assembly as the likelihood that the observed reads are generated from the given assembly, a value which can be accurately estimated by appropriately modeling the sequencing process. We show that our approach is able to automatically and accurately reproduce the reference-

based ranking of assembly tools produced by highly-cited assembly competitions: the Assemblathon [19] and GAGE [20] competitions.

In Chapter 3, we extend our *de novo* LAP framework to evaluate metagenomic assemblies. We will show that by modifying our likelihood calculation to take into account abundances of assembled sequences, we can accurately and efficiently compare metagenomic assemblies. We find that our extended LAP framework is able to reproduce results on data from the Human Microbiome Project (HMP) [25, 26] that closely match the reference-based evaluation metrics and outperforms other *de novo* metrics traditionally used to measure assembly quality. Finally, we have integrated our LAP framework into the metagenomic analysis pipeline MetAMOS [27], allowing any user to reproduce quality assembly evaluations with relative ease.

In Chapter 4, we provide a novel regression testing framework for genome assemblers. Our framework that uses two assembly evaluation mechanisms: *assembly likelihood*, calculated using our LAP framework [28], and *read-pair coverage*, calculated using REAPR [29], to determine if code modifications result in non-trivial changes in assembly quality. We study assembler evolution in two contexts. First, we examine how assembly quality changes throughout the version history of the popular assembler SOAPdenovo [30]. Second, we show that our framework can correctly evaluate decrease in assembly quality using fault-seeded versions of another assembler Minimus [31]. Our results show that our framework accurately detects trivial changes in assembly quality produced from permuted input reads and using multi-core systems, which fail to be detected using traditional regression testing methods.

In Chapter 5, we build on the pipeline described in Chapter 4 and introduce VALET, a *de novo* pipeline for finding misassemblies within metagenomic assemblies. We flag regions of the genome that are statistically inconsistent with the data generation process and underlying species abundances. VALET is the first tool to accurately and efficiently find misassemblies in metagenomic datasets. We run VALET on publicly available datasets and use the findings to suggest improvements for future metagenomic assemblers.

In Chapter 6, we discuss our other contributions to bioinformatics relating to the domains of clustering, compression, and cloud computing.

Chapter 2: Comparing Whole-Genome Assemblies

2.1 Introduction

Here we propose an objective and holistic approach for evaluating and comparing the quality of assemblies derived from a same dataset. Our approach defines the quality of an assembly as the likelihood that the observed reads are generated from the given assembly, a value which can be accurately estimated by appropriately modeling the sequencing process. This basic idea was formulated in the 1990's in the pioneering work of Gene Myers [5], where he suggested the correct assembly of a set of reads must be consistent (in terms of the Kolmogorov-Smirnoff test statistic) with the statistical characteristics of the data generation process. The same basic idea was further used in the arrival-rate statistic (A-statistic) in Celera assembler [32] to identify collapsed repeats, and as an objective function in quasi-species (ShoRAH [33], ViSpA [34]), metagenomic (Genovo [17]), general-purpose assemblers [35], and recent assembly evaluation frameworks (ALE [36], CGAL [37]).

In this chapter, we will describe in detail a mathematical model of the sequencing process that takes into account sequencing errors and mate-pair information, and show how this model can be computed in practice. We will also show that this *de novo* probabilistic framework is able to automatically and accurately reproduce the

reference-based ranking of assembly tools produced by the Assemblathon [19] and GAGE [20] competitions. Our work is similar in spirit to the recently published ALE [36] and CGAL [37]; however, we provide here several extensions of practical importance. First, we propose and evaluate a sampling-based protocol for computing the assembly score which allows the rapid approximation of assembly quality, enabling the application of our methods to large datasets. Second, we evaluate the effect of unassembled reads and contaminant DNA on the relative ranking of assemblies according to the likelihood score. Finally, we will demonstrate the use of our probabilistic quality measure as an objective function in optimizing the parameters of assembly programs. The software implementing our approach is made available, open-source and free of charge, at: `http://assembly-eval.sourceforge.net/`.

2.2 Methods

2.2.1 Theoretical foundation for probabilistic evaluation

In this section, we formalize the probabilistic formulation of assembly quality and the model of the sequencing process that allows us to compute the likelihood of any particular assembly of a set of reads. We will show that the proposed probabilistic score is correct in the sense that the score is maximized by the true genome sequence.

2.2.1.1 Likelihood of an assembly

Let A denote the event that a given assembly is the true genome sequence, and let R denote the event of observing a given set of reads. In the following, we will use the same symbol to denote the assembly sequence and the event of observing the assembly. We will also use the same symbol to denote the set of reads and the event of observing the set of reads.

According to Bayes' rule, given the observed set of reads, the probability of the assembly can be written as:

$$\Pr[A|R] = \frac{\Pr[R|A]\Pr[A]}{\Pr[R]} \tag{2.1}$$

where $\Pr[A]$ is the *prior probability* of observing the genome sequence A. Any prior knowledge about the genome being assembled (e.g., approximate length, presence of certain genes, etc.) can be included in $\Pr[A]$; however, for the purpose of this paper, we will assume that this prior probability is constant across the set of "reasonable" assemblies of a same set of reads. Given commonly available information about the genomes, formulating a precise mathematical framework for defining $\Pr[A]$ is an extensive endeavor beyond the scope of this paper.

Similarly, $\Pr[R]$ is the prior probability of observing the set of reads R. Since our primary goal is to compare multiple assemblies of a same set of reads, rather than to obtain a universally accurate measure of assembly quality, we can assume $\Pr[R]$ is a constant as well. Thus, for the purpose of comparing assemblies, the values $\Pr[A|R]$ and $\Pr[R|A]$ are equivalent. The latter, the posterior probability of

a set of reads, given a particular assembly of the data, can be easily computed on the basis of an appropriately defined model of the sequencing process and will be used in our paper as a proxy for assembly quality.

Under the assumption that individual reads are independent of each other (violations of this assumptions in the case of mate-pair experiments will be discussed later in this section), $\Pr[R|A] = \prod_{r \in R} \Pr[r|A]$. If the set of reads is unordered, we need to account for the different permutations that generate the same set of reads. As this value is a constant for any given set of reads, we ignore it in the rest of our paper.

$\Pr[r|A]$, hereafter referred to as p_r, can be computed using an appropriate model for the sequencing process. Throughout the remainder of the paper, we will discuss increasingly complex models and their impact on the accuracy of the likelihood score.

2.2.1.2 True genome obtains the maximum likelihood

Any useful assembly quality metric must achieve its maximum value when evaluating the true genome sequence; otherwise, incorrect assemblies of the data would be preferred. We prove below that the likelihood measure proposed in our paper satisfies this property.

Assuming that we have a set of reads R from the true genome, produced by generating exactly one single-end read from each location in the genome without errors and with a fixed length. Given the set of reads R, the probability a particular

read is generated from the true genome is precisely the number of times the read occurs in R divided by the size of R (note that multiple reads can have the same sequence, e.g., when generated from repeats). Let N_s denote number of times that the sequence s occurs in R, and $q_s = N_s/|R|$ denote the probability that sequence s is generated from the true genome. To show that the true genome maximizes the likelihood score, let us assume that we have some assembly A and p_s is the probability that the sequence s is generated from the assembly A.

Given assembly A, our likelihood score is then the product of ${p_s}^{N_s}$ over all sequences s in S, which can be rewritten as $\prod_{s \in S} {p_s}^{q_s|R|} = (\prod_{s \in S} {p_s}^{q_s})^{|R|}$. Now, note that since $|R|$ is a fixed constant, maximizing the likelihood score is equivalent to maximizing

$$\prod_{s \in S} {p_s}^{q_s}$$

The likelihood can be re-written as

$$\begin{aligned}
\log(\prod_{s \in S} {p_s}^{q_s}) &= \sum_{s \in S} q_s \log p_s \\
&= \sum_{s \in S} q_s \log(\frac{p_s}{q_s}) + \sum_{s \in S} q_s \log q_s \\
&= -D_{KL}(Q||P) - H(Q),
\end{aligned}$$

where $D_{KL}(Q||P)$ is the KL-divergence for the distributions Q and P, and $H(Q)$ is the Shannon entropy of Q. Since the KL-divergence is always non-negative and only equal to 0 if and only if $Q = P$, the average probability is maximized if the assembly is equal to the true genome.

Even though the true genome does maximize the likelihood in this model,

there may be other assemblies that achieve the same optimal score as long as these assemblies yield probabilities p_s which are equal to the probabilities q_s for every sequence s. This can happen, for example, in the case of a misassembly that is nonetheless consistent with the generated reads. This situation highlights the loss of information inherent in modern sequencing experiments – without additional long-range information, the information provided by the reads themselves is insufficient to distinguish between multiple possible reconstructions of a genome [8].

2.2.1.3 Error-free model for fragment sequencing

The most basic model for the sequencing process is the *error-free model*. In this model, we assume reads of a given fixed length (a more general read length distribution can be included in the model but would not impact comparative analyses of assemblies derived from a same set of reads). We further assume that reads are uniformly sampled across the genome, i.e., that every position of the genome is equally likely to be a starting point for a read. This simplifying assumption is made by virtually all other theoretical models of genome assembly, despite the biases inherent to all modern sequencing technologies. A more accurate, technology-dependent, model can be obtained by including additional factors that account, for example, for DNA composition biases. For the purpose of generality, we restrict our discussion to the uniform sampling model. Furthermore, for the sake of simplicity, we assume (1) that the true genome consists of a single circular contiguous sequence, (2) that our assembly is also a single contig, and (3) that every read can

be mapped to the assembly. We will later discuss extensions of our model that relax these assumptions.

Under these assumptions, we can compute the probability of a read r given the assembled sequence as:

$$p_r = \frac{n_r}{2L} \tag{2.2}$$

where n_r represents the number of places where the read occurs in the assembled sequence of length L. The factor 2 is due to the fact that reads are sampled with equal likelihood from both the forward and reverse strands of a DNA molecule. This formulation was previously used by Medvedev *et al.* [35] to define an objective function for genome assembly.

2.2.2 A realistic model of the sequencing process

The error-free model outlined above makes many simplifying assumptions that are not representative of real datasets. Here we demonstrate how the model can be extended to account for artifacts such as sequencing errors, mate-pair information, assemblies consisting of multiple contigs, and the presence of un-mappable reads.

2.2.2.1 Sequencing errors

All current technologies for sequencing DNA have a small but significant probability of error. Here we focus on three common types of errors: the insertion, deletion, and substitution of a nucleotide.

In the error-free model, the probability of a read having been generated from

a position j in the sequence is one if the read exactly matches the reference at that position and zero otherwise. We now extend this model such that the probability of each read having been generated from any position j of the reference is a real value between zero and one, representing the likelihood that a sequencing instrument would have generated that specific read from that specific position of the reference. This value clearly depends on the number of differences between the sequence of the read and the sequence of the reference at position j. Given the assembled sequence, the probability of a particular read will be the cumulative probability of the read across all possible locations in the genome.

Specifically, let us denote the probability that read r is observed by sequencing the reference, *ending* at position j by $p_{r,j}$. Then, the total probability of the read r is

$$p_r = \frac{\sum_{j=1}^{L} p_{r,j}^{\text{forward}} + \sum_{j=1}^{L} p_{r,j}^{\text{reverse}}}{2L} \tag{2.3}$$

The individual probabilities $p_{r,j}$ can be computed if we do not model insertion and deletion errors and only allow substitution errors which occur with probability ϵ. The per-base probability of a substitution error can be set individually for each based on the quality value produced by the sequencing instrument. Then, $p_{r,j} = \epsilon^s(1-\epsilon)^{l-s}$, where s is the number of substitutions needed to match read r to position j of the reference sequence. In the more general case, $p_{r,j}$ values can be computed using dynamic programming.

2.2.2.2 Exact probability calculation via dynamic programming

For a model of the sequencing process that allows insertions, deletions, and substitutions with specific probabilities, we can exactly compute probability, $p_r = \Pr[r|A]$, of observing a read r given an assembly A using a dynamic programming algorithm. In general, we want to find the sum of the probabilities of all possible alignments of a read to a position of the assembly.

```
ACCG    ACCG
A-CG    AC-G
```

Figure 2.1: Two different optimal alignments of the read **ACG** to the assembly **ACCG**. Our dynamic programming algorithm finds the sum of the probabilities of all possible alignments.

The number of such possible alignments grows exponentially as a function of read length. Most of those alignments have a very small probability. However, several alignments may have probabilities that are equal or close to the optimal. For example, the two alignments of the same pair of sequences in Figure 2.1 have the same probability and are both optimal alignments.

We use a dynamic programming algorithm (similar to the "forward" algorithm in Hidden Markov Models) to efficiently calculate the sum of the probabilities of all alignments of a read to the assembly as follows. In the formula (2.3), $p_{r,j}^{\text{forward}}$ and $p_{r,j}^{\text{reverse}}$ are the sum of the probabilities of *all* possible alignments of the read r to, respectively, the reference and its reverse complement, ending at position j.

We define $T[x, y]$ as the probability of observing prefix $[1 \ldots y]$ of the read

r, if y bases are sequenced from the reference, ending at position x. Therefore, $p_{r,j} = T[j, l]$. $T[x, 0]$ represents the probability of observing an empty sequence if we sequence zero bases and is set to 1. $T[0, y]$ represents the probability of observing prefix $[1 \dots y]$ of the read if y bases are sequenced from the reference, ending at position 0 (before the beginning), and is set to 0.

For $x \geq 1$ and $y \geq 1$, $T[x, y]$ is recursively defined:

$$\begin{aligned} T[x,y] = \quad & T[x-1, y-1] \Pr[\text{Substitute}(A[x], r[y])] \\ & + T[x, y-1] \Pr[\text{Insert}(r[y])] \\ & + T[x-1, y] \Pr[\text{Delete}(A[x])], \end{aligned} \tag{2.4}$$

where $r[y]$ and $A[x]$ represent the nucleotides at positions y and x of the read r and the assembly A, respectively. $\Pr[\text{Substitute}(A[x], r[y])]$ is the probability of observing the nucleotide $r[y]$ by sequencing the nucleotide $A[x]$. In our experiments, we did not distinguish between different types of errors and considered their probability to be ϵ and the probability of observing the correct nucleotide to be $1 - \epsilon$.

The dynamic programming algorithm outlined above has a running time of $O(lL)$ per read. Even though the running time is polynomial, it is slow in practice. However, we can speed it up by using alignment seeds. The seeds would give us the regions of the assembly where a read may align with high probability. We can apply the dynamic programming only to those regions and get a very good approximate value of the total probability. We use exact seeds (k-mers) of a given length to build a hash index of the assembly sequence. Then, each read is compared to the regions

where it has a common k-mer with the assembly sequence.

2.2.2.3 Mate pairs

Many of the current sequencing technologies produce paired reads – reads generated from the opposite ends of the same DNA fragment. This information is extremely valuable in resolving genomic repeats and in ordering the contigs along long-range scaffolds; however, the paired reads violate the assumption that reads are sampled independently, that we made in the discussion above. To address this issue, we can use the pairs rather than the individual reads as the underlying objects from which the assembly likelihood is computed. To address the possibility that assembly errors may result in violations of the constraints imposed by the paired reads, we only consider pairs for which both ends align to a same contig or scaffold within the constraints imposed by the parameters of the sequencing experiment. Any pairs that violate these constraints get classified as unassembled. Note that in addition to sequencing errors, we now also handle fragment sizing errors – deviations of the estimated distance between paired reads from the distance implied by the sequencing experiment. We model the distribution of fragment sizes within a same library by a normal distribution, using user-supplied parameters, and use this information to appropriately scale the likelihood estimate for each possible placement of a mate pair along the genome.

We modify the dynamic programming recurrence from formula (2.4) to handle the probability calculation for the paired reads as follows. The probability of the

first read in the pair is calculated as the same as in the formula (2.4). For the second read, we adjust the dynamic programming to ensure that it is aligned within a certain distance downstream of the alignment of the first read. We modify the first column of the dynamic programming table of the *second* read in the pair to take into account the distance from the first read.

Formally, given a paired read, we define $T_2[x, y]$ as the probability of observing prefix $[1 \dots y]$ of the 2nd read in the pair, if y bases are sequenced from the reference, ending at position x. Assume that the second read occurs after the first read and is separated by a normally-distributed distance with mean μ and with a standard deviation σ. Therefore,

$$T_2[x, 0] = \sum_{i=1}^{x} \Pr[\text{insert}(x - i)|N(\mu, \sigma)))] + T_1[x - i, l], \tag{2.5}$$

where $\Pr[\text{insert}(n)|N(\mu, \sigma)))]$ is the probability of observing an insert size of length n from a normal distribution with parameters μ and σ, and l is the length of the first read in the pair.

Instead of using two tables, we can concatenate the read pair together with a special character (M), which will signal when the insert size should be taken into account.

For $x \geq 1$ and $y \geq 1$, $T[x, y]$ is recursively defined as follows:

$$T[x,y] = \quad \text{if } r[y] == M \begin{cases} \sum_{i=1}^{x} \Pr[\text{insert}(x-i)|N(\mu,\sigma))] + T[x-i, y-1] \\ \text{else} \begin{cases} T[x-1,y-1] \Pr[\text{Substitute}(A[x], r[y])] \\ \quad +T[x,y-1] \Pr[\text{Insert}(r[y])] \\ \quad +T[x-1,y] \Pr[\text{Delete}(A[x])] \end{cases} \end{cases} \tag{2.6}$$

2.2.2.4 Assemblies containing more than one contig

As we mentioned in the introduction, the output of an assembler usually consists of a (large) set of contigs rather than one single contig, representing the genome being assembled. In the extreme case, an "assembler" may return the set of unassembled input reads (or the set of all k-mers in De Bruijn-based assemblers) as its output. Our likelihood score must be modified to account for such fragmented assemblies.

In practice, most assemblers join contigs only if they overlap by more than a certain number of bases; however, we only consider the case where contigs are non-overlapping substrings of the true genome. In this case, the length of the original genome must be *at least* the sum of the lengths of the contigs, that is, $\sum L_j$, where L_j is the length of the jth contig. Therefore, the probability of every read is at most:

$$\frac{n_r}{2 \sum L_j} \tag{2.7}$$

Overlapping contigs can be handled by reducing the length of the contigs by a value representing the minimum overlap required by the assembler, as performed, for example, in Genovo [17].

2.2.2.5 Reads that do not align well

In practice, popular assemblers do not incorporate every read in the assembly. Possible reasons include assembly errors (such as collapsed tandem repeats), reads with high error rates, or contamination in the DNA sample. These "singleton" or "chaff" reads cannot be modeled by our likelihood approach as the likelihood of any assembly that does not incorporate every read is zero. When sequencing errors are modeled, every read obtains a non-zero likelihood, even if it does not align to the assembly. Since, in general, a non-trivial fraction of the total set of the reads cannot be mapped to the assembly, by their sheer number, the singleton reads would dominate the probability calculation.

To account for this factor, we argue that for any read that does not align well, the overall probability of the assembly should not be lower than the probability of the same assembly when the missing read is appended to its sequence as a separate contig. The effect of such an addition on the overall probability can be calculated as follows. First, the probability of observing this read exactly, $\left(\frac{\Pr[\text{exact match}]}{2L}\right)$, is multiplied to the product of the probabilities of all mapped reads. Second, the probabilities of the mapped reads are decreased slightly due to the increase in the length of the assembled sequence.

For simplicity, let us assume an error-free model where each read maps to exactly one position on the assembled sequence. Let k denote the number of the original reads. The ratio between the new probability for all original reads divided by their probability before adding the new read is:

$$\frac{\frac{1}{(L+l)^k}}{\frac{1}{L^k}} = \left(\frac{L}{L+l}\right)^k = \left(1 - \frac{l}{L+l}\right)^k \approx e^{-\frac{lk}{L}}$$

Therefore, if the probability of observing a read is less than

$$\frac{\Pr[\text{exact match}]}{2L} e^{-\frac{l|R|}{L}}, \tag{2.8}$$

we consider this read as "unmapped" and use formula (2.8) as its probability. The probability of an exact match Pr[exact match] is approximated by $(1-\epsilon)^l$, where ϵ is the probability of an error (a mismatch, an insertion, or a deletion).

2.2.3 Performance considerations

2.2.3.1 Estimating the average read likelihood by sampling

Depending on the specific characteristics of the chosen sequencing model, the computation of the probability $\Pr[R|A]$ can be expensive for the dataset sizes commonly encountered in current projects (tens to hundreds of millions of reads). In such cases, we can approximate the likelihood of an assembly by using a random subset of the reads $R' \subseteq R$. To counteract the effect of the size of the sample on the computed probability, we define the assembly quality as the geometric mean of the individual read probabilities:

$$\text{AP}(R') = \left(\prod_{r \in R'} p_r\right)^{\frac{1}{|R'|}} \tag{2.9}$$

The logarithm of this value (Log Average Probability (LAP)) is reported in the remainder of the paper as the assembly quality "score":

$$\mathrm{LAP}(R') = \log_{10}(\mathrm{AP}(R')) = \frac{\sum_{r \in R'} \log_{10} p_r}{|R'|} \tag{2.10}$$

In other words, we define the assembly quality as the average log likelihood of the reads given an assembly. This formulation also allows us to estimate the accuracy of the approximate likelihood value produced by sub-sampling the set of reads. According to sampling theory, the distribution of the scores across multiple samples has the mean equal to the true likelihood of the assembly (computed from all the reads) and a standard error proportional to $\frac{1}{\sqrt{|R'|}}$, i.e., the larger the sample is, the more accurate our estimation is for the likelihood of the true assembly. Since the probability of a read is bounded by formula (2.8), the variance of the sample can also be bounded by this value.

In practice, we increase the sample size until the assemblies can be unambiguously distinguished by the LAP value. Specifically, we increase the sample size, by binary search, until the LAP values are separated by at least a single standard deviation. The level of subsampling required will, thus, be dependent on the extent of the differences between the assemblies - for very different assemblies, low levels of subsampling are sufficient.

2.2.3.2 Approximating the likelihood value using an aligner

Alternatively, when it is impractical to calculate exact probabilities for large sets of reads, we can approximate these probabilities using fast and memory-efficient

alignment search programs, which internally model the sequencing process. We use Bowtie 2 [38] to align the reads to the assembly. However, our programs are easy to adapt for any read alignment tool that stores the alignment results in SAM [39] format.

For each reported alignment, we use the number of substitutions s to compute the probability p_r. The probability of this alignment, $p_{r,j}$, can be approximated by $\epsilon^s(1-\epsilon)^{l-s}$ and

$$p_r = \frac{\sum_{j \in S_r} p_{r,j}}{2L}, \tag{2.11}$$

where S_r is the set of alignments in the SAM file for the read r.

We can further extend this equation to mated reads. A pair of mated reads aligns if the distance and orientation of the alignment of the pair are consistent with the experimental design parameters. Given read i_1 and its mate i_2, we compute $p_{(i_1,i_2)}$ by multiplying the probabilities of individually aligning each mate at their respective positions with the probability that they are separated by their distance from each other. That is,

$$p_{(i_1,i_2)} = \frac{\sum_{(j_1,j_2) \in S_{(i_1,i_2)}} p_{i_1,j_1} p_{i_2,j_2} \Pr[\text{insert}(j_2 - (j_1 + l_1))]}{2(L-l)}, \tag{2.12}$$

where $p_{i_1,j_1} = \epsilon^{s_1}(1-\epsilon)^{l_1-s_1}$. Mate pair insert sizes follow a normal distribution with mean and standard deviation being estimated from the parameters of the sequencing process. Unless otherwise stated, the standard deviation is 10% of the insert size. If only one of the mates, i_1 or i_2, maps, the probability $p_{(i_1,i_2)}$ is 0. We use (2.8) to set the probability for this case.

In our experiments, Bowtie 2 was used to approximate the read probabilities for the larger datasets; however, it could be substituted with any other aligner.

2.2.4 Datasets

The read data for *Rhodobacter sphaeroides* 2.4.1 was downloaded from `http://gage.cbcb.umd.edu/data/Rhodobacter_sphaeroides`, and the corresponding reference sequence was obtained from the NCBI RefSeq database (NC_007493.1, NC_007494.1, NC_009007.1, NC_007488.1, NC_007489.1, NC_007490.1, NC_009008.1). In addition, two more *Rhodobacter* genomes were selected as reference genomes, specifically *R. sphaeroides* ATCC 17025 (NCBI IDs NC_009428.1, NC_009429.1, NC_009430.1, NC_009431.1, NC_009432.1), and *R. capsulatus* SB1003 (NC_014034.1, NC_014035.1).

The read data for *Stapylococcus aureus* USA300 was downloaded from `http://http://gage.cbcb.umd.edu/data/Staphylococcus_aureus`, and the corresponding reference sequence was obtained from the NCBI RefSeq database (NC_010063.1, NC_010079.1, NC_012417.1). In addition, two more *Stapylococcus* genomes were selected as reference genomes, specifically *S. aureus* 04-02981 (CP001844), and *S. epidermidis* ATCC 12228 (AE015929, AE015930, AE015931, AE015932, AE015933, AE015934, AE015935).

The read data for human chromosome 14 was downloaded from `http://gage.cbcb.umd.edu/data/Hg_chr14/`, and the corresponding reference sequence was obtained from the NCBI RefSeq database (NC_000014.8).

The Assemblathon 1 competition evaluates assemblies on the simulated short

read dataset generated from the simulated 110 Mbp diploid genome. The competition provides sequence libraries with varying insert sizes (200-10,000 bp) and coverage (20-40x). Assemblathon 1 allowed teams to submit multiple entries, but for our analyses, we only examine the top ranking assemblies from each team. The raw reads and the consensus sequence of the top ranking assemblies were downloaded from `http://korflab.ucdavis.edu/Datasets/Assemblathon/Assemblathon1/`.

Also used in our analyses is the *E. coli* K12 MG1655 dataset, generated using Illumina MiSeq technology (300 bp insert size, 370x coverage) (`http://www.illumina.com/systems/miseq/scientific_data.ilmn`).

2.3 Results

2.3.1 Performance-related approximations do not significantly affect the likelihood score

The full and exact computation of the assembly likelihood score is computationally intensive and ultimately impractical for the analysis of large genomes sequenced with the next generation technologies. We have highlighted in the Methods section several approaches that can be used to reduce the computational requirements and allow the application of our methods in practical settings, including the computation of the likelihood score on the subsets of the original set of reads and the approximation of the score from the output of an alignment program. As we will show below, our approximations do not affect the comparative ranking of the multiple assemblies derived from a same dataset.

2.3.1.1 The likelihood score is robust under sampling.

To assess the effect of subsampling, we relied on a collection of the assemblies of the human chromosome 14 made available by the GAGE assembly 'bake-off'. We sampled random subsets of increasing size (one trial per size) from the over 60 million reads and computed the likelihood score based only on the sampled reads.

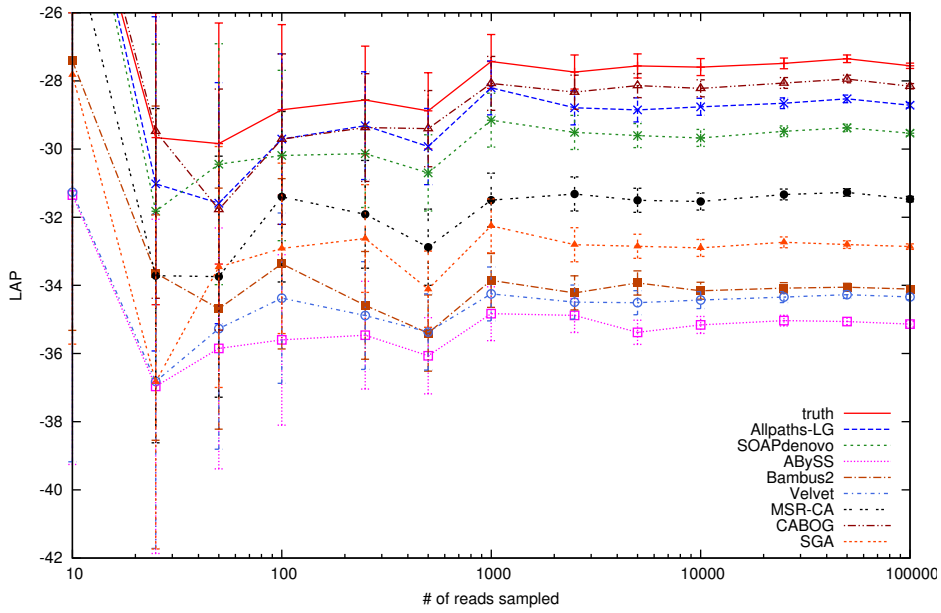


Figure 2.2: LAP-based evaluation of the assemblies for the Human chromosome 14 via sampling. The x-axis represents the number of sampled reads. For each assembly, we plot the corresponding LAP on a chosen subsample along with the standard deviation. The relative ranking of assemblies becomes fixed with 10,000 reads, which is less than 0.02% of the original reads.

As seen in Figure 2.2, the overall ranking of the different assemblies stabilizes after sampling just 10,000 reads, i.e., less than 0.02% of the entire dataset. After this point, the scores of individual assemblies differ by more than the standard

deviation of the sub-sampled scores, indicating the relative ranking of the assemblies can be determined with high statistical confidence. This result suggests a practical strategy for computing the assembly likelihood wherein datasets of increasing size are repeatedly sampled from the set of reads until the likelihood scores of the compared assemblies can be distinguished from each other. The search for the appropriate sample size can start from a reasonable 'guess' (e.g., 0.05% of the total set of reads), which is then iteratively doubled until the likelihood scores are separated from each other by a given multiple of the sampling-induced standard deviation.

2.3.1.2 Aligner-based approximation correlates with the dynamic-programming computation of the likelihood score.

As outlined in the Methods section, we relied on an alignment program (in our case, Bowtie 2 [38]) to estimate the likelihood of individual reads based on their alignment along the assembly. This approach is substantially faster than the more accurate dynamic programming algorithm that computes the cumulative likelihood of all possible alignments of a read against the assembly.

Figure 2.3 compares the per-read likelihood values with respect to the complete genome sequence of *Staphylococcus aureus*, using data provided by the GAGE competition. In this plot, each read is represented by a point whose coordinates represent the corresponding likelihood scores computed through full dynamic programming (y axis) and from Bowtie 2 alignments (x axis). As the full dynamic programming approach sums over all possible alignments, the corresponding likeli-

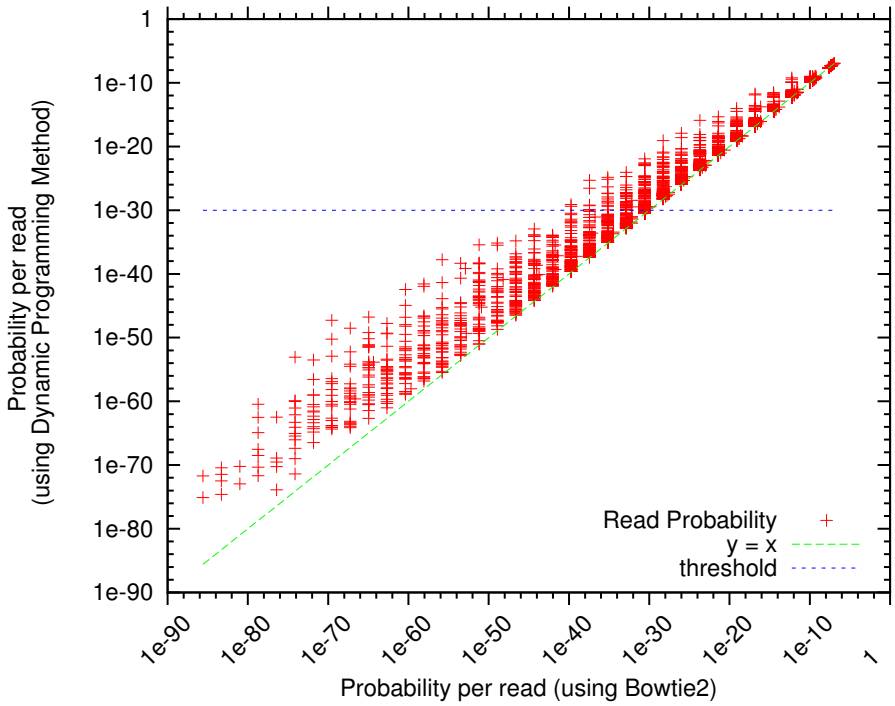


Figure 2.3: Comparison of the read probability calculation methods for *S. aureus* with 4,788,174 reads. Each mark on the plot represents a single read. The read's position is determined by the probability calculated from our dynamic programming method (y-axis) and Bowtie 2 (x-axis). Points on the line $y = x$ denote reads that were given the same probability by both methods. Since Bowtie 2 only finds the best alignment, it usually reports a slightly lower probability. A probability threshold of 1e-30 is shown for the dynamic programming method. The read probabilities that fall below this threshold would be rounded up to 1e-30 during LAP computation.

hood values are higher (points occur above the diagonal) than those estimated by Bowtie 2. The difference between the two methods becomes less noticeable as the likelihood increases as more of the probability mass is concentrated around the best alignment of a read to the reference.

2.3.1.3 The likelihood scores correlate with reference-based validation

The recent assembly competitions GAGE [20] and Assemblathon 1 [19] relied on a combination of *de novo* and reference-based metrics to compare and rank different assemblies. For the majority of these datasets, a complete or high-quality draft sequence was available, allowing the authors to objectively determine all the errors in the assemblies by aligning them to the reference sequences. Based on this information, the GAGE and Assemblathon 1 teams proposed several assembly quality metrics that simultaneously capture some aspects of the contiguity and correctness of an assembly. Here we compare our *de novo* likelihood score to these reference-based metrics.

Generally, the *de novo* LAP scores agree with the reference-corrected contiguity values (see Tables 2.1, 2.2, and 2.1). Furthermore, the reference genome assembly (assumed to be the most correct reconstruction of the genome being analyzed) achieves the highest LAP score while the references derived from the closely-related organisms are considerably worse than all the other assemblies. In other words, the *de novo* LAP scores accurately capture the relative quality of the different assemblies.

It is important to note that there are several exceptions to these general observations. In the case of *S. aureus* USA300 (Table 2), the read-based LAP scores for the Abyss assembly (computed on both contigs and scaffolds) are better than those obtained for the reference genome, contradicting our intuition, since ABySS's

reference-corrected contiguity is worse. This result highlights the importance of accurately modeling the sequencing experiment when computing the LAP scores. Once mate-pair information is taken into account, the LAP scores correctly identify the best assembly. This phenomenon is due to the fact that the Abyss assembly is able to incorporate more of the reads however their placement in the assembly is inconsistent with the mate-pair linkage information.

	Contigs				Scaffolds					
Assembler	LAP reads	LAP mates	N50 (kb)	CN50 (kb)	LAP reads	LAP mates	N50 (kb)	CN50 (kb)	Unaligned reads (frac)	Unaligned mates (frac)
ABySS	-20.924	-27.365	5.9	4.2	-20.929	-27.320	9	5	0.228	0.524
Allpaths-LG	**-20.795**	**-27.141**	42.5	**34.4**	**-20.796**	**-27.099**	**3,192**	**3,192**	**0.212**	**0.441**
Bambus2	-21.528	-27.439	93.2	12.8	-21.531	-27.424	2,439	2,419	0.270	0.501
CABOG	-22.550	-27.749	20.2	17.9	-22.550	-27.714	66	55	0.345	0.540
MSR-CA	-21.496	-27.407	22.1	19.1	-21.497	-27.324	2,976	2,966	0.268	0.478
SGA	-20.896	-27.575	4.5	2.9	-21.030	-27.416	51	51	0.237	0.541
SOAPdenovo	-20.816	-27.160	**131.7**	14.3	-20.816	-27.152	660	660	0.214	0.453
Velvet	-20.903	-27.314	15.7	14.5	-20.907	-27.246	353	270	0.219	0.471
R. sphaeroides ATCC 17025	-29.391	-29.973	3,218	3,218	-29.391	-29.973	3,218	3,218	0.813	0.904
R. capsulatus	-29.953	-29.997	3,739	3,739	-29.953	-29.997	3,739	3,739	0.978	0.995
truth	-20.769	-27.071	3,189	3,189	-20.769	-27.071	3,189	3,189	0.209	0.432

Table 2.1: Assembly likelihood scores for *Rhodobacter sphaeroides* 2.4.1 from the GAGE project [19]. The results are presented separately for the contigs and scaffolds and include the number of unassembled reads (singletons), the LAP scores computed on unmated reads (LAP reads) or mate-pairs (LAP mates), the N50 contig/scaffold sizes (N50), and the reference-corrected N50 contig/scaffold sizes (CN50). The best (maximum) value for each genome-measure combination is highlighted in bold. The results for the reference assembly (either complete genome or high-quality draft) is given in the row marked *truth*. In addition, we provide the results for a closely related strain and species. All values, except the LAP scores, were taken from the GAGE publication. A threshold probability of 1e-30 was used for calculating the LAP scores. The standard deviations for the LAP's reads and LAP's mates scores are 0.00685 and 0.00969, respectively.

	Contigs				Scaffolds					
Assembler	LAP reads	LAP mates	N50 (kb)	CN50 (kb)	LAP reads	LAP mates	N50 (kb)	CN50 (kb)	Unaligned reads (frac)	Unaligned mates (frac)
ABySS	**-16.608**	-24.692	29.2	24.8	**-16.611**	-24.584	34	28	**0.318**	0.522
Allpaths-LG	-18.018	-23.974	96.7	**66.2**	-18.018	**-23.760**	1,092	**1,092**	0.374	**0.494**
Bambus2	-18.083	-24.256	50.2	16.7	-18.085	-23.899	1,084	1,084	0.375	0.503
MSR-CA	-18.282	-24.258	59.2	48.2	-18.282	-23.926	**2,412**	1,022	0.389	0.508
SGA	-17.937	-27.019	4	4	-18.250	-24.906	208	208	0.384	0.578
SOAPdenovo	-17.830	**-23.892**	**288.2**	62.7	-17.830	-23.862	332	288	0.362	0.499
Velvet	-17.867	-24.258	48.4	41.5	-17.867	-23.925	762	126	0.363	0.503
S. aureus 04-02981	-19.960	-25.314	2,821	2,821	-19.960	-25.314	2,821	2,821	0.456	0.572
S. epidermidis	-29.635	-29.951	2,499	2,499	-29.635	-29.951	2,499	2,499	0.972	0.988
truth	-17.741	-23.509	2,873	2,873	-17.741	-23.509	2,873	2,873	0.358	0.473

Table 2.2: Assembly likelihood scores for *Staphylococcus aureus* USA300 from the GAGE project [19]. The results are presented separately for the contigs and scaffolds and include the number of unassembled reads (singletons), the LAP scores computed on unmated reads (LAP reads) or mate-pairs (LAP mates), the N50 contig/scaffold sizes (N50), and the reference-corrected N50 contig/scaffold sizes (CN50). The best (maximum) value for each genome-measure combination is highlighted in bold. The results for the reference assembly (either complete genome or high-quality draft) is given in the row marked *truth*. In addition, we provide the results for a closely related strain and species. All values, except the LAP scores, were taken from the GAGE publication. A threshold probability of 1e-30 was used for calculating the LAP scores. The standard deviations for the LAP's reads and LAP's mates scores are0.00740 and 0.0105, respectively.

	Contigs				Scaffolds						
Assembler	LAP reads	LAP mates	N50 (kb)	CN50 (kb)	LAP reads	LAP mates	N50 (kb)	CN50 (kb)	CGAL Score	Unaligned reads (frac)	Unaligned mates (frac)
ABySS	-18.473	-23.801	2	2	-18.474	-23.787	2.1	2	-15.21 x 10^8	0.257	0.504
Allpaths-LG	-15.813	-21.413	36.5	21	-15.824	-21.314	**81,647**	**4,702**	-13.11 x 10^8	0.115	0.239
Bambus2	-18.606	-23.474	5.9	4.3	-18.642	-23.343	324	161	-	0.258	0.422
CABOG	**-15.625**	**-21.128**	**45.3**	**23.7**	**-15.626**	**-21.041**	393	26	**-12.25 x 10^8**	0.109	**0.229**
MSR-CA	-16.421	-22.428	4.9	4.3	-16.436	-21.861	893	94	-	0.122	0.276
SGA	-15.712	-22.990	2.7	2.7	-16.909	-22.326	83	79	-	0.134	0.328
SOAPdenovo	-15.702	-21.705	14.7	7.4	-15.734	-21.594	455	214	*	**0.101**	0.269
Velvet	-18.000	-23.468	2.3	2.1	-18.140	-23.375	1,190	27	-	0.214	0.442
truth	-15.466	-21.001	107,349.50	107,349.50	-15.466	-21.002	107,349.50	107,349.50	-11.25 x 10^8	0.093	0.211

Table 2.3: Assembly likelihood scores for human chromosome 14 from the GAGE project [19] using a 10,000 read sample. The results are presented separately for the contigs and scaffolds and include the number of unassembled reads (singletons), the LAP scores computed on unmated reads (LAP reads) or mate-pairs (LAP mates), the N50 contig/scaffold sizes (N50), and the reference-corrected N50 contig/scaffold sizes (CN50). The best (maximum) value for each genome-measure combination is highlighted in bold. The results for the reference assembly (either complete genome or high-quality draft) is given in the row marked *truth*. In addition, we provide the results for a closely related strain and species. CGAL scores calculated from the long insert library were taken from the CGAL publication. The authors only provided scores for the top three assemblies (Bowtie2 could not successfully map reads to the SOAPdenovo assembly). All values, except the LAP and CGAL scores, were taken from the GAGE publication. A threshold probability of 1e-30 was used for calculating the LAP scores. The standard deviation for both the LAP's reads and LAP's mates scores is 0.15.

In the case of the human chromosome 14 assembly (Table 3), the scaffold-based results do not agree with the reference-corrected contiguity values: the CABOG assembler outperforms Allpaths-LG in all but the corrected scaffold N50 measure. This result highlights the inherent difficulty of assessing the assembly quality even when a reference sequence is available. In this case, Allpaths-LG scaffold covers a larger stretch of the genome; however, at the cost of errors both within the contigs and in their relative placement. Furthermore, the CABOG assembler is able to align nearly 0.1% more mate-pairs than Allpaths-LG, despite having a far smaller scaffold size.

The Assemblathon 1 competition [19] further demonstrated the difficulty of accurately assessing the relative quality of genome assemblies even when a correct reference sequence is available. The authors developed a collection of quality metrics that measure the stretch of a correctly assembled sequence (for example, contig path NG50 and scaffold path NG50), the amount of structural errors (such as insertions, deletions, and translocation), the long range contiguity (for example, the average distance between correctly paired genomic loci), the number of copy number errors, and the coverage within the assembly or only within coding regions. All these metrics were computed with respect to two reference haplotypes, from which the read data were simulated. The authors ranked the different assemblies by each of the metrics and used the combined information to rank the assemblies quality.

In Figure 2.4, we compare the rankings provided by our LAP score to the rankings generated by the Assemblathon 1 competition. In addition to LAP, the figure also includes two variants of the most commonly used *de novo* measure of

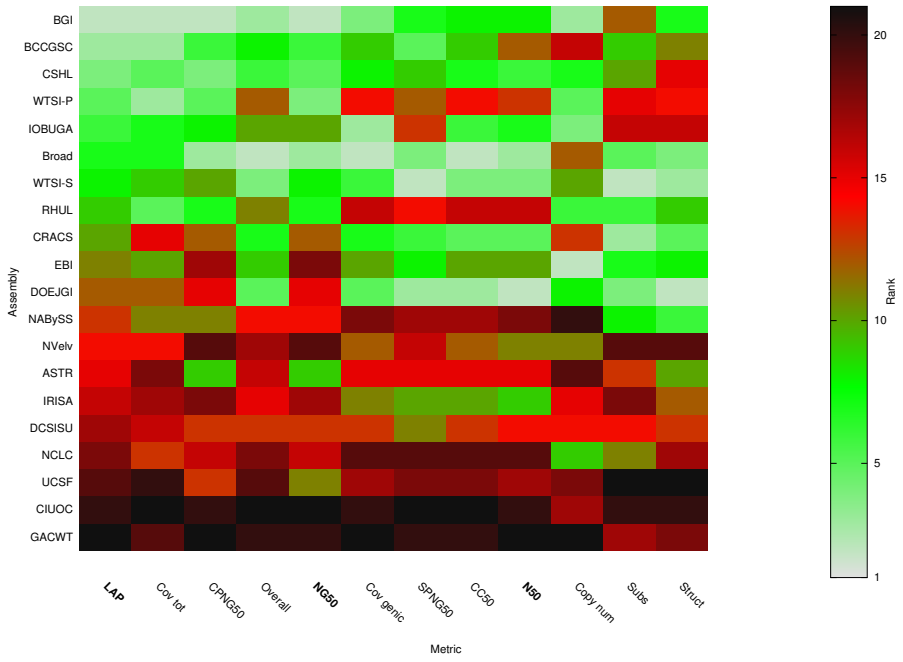


Figure 2.4: Comparison between LAP scores and the rankings of the top assemblies generated in the Assemblathon 1 competition. The colors represent the relative ranking provided by the individual metrics (best - green, worst - red): log average probability (LAP), overall coverage (Cov tot), contig path NG50 (CPNG50), sum of all rankings from Assemblathon 1 (Overall), weighted median contig size based on estimated genome size (NG50), coverage within coding sequences (Cov genic), scaffold path NG50 (SPNG50), length for which half of any two valid columns in the assembly are correct in order and orientation (CC50), weighted median contig size based on total assembly size (N50), proportion of columns with a copy number error (Copy num), total substitution errors per correct bit (Subs), and sum of structural errors (Struct). Column descriptions and underlying data obtained from Table 3 in Earl et al. [19]. Columns are sorted according to the level of concordance with the LAP ranking. *De novo* measures are highlighted in bold.

assembly size, N50 – the weighted median contig size, that is, the length of largest contig c such that the total size of the contigs larger than c exceeds half of the genome size. N50 uses the total assembly size as a proxy for the genome size while the NG50 value uses a guess of the actual genome size to compute the N50 value.

The more accurate estimation of the genome size results in a better NG50's ranking, confirmed by the concordance with our LAP score.

The overall coverage measure (percentage of the reference haplotypes covered by a particular assembly) correlates better with the LAP score than the other metrics. This result is not surprising as the LAP score is strongly affected by the number of the reads that can be correctly mapped to an assembly, which is ultimately correlated with the concordance between the assembly and the correct reference sequence. Interestingly, the overall rankings differ between LAP and the conclusions of the Assemblathon 1 study. Our analysis suggests that the BGI assembly is the best while the Assemblathon 1 picked the Broad assembly as the winner. This discrepancy can be partially explained in part by the Broad's high performance within the genic regions (LAP does not distinguish between genic and inter-genic segments) and the large weight placed on the BGI's assembly's poor performance in terms of substitution errors which have a relatively small effect on the LAP score.

It is important to note that while LAP and the Assemblathon 1 results disagree in the exact total ranking of the assemblies, the top 11 assemblies are the same, meaning they are fundamentally of better quality than the remaining 9 assemblies presented in the Assemblathon 1 paper. In fact, the Assemblathon overall score jumps from 74 for the 11th (WTSI-P) assembly to 99 for the 12th (DCSISU) assembly, indicating a substantial qualitative difference. This is also reflected in the corresponding jump in the LAP score from -37.326 to -39.441 for the 11th (DOEJGI) and 12th (NABySS) assemblies, respectively.

2.3.1.4 The effect of a contaminant DNA on the assessment of the assembly quality

The Assemblathon 1 dataset provides an interesting challenge to the assembly assessment. The simulated libraries, generated in this project from the human chromosome 13, also included approximately 5% of the contaminant DNA from an *Escherichia coli* genome to simulate commonly encountered laboratory contamination that possibly occur due to the fragments of the cloning vector being sequenced along with the genome of interest. The participants to the Assemblathon 1 competition were given the option to either remove the contaminant DNA prior to assembly or retain the corresponding sequences in their assembly. This decision has little effect on comparison between the resulting assembly and the correct reference genome in the Assemblathon 1; however, the ability of an assembler to correctly reconstruct the contaminant genome significantly affects the corresponding LAP score.

Indeed, the LAP score (Figure 2.5) computed from the entire set of reads (the red crosses) and that computed after the contaminant reads were removed (the blue crosses) are strongly correlated, the latter scores are slightly lower since they were computed on the smaller dataset. In several cases, the assembly was performed after removal of the contaminant DNA (see "jumps" in Figure 2.5). These assemblies are penalized by our framework for not assembling the contaminant DNA, a penalty that is removed once the same set of reads is used for both assembly and quality assessment.

It is important to stress that the LAP scores can only be meaningfully com-

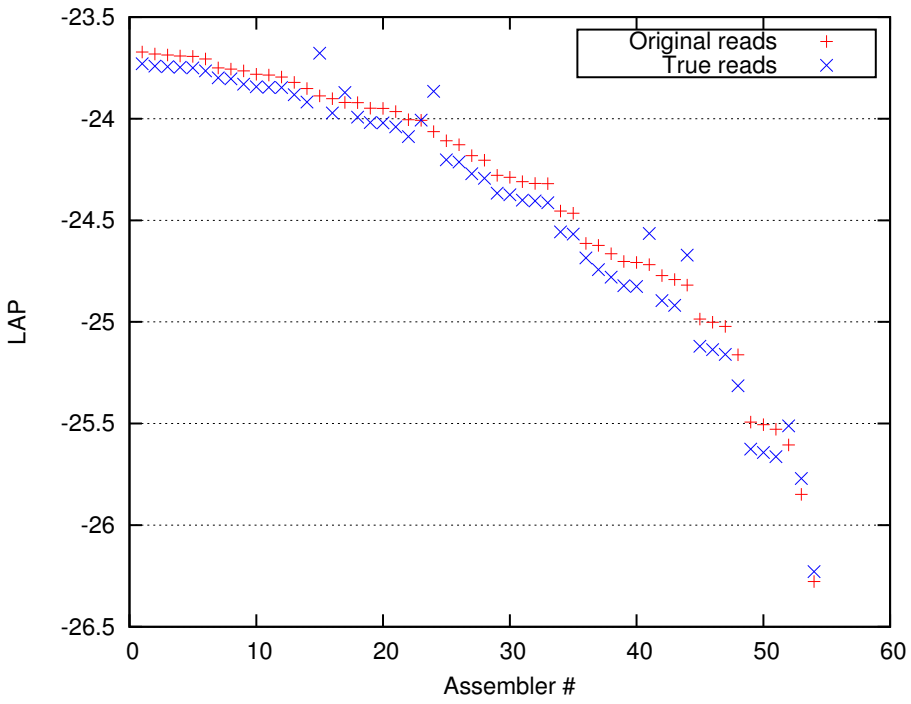


Figure 2.5: Effect of a contaminant DNA on the computation of the LAP scores. Red crosses are the LAP scores computed on the entire read set (including contamination). Blue crosses are the LAP scores computed only on the 'true' reads that map to the genome of interest. The corresponding LAP scores are quite similar (those obtained from a smaller set of reads are correspondingly smaller) except for those of assemblies that removed the contaminant DNA prior to assembly, and receive a boost in the LAP scores obtained on the "true" data.

pared across the assemblies generated from the same read set. If a contaminant is known it should either be removed from or retained within the dataset for all assemblers being compared; otherwise, the corresponding scores can not be directly compared. Note that this property is not unique to our measure: ignoring or assembling contaminant DNA also affects other traditional measures of quality, such as the N50 value or any reference-based measures, for example, in the case where the contaminant DNA shares significant similarity to the genome being assembled.

In practice, a 'contaminant' is not known *a priori*, and its definition depends on the specifics of an experiment. In general, it is difficult, if not impossible, to distinguish between environmental contaminants and true artifacts in the data, both in the context of metagenomic projects and in the case of isolate genomes. For example, the *Bacillus anthracis* samples from the bioterror attack in 2001, which were originally presumed to be uniform, contained a mixture of very closely related strains, and the characteristics of this mixture formed an important forensic marker in the investigation [40].

2.3.1.5 A useful application: tuning assembly parameters

Our discussion so far has focused on comparing the output of different assembly software with the goal of choosing the best assembler for a particular dataset. The developed probabilistic framework can also be used to better choose the combination of parameters that allow a particular assembly to achieve better results. To demonstrate this use case, we target the task of selecting the "best" (in terms of final assembly quality) k-mer length for a de Bruijn graph-based assembler. We focus here on SOAPdenovo assemblies of the *Escherichia coli* K12 MG1655 genome (Figure 2.6).

Without the availability of a reference sequence, users of assembly software usually rely on the N50 value as a proxy for the assembly quality. In this case, there is a clearly defined peak in N50 at k=79 (114,112 bp). After adjusting for the assembly errors, there is a collection of the assemblies (k=47-51, 55-75) with nearly

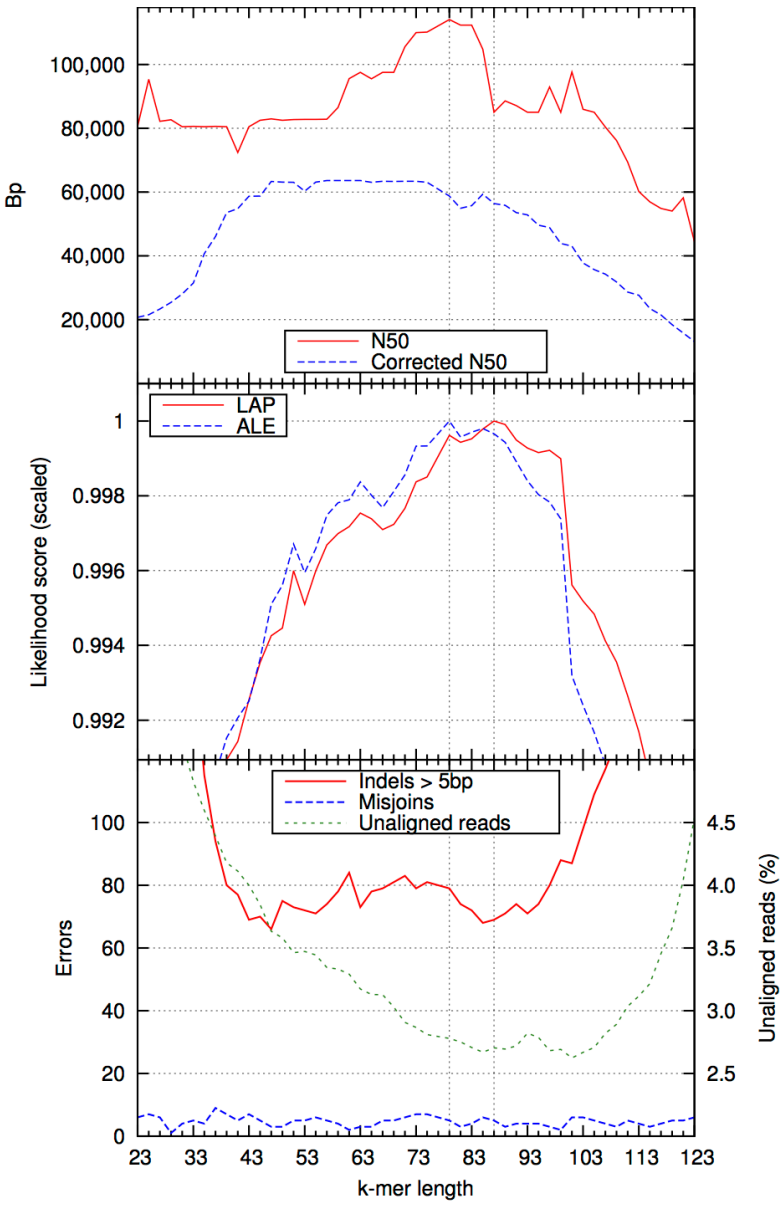


Figure 2.6: Tuning SOAPdenovo k-mer parameter using LAP. LAP, N50, and corrected N50 are plotted for various SOAPdenovo assemblies of *E. coli* K12 MG1655 dataset for different k-mer sizes (k=23-123). ALE [36] scores are plotted alongside the LAP to show the differences between their underlying likelihood models. Also included is a breakdown of the errors along with the percentage of the unaligned reads for the various SOAPdenovo assemblies. Two vertical lines (at k=79 and k=87) correspond to the maximum ALE and LAP score, respectively.

identical corrected N50s (∼64,000 bp). These assemblies range in N50 from ∼80-115 kbp. Our *de novo* measure LAP shows a clear peak at k=87, which corresponds to a corrected N50 of 59,352 bp. It is important to note that despite roughly a 7% difference from the peak in corrected N50 (k=63), the best LAP assembly contains 4 fewer indels larger than 5 bp, while also aligns roughly 54,000 more reads.

Alongside our LAP, we plot the likelihoods calculated from another assembly evaluator framework, ALE [36]. The assembly with the highest ALE score (k=79) corresponds to the N50 peak. Compared to the LAP selected assembly, the ALE selected assembly contains 10 more indels larger than 5 bp and has a 49% drop from N50 to corrected N50 compared to the 35% drop between those values for the LAP's selected assembly.

2.4 Discussion

In this chapter, we have proposed a paradigm for the *de novo* evaluation of genome assemblies. While the general paradigm could, in principle, be used to provide an objective score of assembly quality, our practical implementation of this paradigm, called the Log Average Probability (LAP), is dataset specific and should only be used to provide relative rankings of different assemblies of the same dataset. Unlike traditional measures of assembly contiguity (such as the N50 value), our reference-independent LAP scores correlate with reference-based measures of assembly quality.

We would like to stress that *de novo* measures of assembly quality, such as

ours, are critically needed by researchers targeting an assembly of yet unknown genomes. The specific characteristics of the data being assembled have a significant impact on the performance of genome assemblers (in the Assemblathon 1 [19] and GAGE [20] competitions, for example, different assemblers 'won' the competition depending on the analyzed dataset); thus, the reference-based quality assessments cannot be reliably generalized to new genome projects.

In this chapter, we have made a number of simplifying assumptions for modeling the sequencing process; specifically, that the sequencing process is uniform (both in the coverage, and the error profile), and that the reads are independently sampled from the genome (with the exception of the dependence imposed by mate-pair experiments). While our approach can detect copy number differences (unless the entire genome is exactly duplicated), it is with the caveat that sequencing biases within repetitive regions can possibly mask mis-assemblies. More precise models of the sequencing process that relax these assumptions can be easily incorporated into our framework (e.g., effects of G/C content on sequencing depth, or technology-specific error profiles). We plan to create technology-specific variants of our score to keep up with the rapid changes in the characteristics of the sequencing data as new instruments and/or chemistries become available. Furthermore, the probabilistic framework presented here can be used to incorporate other types of information on the assembly quality, for example, optical mapping data [21].

In our assembler parameter-tuning experiment, we generated assemblies of *Escherichia coli* K12 MG1655 using every allowed k-mer value. While this approach may be computationally feasible for smaller genomes, it is inefficient for very large,

complex genomes. One solution would be to use an optimization strategy for selecting potential k-mer values, e.g., with simulated annealing.

While there are differences between the LAP score and recent likelihood-based metrics, ALE and CGAL, these differences are quite small (Table 3 and Figure 2.6). Thus, it is important to discuss the technical improvements over ALE and CGAL. ALE's score did not perform quite as well as our LAP score on the parameter tuning experiment, and CGAL is unable to evaluate all of the GAGE assemblies due to the technical limitations of Bowtie 2. Bowtie 2 was not designed for reporting *all* read alignments, which makes it very slow on large genomes. This problem will become more prevalent as sequencing costs continue to decrease, allowing for more complex genomes to be sequenced and assembled. Our framework overcomes CGAL's limitations by allowing users to calculate the LAP score via the dynamic programming method on a subset of the reads or by using the SAM file produced from a read alignment tool designed for finding all alignments (e.g., mrsFAST [41]).

Our original goal was not to detect assembly errors, but to provide a global measure of how good an assembly may be. We plan to extend our framework to detect assembly errors by adopting a similar approach to that demonstrated by ALE.

It is important to note that we have focused on a very specific use case for assembly – the complete reconstruction of a given genome. Assembly algorithms are used in a number of other biological applications, whose specific characteristics affect the validation of the resulting assembly. For example, studies targeting the genic regions of an organism may tolerate large-scale rearrangements as long as the

individual genes are correctly reconstructed. In this context, the validation framework would penalize substitution errors and small insertions or deletions (which potentially affect gene structure) more than mis-joins within intergenic regions. Such application specific tuning is possible within the proposed overall framework, and we envisage the creation of a collection of community-supported modules that compute application-specific LAP scores.

Our discussion has focused on the assembly of single genomes, however the LAP score, as described, can also be directly used in the context of diploid genomes or metagenomic mixtures. In this case, our score implicitly assumes that the goal of the assembler is to correctly reassemble both the sequence and the relative abundances of the individual haplotypes. Assume, for example, a simple metagenomic sample that contains two organisms; one that is twice as abundant as the other one. An assembler that produces three sequences, corresponding to the three 'haplotypes' in the sample (whether explicitly outputting two, perhaps identical, versions of the abundant organism or reporting the copy-number difference in some other way) would obtain a better LAP score than an assembler that only reported two sequences without any indication of their relative abundance. As a result, the majority of metagenomic assemblers available today, which only output the consensus sequence and not the relative abundance of the contigs, would score poorly under our score. We hope that our work will inspire the developers of future metagenomic assemblers to also output information on the relative abundance of the reconstructed sequences, information that is critical to the analysis of the data, yet rarely reported by existing tools.

Finally, we propose that measures such as ours, which objectively capture the fit between the data being assembled and the output produced by the assembler without relying on curated reference data sets, become a standard tool in evaluating and comparing assembly tools, allowing the community to move beyond simplistic measures of contiguity such as the ubiquitous N50 measure.

Chapter 3: Comparing Metagenomic Assemblies

3.1 Introduction

Despite the unresolved challenges of clonal genome assembly, the decreasing costs of sequencing technology has led to a sharp increase in metagenomics projects over the past decade. These projects allow us to better understand the diversity and function of microbial communities found in the environment, including the ocean [42–44], Arctic regions [45], other living organisms [46] and the human body [47,48]. Traditional *de novo* genome assemblers have trouble assembling these datasets due to the presence of closely related species and and the need to distinguish between true polymorphisms and errors arising from the sequencing technology. Metagenomic assemblers often use heuristics based on sequencing (Meta-IDBA [49] and MetaVelvet [50]) and k-mer (Ray Meta [51]) coverage to split the assembly graph into subcomponents that represent different organisms, then apply traditional assembly algorithms on the individual organisms.

As the number of metagenomic assemblers available to researchers continues to increase, the development of approaches for validating and comparing the output of these tools is of critical importance. Despite the incremental improvements in performance, none of the assembler tools available today outperforms the rest in all

cases (as highlighted by recent assembly bake-offs GAGE [20] and Assemblathons 1 [19] and 2 [52]). Different assemblers "win" depending on the specific downstream analyses, structure of the genome, and sequencing technology used. These competitions highlight the inherent difficulty of assessing assembly quality – where do you set the line between increased contiguity and decreasing accuracy of the resulting sequence? Evaluating the trade-off between increased contiguity and errors is difficult even when there is a gold standard reference genome to compare to, which is not available in most practical assembly cases. Thus, we are forced to heavily rely on *de novo* approaches based on sequence data alone.

Most of the previous *de novo* and reference-based validation methods have been designed for single genome assembly. Currently, there are no universally-accepted reference-based metrics for evaluating metagenomic assemblies. Despite reference sequences being available for a small fraction of organisms found in metagenomic environments [53, 54], it is not clear how to distinguish errors from genomic variants found within a population. Furthermore, it is not clear how to weigh errors occurring in more abundant organisms. Likelihood-based frameworks, such as ALE [36], CGAL [37], and LAP [28], rely on the assumption that the sequencing process is approximately uniform across the genome; however, the sequencing depth across genomes in metagenomic samples can vary greatly [55–58].

In this chapter, we describe an extension to our LAP framework to evaluate metagenomic assemblies. We will show that by modifying our likelihood calculation to take into account contig abundances, we can accurately and efficiently evaluate metagenomic assemblies. We evaluate our extended framework on data from the

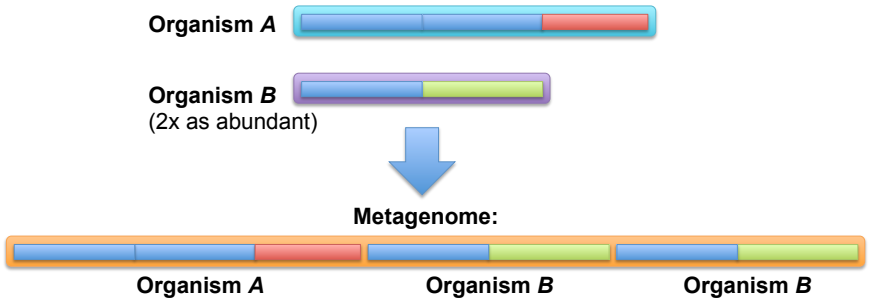


Figure 3.1: The metagenome of an environment can be viewed as the concatenation of the organisms found in the environment whose multiplicity is determined by their abundance.

Human Microbiome Project (HMP). Finally, we show how our LAP framework can be used automatically by the metagenomic assembly pipeline, MetAMOS [27], to select the best assembler for a specific dataset, and to provide users with a measure of assembly quality. The software implementing our approach is made available, open-source and free of charge, at: `http://assembly-eval.sourceforge.net/` and with the MetAMOS package: `https://github.com/treangen/MetAMOS`.

3.2 Methods

3.2.1 Extending LAP to metagenomic assemblies

An important simplifying assumption of our framework is that the sequencing process is uniform in coverage. In metagenomics, however, the relative abundances of organisms are rarely uniform [55–58], reflecting the difference in abundance between the different organisms within a community. Here we show that taking this abundance information into account allows us to extend the LAP framework to

metagenomic data. We now assume that while the abundances of each organism may vary dramatically, the sequencing process still has uniform coverage across the *entire* community. For example, consider a simple community containing two organisms (A and B), one which is twice as abundant as the other. This community, thus, comprises twice as much of A's DNA than that of B. Assume, for simplicity, that the community contains exactly three chromosomes (two of A and one of B). A random sequencing process would sample each of these equally, and an ideal metagenomic assembler would produce two contigs, one covered twice as deep as the other.

In essence, we view the collection of individual genomes and their relative abundances as a single *metagenome* where each genome is duplicated based on their abundance (Figure 3.1). This setting is similar to that of repeats in single genome assembly, where a repetitive element can now include an entire genome. Like in the case of single genomes, the assembler that correctly estimates these repeat counts maximizes the LAP score. In other words, in order to accurately evaluate the metagenomic assemblies using our LAP framework, the abundance (or copy number) of each contig is needed. As most metagenomic assemblers do not report this information, here we use the average coverage of the contig (provided by the MetAMOS pipeline) to represent the copy number.

In the error-free model, we compute the probability of a read, p_r, given the assembled sequence and abundance as:

$$p_r = \frac{\sum_{c \in \text{Contigs}} \text{abun}(c) * n_{rc}}{2\hat{L}} \tag{3.1}$$

$$\hat{L} = \sum_{c \in \text{Contigs}} \text{abun}(c) * L_c \tag{3.2}$$

where abun(c) is the abundance of contig c, n_{rc} is the number of times read r occurs in contig c, and $\hat{L}$ is the adjusted total assembly length. In the case where the abundance of each contig is 1, calculating p_r is identical to the original LAP (single genome) formulation. A similar modification can be done to handle sequencing errors outlined in [28].

Our prior work has shown we can approximate the probabilities using fast and memory efficient search alignment programs (e.g., Bowtie2 [38]) when it is impractical to calculate the exact probabilities for large read sets. We can apply the metagenomics modification above to the alignment tool-based method:

$$p_r = \frac{\sum_{j \in S_r} \text{abun}(j_{\text{contig}}) * p_{r,j_{\text{subs}}}}{2\hat{L}} \tag{3.3}$$

where S_r is the set of alignments in the SAM file for the read r and the probability of alignment, $p_{r,j_{\text{subs}}}$, is approximated by $\epsilon^{subs}(1-\epsilon)^{l-subs}$ where ϵ is the probability of an error (a mismatch, an insertion, or a deletion).

An important factor in any likelihood-based assembly evaluation framework is the handling of reads that do not align well to the given assembly. In practice, unalignable reads are often the result of sequencing errors and contaminants. If these reads are given a probability close to 0, then the best assembler would be the one that incorporates the most reads. In our original LAP framework, a read that does not align well does not decrease the overall assembly probability more than the probability of an assembly that contains the appended read as an independent

contig. This does not change when we handle metagenomic data, since the average coverage of the "new" contig is one.

3.2.2 Integration into MetAMOS

In addition to being a standalone framework, the software implementing our metagenomic LAP approach comes packaged with the MetAMOS pipeline [59]. This allows users the option to run MetAMOS with different assemblers and have our framework automatically select the assembly with the highest LAP score without any prior knowledge from the user. The first step of the MetAMOS pipeline is to `Preprocess` the reads, optionally filtering out low quality reads. Those reads are used by the next step `Assemble`. Users specify the desired assembler using the `-a` parameter of `runPipeline`. We modified MetAMOS so users can now specify multiple assemblers (comma-separated) after the `-a` parameter, and `runPipeline` will run all assemblers and select the assembly yielding the highest LAP score to be used in downstream analyses.

3.3 Results

3.3.1 Likelihood score maximized using correct abundances

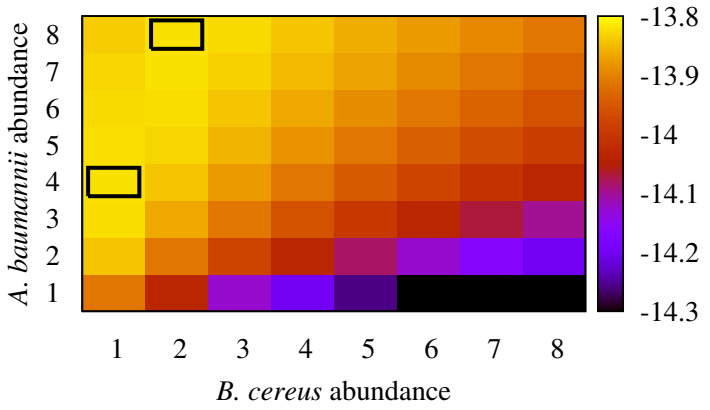
A key property of our framework is that the correct copy numbers (abundances) and assemblies maximizes our LAP score. To illustrate this property, we simulated two metagenomic communities and calculated the LAP of the reference genomes with a combination of abundances. The first simulated community con-

sisted of *Bacillus cereus* and *Acinetobacter baumannii* at a ratio of 1:4. We generated 200bp reads at 20x coverage of the metagenome (20x of *B. cereus* and 80x of *A baumannii*). We calculated the LAP scores of the error-free reference genomes for all combinations of abundances (ranging from 1 copy to 8 copies) for each bacteria. The second simulated community consisted of *Bacillus cereus* and *Actinomyces odontolyticus* at a ratio of 4:7. We generated 200bp reads at 20x coverage of the metagenome (80x of *B. cereus* and 140x of *A odontolyticus*). We calculated the LAP scores of the error-free reference genomes for all combinations of abundances (ranging from 1 copy to 13 copies) for each bacteria.

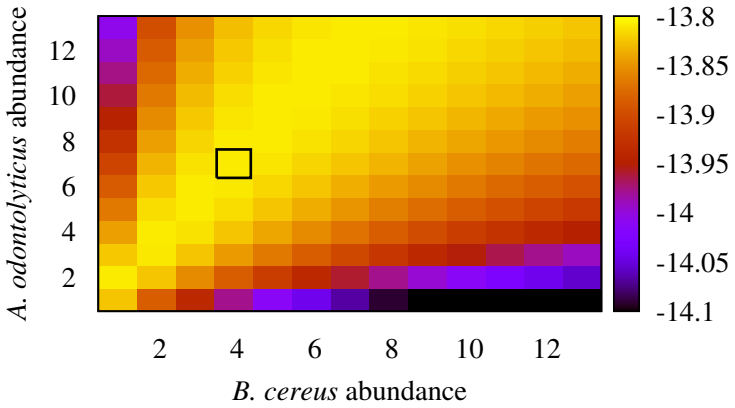
We expect the highest LAP scores for the assemblies that contain the correct abundance ratios (1:4 or 2:8 in the first community, and 4:7 in the second community). As seen in Figs. 3.2(a) and 3.2(b), our LAP score is able to accurately reflect the varying organism abundance ratios present in the sample. The LAP score increases as the estimates approach the true abundance ratios, with the true ratio yielding the highest LAP scores in both communities.

3.3.2 Impact of errors on synthetic metagenomes

One of the often overlooked aspects of metagenomic assembly evaluation is the weighing of errors that occur in contigs with different abundances. In metagenomic samples the relative organism abundances can vary by orders of magnitudes. A typical reference-based evaluation would equally weight the errors irrespective of the abundance of the erroneous contigs. The proposed metagenomic LAP score, how-



(a) *B. cereus* (1 copy, 5.2MB) and *A. baumannii* (4 copies, 4.0MB)



(b) *B. cereus* (4 copies, 5.2MB) and *A. odontolyticus* (7 copies, 2.4MB)

Figure 3.2: LAP scores for simulated metagenomic communities. Each cell (x,y) represents the LAP score for a mixture of x copies of the x-label bacteria and y copies of the y-label bacteria. In both groups, the true abundance ratios maximize the LAP score (indicated by a black rectangle in respective plots).

ever, automatically handles this situation and appropriately weighs errors according to genome abundance. To illustrate this, we simulated a small metagenomic community consisting of *Escherichia coli* and *Bacillus cereus* at a 5:1 ratio. We introduced an increasing number of common assembly errors (single-base substitutions, insertions, deletions, and inversions) into the two organisms assemblies and observed the resulting LAP score.

As shown in Figure 3.3, the higher the number of synthetic errors, the lower the LAP score. Insertions/deletions were more deleterious to the LAP score than substitutions, since in addition to causing a mismatch, an insertion/deletion changes the overall genome size. Although inversions did not change the overall genome size (and would therefore not be detected by simplistic measures such as N50), these errors had the greatest impact on the LAP score because they prevented the alignment of reads across the boundaries of the inversions.

As expected, errors introduced into the more abundant organism, *E. coli*, had a greater affect on the LAP score than those inserted into *B. cereus*. Our LAP score was able to accurately weigh the errors by the abundance of each organism.

3.3.3 Likelihood scores correlate with reference-based metrics

With real metagenomic samples, it is difficult to make evaluations given the lack of high quality references. Using purely simulated data has the issue of not accurately capturing the error and bias introduced by sequencing technology. Thus, to evaluate our LAP score, we use two 'mock' communities (Even and Staggered)

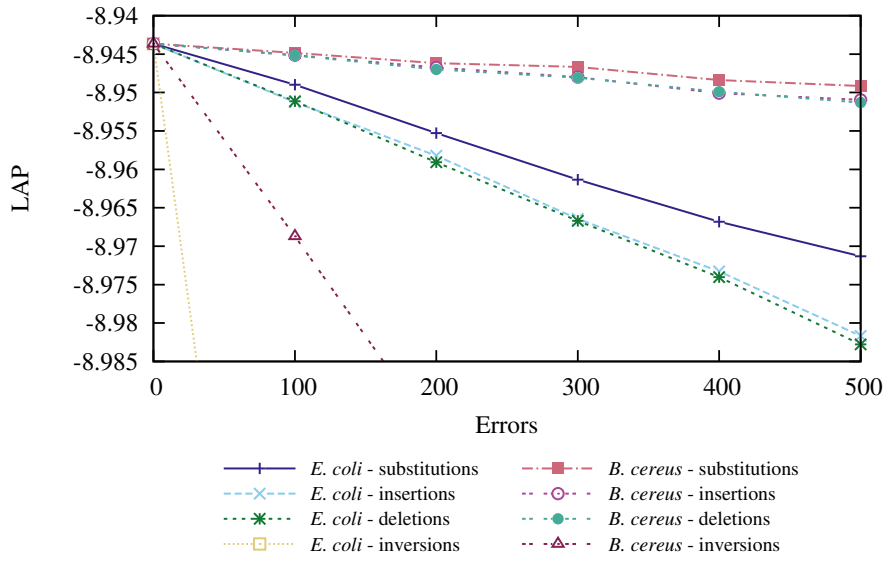


Figure 3.3: Synthetic errors in simulated *E. coli* (5 copies, 4.9Mbp) *B. cereus* (1 copy, 5.2Mbp) community.

provided by the Human Microbiome Project (HMP) consortium [25, 26]. These communities were created using specific DNA sequences from organisms with known reference genomes (consisting of over 20 bacterial genomes and a few eukaryotes) and abundances. The mock Even community consisted of 100,000 16S copies per organism per aliquot, while the mock Staggered community consisted of 1,000 to 1,000,000 16S copies per organism per aliquot. Data used from the HMP mock communities are available at `http://www.ncbi.nlm.nih.gov/bioproject/48475`. We calculated the LAP score on assemblies produced by MetAMOS [27] using several assemblers: SOAPdenovo [30], Metavelvet [50], Velvet [10], and Meta-IDBA [49]. The additional *de novo* and reference-based metrics for the assemblies were taken from MetAMOS [27]. These metrics include:

- number contigs (# ctgs) – total number of contigs/scaffolds in the assembly
- good contigs (Good Ctgs) – fraction of contigs that mapped without errors to reference genomes
- total aligned (Total aln) – total amount of sequence (in Mbp) that can be aligned to the reference genomes
- slight mis-assemblies (Slt) – alignments that cover 80% or more of the aligned contig in a single match (Slt)
- heavy misassemblies (Hvy) – alignments that cover less than 80% of the aligned contig in a single match or have two or more matches to a single reference
- chimeras (Ch) – contigs with matches to two distinct reference genomes
- size at 10 megabases (Size @ 10 Mbp) - the size of the largest contig c such that the sum of all contigs larger than c is more than 10 Mbp (similar to the commonly used N50 size)
- max contig size (Max ctg size) – size of the largest contig in the assembly
- errors per megabase (Err per Mbp) – average number of errors per Mbp in the assembly

Generally, the *de novo* LAP scores agree with the referenced-based metrics (Table 3.1). In the mock Even dataset, SOAPdenovo has the greatest LAP score, the highest fraction of contigs that can align to a reference genome without error, total amount of sequence that can be aligned to a reference genome, while also having

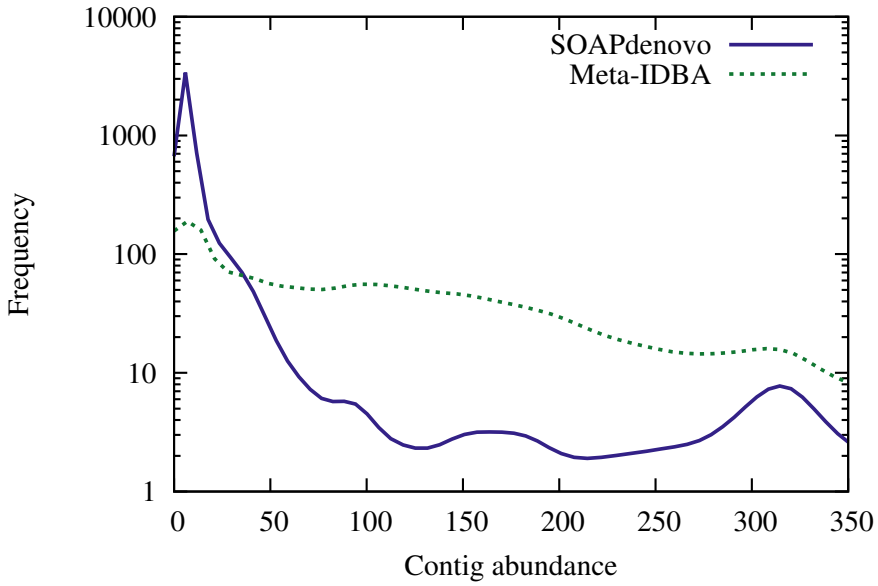


Figure 3.4: Frequency of contig abundances for assemblies of the HMP mock Staggered dataset.

the lowest amount of misassemblies (including chimeric) and errors per Mbp. It is important to note that if user selected an assembly based on the best contiguity at 10Mbp, they would select the MetaVelvet assembly, which contains double the error rate per Mbp as the SOAPdenovo assembly while aligning 2Mbp less to the references.

Dataset	Assembler	LAP	#ctgs	Good ctgs	Total aln	Slt	Hvy	Ch	Size @ 10 Mbp	Max ctg size	Err per Mbp	Aligned reads
mockE	SOAPdenovo	**-27.031**	63107	**99.3%**	**51**	**166**	131	**1**	28,208	249,819	**5.8**	**85.75%**
mockE	Velvet	-28.537	12,830	96.2%	41	256	**100**	2	42,269	179,673	8.7	83.30%
mockE	MetaVelvet	-27.102	22,772	96.8%	49	462	156	4	**62,138**	**367,458**	12.7	85.65%
mockE	Meta-IDBA	-31.166	22,032	95.4%	47	362	151	3	26,141	249,069	11	81.81%
mockS	SOAPdenovo	-60.161	44,928	**98.8%**	**28**	135	98	**0**	5,672	186,064	**8.3**	69.78%
mockS	Velvet	-60.711	21,050	95.8%	**28**	485	115	1	6,060	119,120	21.5	67.26%
mockS	MetaVelvet	-60.442	20,551	95.3%	**28**	517	143	3	6,685	**217,330**	20.1	67.72%
mockS	Meta-IDBA	**-58.851**	4,559	92.5%	18	**101**	**83**	**0**	**13,150**	119,604	10.2	**70.67%**
Tongue	SOAPdenovo	**-13.844**	35,230	**89.10%**	**11**	1,138	2,618	**0**	11,359	**238,051**	**341.5**	**88.14%**
Tongue	Meta-IDBA	-21.368	25,698	88.70%	7	**710**	**2,087**	**0**	4,215	59,188	399.6	58.89%

Table 3.1: Comparison of assembly statistics for HMP mock Even and mock Staggered datasets.Numbers in bold represent the best value for the specific dataset.

Since the abundances of each organism in the mock Even dataset are fairly similar, the mock Staggered abundance distribution creates a more realistic scenario encountered in metagenomic environments. Here, the Meta-IDBA assembly has the greatest LAP score, but aligns roughly a third less sequences to the reference genomes than SOAPdenovo. The Meta-IDBA assembly contains approximately a tenth of the amount of contigs (4,559 vs. 44,928) as SOAPdenovo. The SOAPdenovo assembly contains a greater number of contigs at a very low abundance (Figure 3.4). On large contigs Meta-IDBA performs better than SOAPdenovo and has a lower error rate (see Figure 4 in [27]). However, Meta-IDBA assembles a smaller fraction of the low-abundance genomes than SOAPdenovo, leading fewer sequences to align. The LAP score penalizes misassemblies within abundant contigs in the SOAPdenovo results.

Next, we applied our framework on real data where we did not know the actual genomes comprising the sample (HMP tongue dorsum female sample, SRS077736) (Table 3.1). Although we do not know for certain which organisms are present in the sample, the HMP identified a reference genome set with high similarity to the sequences within the sample (HMP Shotgun Community profiling SRS077736). The reference-based error metrics only consider chimeric errors due to the possibility of structural difference between an organism and its version in the reference database. We calculated the LAP of the assemblies using a single library consisting of 42,013,917 reads. The SOAPdenovo assembly had a far greater LAP score than the Meta-IDBA assembly. The higher score is due to the SOAPdenovo assembly recruiting more reads (88.14% to 58.89%) in relation to its genome size (46Mbp to

Assembler	Contigs	LAP	N50 (Kbp)	Errors
newbler	**1**	**-13.064**	**156**	1
SOAPdenovo	23	-14.238	9	**3**
Velvet	3	-13.157	92	**0**
MetaVelvet	3	-13.157	92	**0**

Table 3.2: Self-tuning MetAMOS using *C. ruddii* test dataset.

37Mbp) than the Meta-IDBA assembly. Furthermore, the MetAMOS metrics show that the SOAPdenovo assembly contained approximately 60 less errors per Mbp than the Meta-IDBA assembly.

3.3.4 Tuning assembly parameters for MetAMOS

Assemblathon1 [19] has shown that assembly experts can often get drastically different assemblies using the same assemblers, highlighting the difficulty of choosing the *right* parameters for a given assembler. Our metagenomic LAP framework comes packaged with the MetAMOS pipeline, allowing users the option to run MetAMOS with different assemblers and automatically select the assembly with the highest LAP score. This step occurs without any prior knowledge from the user.

We showcase the ease of use of the automated assembler selection within MetAMOS using the *Carsonella ruddii* (156Kbp) dataset packaged with MetAMOS (Table 3.3.3). Errors were found using DNADIFF [60] and MUMmer [61]. The newbler assembly produced one contig containing the complete *C. ruddii* genome. The SOAPdenovo assembly produced a severely fragmented assembly with the most number of errors. The MetaVelvet and Velvet produced identical assemblies, containing 3 contigs of sizes 92Kbp, 65Kbp, and 1.7Kbp, but contained an additional

158bp compared to the *C. ruddii* genome. Upon closer inspection, there were overlaps between the contigs ranging from 38bp to 73bp. This is not surprising given MetaVelvet's and Velvet's de bruijn graph-based approach could not resolve repetitive regions between the contigs. Newbler, on the other hand, contained only a single insertion error. The LAP score of the Newbler assembly was greater due to more reads being able to align across the regions that were broken apart in the MetaVelvet and Velvet assemblies. Additionally, the Newbler assembly did not contain the duplicated sequence found in the other assemblies. MetAMOS was able to select the most likely assembly without requiring any additional input from the user.

3.4 Discussion

In this chapter, we have proposed an extension to our LAP framework to perform *de novo* comparisons of metagenomic assemblies. Unlike traditional *de novo* metrics used for measuring assembly quality, our extended LAP score correlates well with reference-based measures of metagenomic assembly quality. However, in this study, we have realized that there is a lack of reference-based metrics when evaluating metagenomic assemblies. Misjoins betweens organisms may be more deleterious than misjoins within a single genome. Furthermore, current reference-based metrics do not take into account the relative abundances of the organisms when evaluating metagenomic assemblies. The metrics provided by MetAMOS do not factor in the contig abundances when examining assembly errors. This made it difficult to com-

pare our LAP score to their reference-based metrics because, intuitively, an error in a highly abundant organism should be *worse* than an error in a rare, low coverage organism. Our LAP score implicitly weighs the errors in abundant contigs more than those in lesser abundant contigs. In our results, we have proposed one such reference-based metric that scales the errors by the relative abundance of the contig it occurs within.

It is important to note that we have only focused on the complete reconstruction of the metagenome from the set of reads. Assembly algorithms are designed with specific biological applications in mind, such as, the conservative reconstruction of the genic regions. Studies focusing on the genic regions may tolerate large-scale rearrangements as long as the genic regions were correctly assembled. Conversely, other studies may want to focus on the reconstruction and detection of rare pathogenic bacteria in an environment. These application specific assembly algorithms all attempt to optimize their formulation of the assembly problem.

Our metagenomic LAP extension relies heavily on the idea that the sequencing process of the metagenome is roughly uniform, and that the reads are independently sampled from the genome. Biases exist in all steps of the sequencing process, from the extraction of DNA from organisms with different cell membranes/walls [55, 56] to the sequencing protocol used [57, 58]. In the future, we would like to implement a more specific model that better captures the sequencing process.

Results from GAGE [20] and Assemblathon [19,52] have shown that the specific characteristics of the data being assembled has a great impact on the performance of the assembler. This problem is magnified in metagenomic assembly. By integrating

our LAP framework in MetAMOS, we have allowed researchers to accurately and effortlessly run and evaluate assemblies without any prior knowledge on evaluating assembly quality.

In our framework, we use the average coverage of the contig (provided by MetAMOS) to estimate abundance. There are issues with this measure as it is possible that mis-assembled repeats within a contig will affect our estimate of depth of coverage and could impact our underlying statistics. A better approach is to use something more robust than the mean coverage, such as the median coverage, to avoid being influenced by such regions. While the user can supply the median coverage to our standalone framework, future work includes building this feature into MetAMOS. Another approach involves breaking contigs at regions of differing coverage (using tools such as AMOSvalidate [60]), so there will be less deviation in the average coverage within the contig.

It should be noted that in some cases it may not be tractable to run the complete collection of assemblers with MetAMOS. In such cases, we should first employ heuristics (such as [62]) to aid in selecting potential assemblers (and parameters) to run. For the assembler selection process, we can use the LAP framework's sampling procedure in combination with calculating read probabilities in parallel to reduce runtime.

Our goal was to provide a global measure of how good a metagenomic assembly may be, not to detect assembly errors. Other likelihood-based frameworks, such as ALE, use frequencies of certain sequences to aid in detection of possible chimeric contigs. We are able to apply similar modifications to our LAP framework to find

regions of possible misassembly. Finally, we plan to extend our framework to give a more detailed breakdown of the LAP scores of segments assembled using the same subset of reads across different assemblies. The goal would be to take high-scoring assembled segments from individual assemblies to recreate an assembly with overall greater likelihood. This approach will be of great benefit to the field of metagenomic assembly since assemblers are often designed with different constraints and goals in mind, e.g., low memory footprint, assembling high/low coverage organisms, or tolerating population polymorphisms. For example, on the mock Staggered dataset, Meta-IDBA best assembled the most abundant genomes while SOAPdenovo had a better representation of the low abundance organisms. Providing a systematic way of combining assembler approaches using our LAP score will produce better assemblies for downstream analyses.

3.5 Conclusion

In this chapter we have described an extension to our *de novo* assembly evaluation framework (LAP) for comparing metagenomic assemblies. We showed that the true metagenome and correct relative abundances maximizes our extended LAP score. Furthermore, we have integrated our framework into the metagenomic assembly pipeline MetAMOS, showing that any user is able to reproduce quality evaluations of metagenomic assemblies with relative ease.

Chapter 4: Regression Testing of Genome Assemblers

4.1 Introduction

The issues with testing "non-testable programs" were first raised by Davis and Weyuker in a 1981 paper [63]. An important characteristic of such programs is the absence of a *test oracle*, the mechanism that determines whether a software under test (SUT) executed correctly for a test case. Without a test oracle, a test case has no way to *pass* or *fail*. This calls into question the overall purpose and value of software testing.

Although there has been work in the area of testing such non-testable programs [64–72], in practice this problem continues to be a significant hurdle for test automation in many scientific domains, where it is either very expensive or impossible to determine the correct answer for a scientific problem [73] (e.g., validating machine learning classifiers [74], analyses of feature models [75]) or computation (e.g., processing large XML files [76]. image segmentation [77], mesh simplification [78]). This is especially problematic for the domain of bioinformatics, a largely software-intensive field. Bugs in bioinformatics software have the potential to lead to incorrect scientific conclusions. As observed by Chen et al. [79], "*incorrectly computed results may lead to wrong biological conclusion, and ... misguide downstream*

experiments."

Consider the classic problem in bioinformatics – *de novo genome sequence assembly*. The *genome sequence* of an organism is important for understanding its life cycle and evolution. Current sequencing technology is only able to produce *reads* (sequenced fragments) that are drastically shorter than the genome. Therefore, in order to carry out meaningful biological analyses, one must first *assemble* the original genome using *assemblers*. The result of the assembly is one or more *contigs* (contiguous sequence fragments) that can be ordered and oriented into *scaffolds* with *gaps* (unknown parts in the sequence). Current formulations of the genome assembly problem are optimization problems on graphs, which are known to be NP-hard [6]. In practice, assemblers are only able to return an approximate solution.

Because of the nature of the domain, it is very difficult to validate the correctness (*quality* [1]) of an assembly – the correct/expected solution is not known. In software testing terms, the *test oracle* is unavailable. Moreover, when researchers develop a new assembler, they often run it on a new dataset, making comparisons difficult. Monya [1] notes that the bioinformatics community needs to find "*ways to assess and improve assemblers in general*."

Hence, the community faces the following scenario: *iteratively improve the assember*, ensuring at each step that the assembly did indeed improve, and that no new bugs that might degrade the assembler's output were introduced. This puts us in the realm of *regression testing*. One way to determine whether bugs have not degraded a software's output is by using what we call a *diff*-based approach, i.e., running test cases on the old and new versions of the code and identifying differ-

ences in the tests' outcome [80]. Thus, regression testing employed by assemblers may compare the text output of an assembler on test datasets with previously computed assemblies to determine if the code changes produced a different assembly. Comparing the raw text outputs of an assembler is not robust enough to capture whether there were actual differences in the quality of assembly. Multiple assemblies of the same set of *reads* are acceptable as correct. Reordering the reads may produce different assemblies that have the same overall quality, but contain trivial differences, such as the assembly starting a different position in circular bacteria genome (Fig. 4.1).

```
>sample circular sequence
AGCATCTTTATTGGAGATGTGCCACAGCACATTG
```

```
>sample circular sequence rotated 1 char
GAGCATCTTTATTGGAGATGTGCCACAGCACATT
```

Figure 4.1: FASTA file containing two entries that represent the same circular sequence. Each entry consists of a single line descriptor starting with the > symbol, followed by the biological sequence. Text comparison tools would detect that these two sequences are different.

In this chapter, we present a novel assembler-specific regression testing framework that uses two assembly evaluation mechanisms: *assembly likelihood*, calculated using LAP [28], and *read-pair coverage*, calculated using REAPR [29], to determine if code modifications result in non-trivial changes in assembly quality. The log av-

erage probability, LAP, is the log of the geometric mean of the probability that the observed reads are generated from the given assembly. By modeling the sequencing process, we are able to accurately calculate this probability. REAPR is tool for detecting misassemblies using the coverage of read-pairs.

We evaluate our framework using SOAPdenovo [11] and Minimus [31]. SOAPdenovo is a widely popular *de novo* assembler designed for short reads that has been used in many high profile genome assemblies, including the giant panda [39]. Minimus is one of the several assembly pipelines in the AMOS software package. Minimus provides a good case study for software engineering in bioinformatics due to its open-source nature, modular design, and active developer community.

We study assembler evolution in two contexts. First, we examine how assembly quality changes throughout the version history of SOAPdenovo. Second, we show that our framework can correctly evaluate decrease in assembly quality using fault-seeded versions of Minimus. Our results show that our framework accurately detects trivial changes in assembly quality produced from permuted input reads and using multi-core systems, which fail to be detected using traditional regression testing methods.

Here we make the following contributions:

- Provides a regression testing framework novel to the domain of *de novo* genome assembly.
- Illustrates the benefits of developing a regression testing framework for untestable software leveraging existing third party tools.

- *All* of the software from our regression testing framework, experimental sequence data, assemblers, and results are available online.

We believe that this research is both timely and relevant. As sequencing technology becomes cheaper, assemblers will operate on increasingly larger data sets, *requiring* large multi-core machines in order to assemble these datasets in a reasonable time frame. Developers need to design test cases that match the complexity and size of practical datasets to adequately test their assembler. Depending on the underlying algorithms, concurrent programs may produce different outputs. Assemblers may produce slightly different assemblies than their single-threaded version, making it difficult to compare the raw text outputs.

In the next section, we describe how the problem of testing without an oracle is not limited to bioinformatics, and the different strategies typically used and their limitations. In Section 4.3, we describe our regression testing approach, briefly outlining the theory behind assembly likelihood and read-pair coverage and why they are our main measure of assembly quality. In Section 4.4, we show how our framework is able to accurately evaluate the trivial changes in assembly quality using real sequencing data. Then, we examine how assembly quality changes throughout the release history of SOAPdenovo. We wrap up our results showing the fault detection power of our framework using manually seeded faults within Minimus. In Section 4.5, we discuss the significance and limitations of our framework, and the lack of adequate testing within the assembler community. Finally, in Section 4.6, we conclude with a discussion of future research directions.

4.2 Related work

The difficulty in software testing without an oracle is not limited to genome assembly, but has been encountered in many other fields such as bioinformatics, weather prediction, and image and speech processing. In bioinformatics, for example, a common task is to find all potential mappings of a sequence to another reference sequence which contains at most a certain number of mismatches. Without an oracle, it is hard to check whether a sequence has been mapped to *all* positions in the reference sequence [81]. In weather prediction, software has no oracle to verify if it is functioning correctly. Discrepancies between the predicted and actual result can be attributed to an error in the model employed by the weather prediction program rather than an error in the software. However, this prediction model involves very complex computation, which makes it very hard to verify its output. Testing done on weather prediction is frequently used to test performance and scalability of the framework instead of the accuracy of the predictions [82], [83].

Literature has witnessed several techniques developed to address the lack of oracle in software testing. The first technique is *dual coding* or "pseudo-oracle" [84], where developers independently create a program with the same specification as the original. Identical input datasets are used and outputs from both programs are, then, compared. The extra overhead involved in creating a duplicate program as complex as genome assembler often makes this technique impractical. In order to reduce this overhead in some instances, McMinn [64] proposed program transformation which automatically creates pseudo-oracles by transforming aspects of the

SUT into alternative versions. He also proposed using search-based testing techniques to generate two types of inputs that have the potential to produce different outputs from the pseudo-oracles and the original program. The first type of input targets programs with numerical computations while the second one focuses on multi-threaded code with the presence of race conditions.

Metamorphic testing [85] proposed by Chen et al. is another common technique to deal with testing applications without oracle. It identifies expected relation properties among inputs and their corresponding outputs, which can detect incorrect output but cannot validate the correct one. Metamorphic testing is widely-applied in many specific domains such as mesh simplification programs, stochastic optimization algorithms, machine learning classifiers and feature models. Chan et al. applied metamorphic testing to mesh simplification programs which create 3-D polygonal models similar to an original polygonal model, yet with fewer polygons [68, 78]. The test oracle problem in this domain is that the programs produce different graphic despite the same original polygonal model being used. The proposed iterative solution uses a reference model of the SUT as the pseudo-oracle to train a classifier which categorizes a test case into "failed" or "passed". However, since the passed test cases may be misclassified, they are then inputted into a metamorphic testing model as initial test cases to generate follow-up test cases which in turn are classified by the classifier. Yoo [67] focused on solving the same oracle problem in stochastic optimization algorithms whose performance depends not only on the correctness of implementations but also on the problem instances they are used to solve. The paper provides a comparison and evaluation of the impact of different problem in-

stances on the effectiveness of metamorphic testing of stochastic optimization. Xie et al. [74] used metaphoric testing to test machine learning classifiers. Their solution first identifies all the necessary metamorphic relations that classifiers would be expected to demonstrate, then checks if the corresponding classifier algorithm satisfies these relations. A failure to exhibit the relation indicates a fault. In feature model analysis tools, output is very difficult to evaluate due to the combinatorial complexity and numerous operations of the analyses. The current testing method is very time-consuming and error-prone; thus, metamorphic testing is used to automatically generate test data for the tools.

There are, however, some limitations in metamorphic testing such as manually intensive, insufficient number of metamorphic properties and ineffective fault detection in individual functions. In order to reduce these limitations, Christian proposed metamorphic runtime checking [65], which specifies the metamorphic properties at the function level rather than at the application level, and automated metamorphic system testing, [66] which requires little manual intervention.

4.3 Methods

Here we present an assembler regression testing framework that utilizes a non-traditional test oracle, one that need not assess whether a test case passed or failed; rather, it computes "goodness of output" measure or quality for assemblers. Our framework uses two mechanisms to accurately assess assembly quality: *assembly likelihood* and *read-pair coverage*. These mechanisms serve as our testing oracle

in the sense that we will be modeling a process (sequencing) that the software (assembler) is trying to reverse. We will outline the importance of each mechanism and the software used in our framework.

4.3.1 Regression testing framework

4.3.1.1 Assembly likelihood

The correct assembly of a set of sequences should be consistent with the statistical characteristics of the data generation process [86]. In other words, we can evaluate an assembly based on the likelihood that the reads could have been generated from it. An important property of this mechanism is that the true genome maximizes this likelihood [28]. Recent tools have utilized this theory: ALE [36], CGAL [37], and LAP [28].

For our testing framework, we have selected LAP as our tool to evaluate assembly likelihood. The LAP framework defines the quality of an assembly as the probability that the observed reads, R, are generated from the given assembly, A: $\Pr[R|A]]$. This conditional probability is the product of the individual read probabilities, p_r (assuming that the event of observing each read is independent). That is,

$$\Pr[R|A] = \prod_{r \in R} p_r \tag{4.1}$$

The probability of each read, p_r, is calculated by modeling the data generation process, which varies depending on the sequencing technology used. If we assume the reads are generated error-free and uniformly at random from the given genome,

then a read may be sequenced starting from any position of the genome with equal probability. Thus, if a read matches at n_r positions on the assembly of length L, then

$$p_r = \frac{n_r}{2L} \tag{4.2}$$

The assembly length is doubled due to the double-stranded nature of DNA molecules.

Modifying the calculation of p_r to handle practical contraints such as sequencing errors, paired-reads, and large datasets are detailed in [28].

We can provide a brief demonstration of the effectiveness of the LAP in detecting trivial differences in assembly quality using the sample circular sequence from Fig. 4.1. The length of the circular sequence is 35 characters, known as base pairs (bps). Due to the inability to represent the circular nature of the sequence in the file format used for storing sequences (FASTA), we must arbitrarily break the circular sequence into a linear fragment. Let's assume we are able to generate error-free reads of length 5 from each position in the circular sequence, resulting in 35 reads:

```
>sample circular sequence
AGCATCTTTATTGGAGATGTGCCACAGCACATTG


Reads:
    AGCAT, GCATC, ..., CATTG (31 total)
Reads that wrap around:
    ATTGA, TTGAG, TGAGC, GAGCA (4 total)
```

Assuming that each read aligns exactly at most one location, then 31 reads

will align exactly 1 time, while the 4 reads that span the end of the sequence will be unable to align. If we align the reads to the sample circular sequence *rotated* 1 base pair, then it should be apparent that we get the same number of reads that match exactly 1 time (albeit different reads), and the same number of reads that do not match. Therefore, $\Pr[R|A] = \Pr[R|A_{rotated}]$ and the LAP of each assembly will be equal. We are able to determine that these assemblies are of equivalent quality, unlike the **diff**-based method.

We use LAP in our framework over CGAL and ALE because the LAP score can be calculated accurately and efficiently using a sample of the reads, making it practical for large datasets.

4.3.1.2 Read-pair coverage

Many current sequencing technologies produce read-pairs, where reads are sequenced from opposing ends of the same fragment. These read-pairs are used to resolve genomic repeats as well as orient contigs into scaffolds (contigs with gaps that are connected by a known distance). Since we know what the distribution of fragment sizes should be, we can use this as a constraint when evaluating the quality of our assembly. REAPR [29] is a tool that leverages this constraint and evaluates the accuracy of the assembly using read-pair coverage. REAPR determines the fragment coverage by first independently aligning the read-pairs to the assembly. A fragment is defined as the distance from the end points of proper read-pairs (those that are determined to have correct orientation and separated by the correct

distance). REAPR is able to find base-level errors by comparing the fragment coverage of a given base with the theoretical coverage.

Although the LAP score implicitly captures the quality of alignable read-pairs, REAPR provides assembler developers with a detailed breakdown of the specific errors in their assembly, giving the user the option to automatically break assemblies at locations of error. We use the specific error locations to calculate the percentage of error-free bases of the assembly.

4.3.2 Evaluating changes in assembly quality

Since the LAP is computed over a sample of the reads, for our experiments, we consider assemblies of equal quality if they are within one standard deviation of each other (see Section *Estimating the average read likelihood by sampling* in [28]). Users are free to select how large of a deviation they want to allow between assemblies. We also provide the user with the percentage of error-free bases in the assembly using results generated by REAPR. Both of these analyses are performed on the assemblies during the validation step of the assembly pipeline iMetAMOS [59, 87]. MetAMOS generates a summary HTML page for the assembly quality results.

4.4 Results

To illustrate the inadequacy of the *diff*-based approach, we first generate assemblies with SOAPdenovo and Minimus using bacterial sequences from a recent high-profile assembly competition, GAGE [20]. The *Rhodobacter sphaeroides*

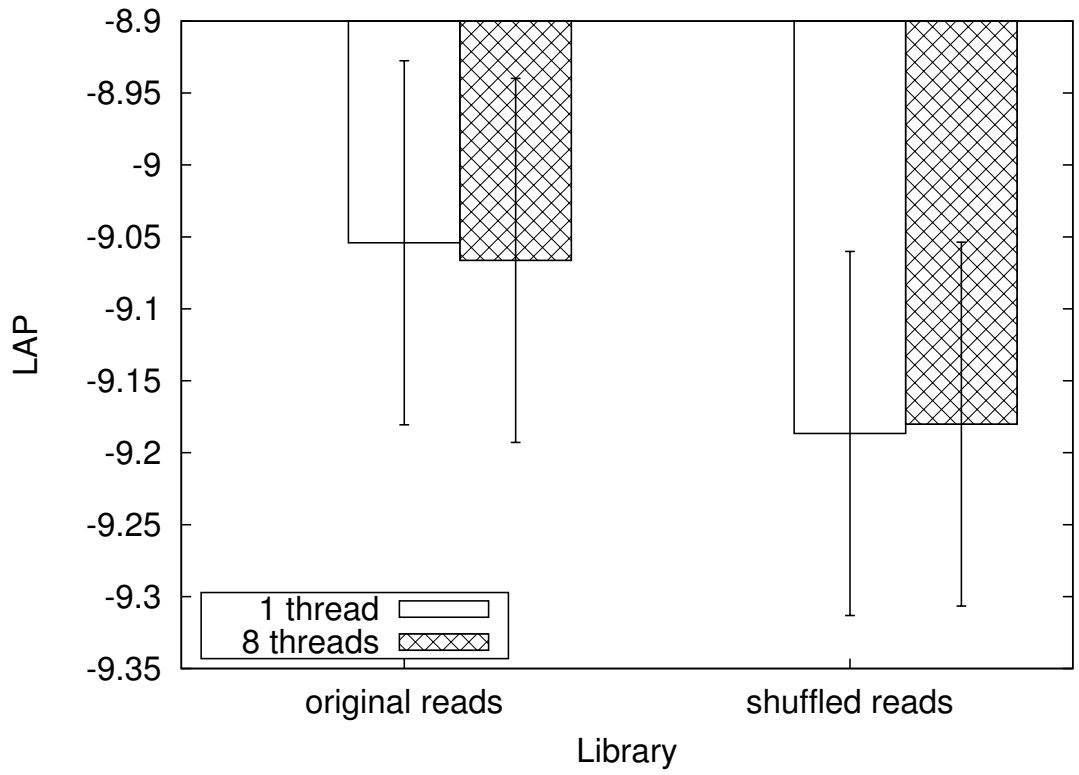


Figure 4.2: SOAPdenovo assemblies of the 1,408,188 *R. sphaeroides* error-corrected reads. The LAP was determined using a sample of 100,000 reads. The assemblies produced from the original reads and shuffled reads are within the acceptable standard deviation.

dataset contains 1,408,188 reads of length 101 bps. Assemblies are created using the original reads, along with a shuffled version of the original reads and compared using unix command **diff**. Since SOAPdenovo allows the use of multiple threads, we run the assembler using one and eight (the default in SOAPdenovo) threads. Both assemblers produce different assemblies depending on whether they use the original or shuffled reads. By doing a **diff** of the text outputs, we are unable to determine if there is a non-trivial change in assembly quality.

Next, we evaluate the *Rhodobacter sphaeroides* assemblies using our proposed

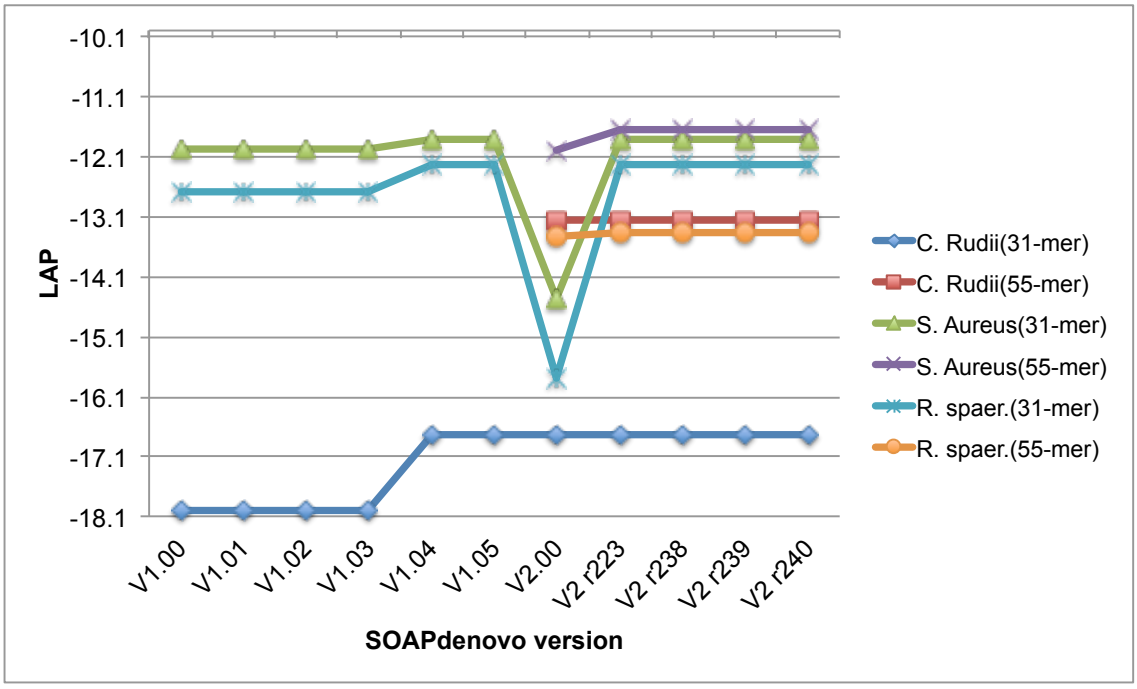


Figure 4.3: LAP scores for SOAPdenovo assemblies produced from different versions, using *R. sphaeroides*, *S. aureus*, and *C. Rudii* datasets. The LAP was determined using all reads. LAPs approaching zero represent more probable assemblies.

regression testing framework. The LAP is able to accurately detect trivial changes in assembly quality that raw text comparisons cannot (Fig. 4.2). For SOAPdenovo, the LAP of the assemblies generated from the original and shuffled reads are within the acceptable standard deviation. Furthermore, assemblies produced using 1 and 8 threads contain nearly identical LAPs.

The purpose of regression testing is to ensure that a code change does not introduce new faults, but our framework has the added benefit of detecting positive changes in assembly quality. Ideally, we want to use our framework alongside the development of an assembler to evaluate how code changes affect assembly qual-

ity. Fortunately, SOAPdenovo's source code and previous versions are available for download. We have created custom assembler specification files for each of the 11 assembler versions so that they can be run by MetAMOS [87]. We evaluate the different versions of SOAPdenovo using LAP and REAPR on the *Rhodobacter sphaeroides*, *Staphylococcus aureus*, and *Carsonella ruddii* datasets (Fig. 4.3). The *R. sphaeroides* dataset contains 762,266 Quake-corrected [88] read-pairs with insert sizes of 180bps. The *S. aureus* dataset contains 408,285 Quake-corrected read-pairs with insert sizes of 180bps. The *Carsonella ruddii* dataset contains 50,000 read-pairs with insert sizes ranging from 500bps to 3,500bps and comes packaged as test data for MetAMOS. We assemble the data using MetAMOS with our custom SOAPdenovo assemblers, then run the LAP and REAPR tools on the resulting assemblies at the contig-level. SOAPdenovo is a de Bruijn assembler and requires the user to specify a parameter, k-mer, to construct the de Bruijn graph. For each dataset we construct assemblies using two different k-mer sizes: 31 and 55. The default binaries for SOAPdenovo versions 1.00 - 1.05 were only able to build assemblies using k-mer sizes <= 31. Versions 2.00 and higher were used to process up to 127-mers.

The respective assemblies of each dataset using 31-mers are identical across SOAPdenovo versions 1 to 1.03. The changelists for versions 1.01 through 1.03 state that only bugs were fixed. The bug fixes in these versions are not covered by our test cases. The changelist for version 1.04 mentions an improved gap filling module, which is used during the scaffolding step of assembly. Our framework detects an increase in assembly quality between version 1.04 and the previous versions across the *S. aureus*, *R. sphaeroides*, and *C. Rudii* datasets. The LAP indicates a more

Table 4.1: Regression testing results for different SOAPdenovo versions using *S. Aureus* (31-mer) dataset. The percentage of error-free basepairs are calculated using REAPR. N50 is a commonly-used metric to measure contiguity.

Assembler	LAP	Error-free bps (%)	N50 (bps)
V1.00	-11.961	78.34	8,751
V1.01	-11.961	78.34	8,751
V1.02	-11.961	78.34	8,751
V1.03	-11.961	78.34	8,751
V1.04	-11.816	81.93	12,568
V1.05	-11.816	81.94	12,568
V2.00	-14.474	49.04	2,428
V2 r223	-11.816	81.62	12,568
V2 r238	-11.816	81.62	12,568
V2 r239	-11.816	81.62	12,568
V2 r240	-11.816	81.62	12,568

probable assembly was produced in 1.04. This positive change in assembly quality is supported by our REAPR results. The percentage of error-free bases increased from 78.34% to 81.93% and 65.28% to 72.75% in the *S. aureus* and *R. sphaeroides* datasets, respectively. A breakdown of the *S. Aureus* (31-mer) assemblies are given in Table 4.4. REAPR agrees with the LAP scores, showing a correlation with the percentage of error-free bases across the different versions of SOAPdenovo.

Interestingly, SOAPdenovo version 2.00 produces a less likely assembly for the *S. aureus* and *R. sphaeroides* datasets (using 31-mers) than the earlier versions. Our framework detects that this code change produced a lower quality assembly in terms of LAP and percentage of error-free bases, signaling that developers need to investigate further. SOAPdenovo versions beyond 2.00 appear to have fixed the issue resulting in the lower quality assemblies.

The quality of assemblies using 55-mers remain largely unchanged across the

version history. There was a slight increase in LAP of the *S. Aureus* assembly from versions 2.00 to 2r223, but there was only an increase of 0.01% in error-free bases.

Finally, we introduce faults into the core modules of the Minimus source code to evaluate how well our framework detects the resulting change in assembly quality (Fig. 4.4). Minimus consists of three core modules: `hash-overlap`, `tigger`, and `make-consensus`. `Hash-overlap` computes the overlaps between reads using a special type of hash seed called a *minimizer* [89]. The `tigger` uses the computed overlaps to assemble reads into individual contigs. The `make-consensus` module then improves the layout of the contigs using alignment data from the reads. For our tests, we seed faults into the `hash-overlap` and `tigger` modules and produce assemblies using the Influenza-A and zebrafish gene datasets packaged with Minimus. In order to find the shared regions of code executed between both datasets, we first obtain the code coverage (summarized in Table 4.2).

Table 4.2: Code coverage for Influenza-A and zebrafish gene test cases.

Testcase	Lines(Hit / Total)	Functions(Hit / Total)	Branches(Hit / Total)
Influenza-A	5724 / 46333 = 12.4%	3170 / 19019 = 16.7%	4333 / 21029 = 20.6%
Zebrafish	5606 / 46333 = 12.1%	3108 / 19019 = 16.3%	4247 / 21029 = 20.2%

We insert three faults into the `hash-overlap` module. The first fault allows all errors to be accepted between overlaps that contain a minimizer. Accepting all overlaps, regardless of quality, will increase the number of reads that can be combined. This fault produces an identical assembly as the fault-free version of the code in the Influenza-A dataset, thus is not detected by our framework. A noticeable

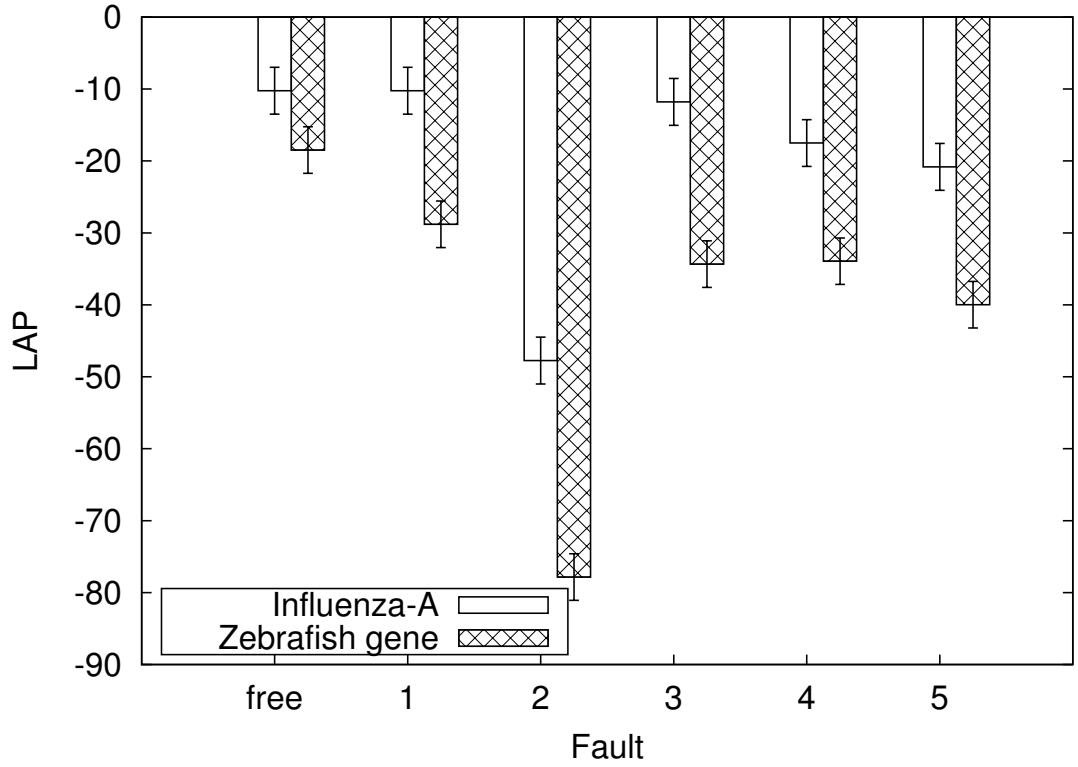


Figure 4.4: The LAP of fault-seeded versions of Minimus using the Influenza-A and zebrafish gene datasets. Faults 1, 2, and 3 were inserted into the **hash-overlapper** module and Faults 4, and 5 were inserted into the **tigger** module. All faults were detected in the zebrafish gene dataset; however, faults 1 and 3 were not detected in the Influenza-A dataset.

drop in assembly quality is detected in the zebrafish gene dataset.

The next two faults modify the functionality of the minimizers. Minimizers need to be sorted so a more computationally expensive dynamic programming algorithm can be used to connect them across mismatching sequence. Both faults attempt to break the initialization and sorting of the minimizers. The faults produced assemblies of lower quality in the zebrafish gene dataset; however, only one fault produced a lower quality assembly of the Influenza-A dataset.

We insert two faults into the `tigger` module. The first fault disrupts a method that hides transitive edges within the assembly graph. An edge between nodes A and C is transitive if there exists an edge between nodes A and B and an edge between nodes B and C. Without the ability to hide transitive edges, Minimus will encounter more nodes that have multiple outgoing edges. Minimus will be unable to compress these paths into unitigs, resulting in additional contigs. Our framework correctly detects the drop in assembly quality in both the zebrafish gene and Influenza-A datasets.

The second fault is related to the first, but breaks Minimus's ability to remove nodes from the graph that are contained by an overlap between two other nodes. Our framework correctly detects the resulting drop in assembly quality across both test datasets. The Influenza-A assembly produced using this faulted version has the same N50 size as the fault-free version of Minimus. The N50 size is the weighted median contig size, i.e., the length of largest contig c such that the total size of the contigs larger than c exceeds half of the genome size. Including this commonly-used metric serves as an example of the importance of using the LAP as the main criteria for accessing changes in assembly quality. The N50 metric would mislead developers into believing that the assemblers were of equal quality due to their nearly equal size.

4.5 Discussion

Regression testing without an oracle can potentially delay the release of software as developers attempt to track down a non-existent error due to differing results. In the worst case, developers may modify a correct program in order to reproduce incorrect results. Utilizing *assembly likelihood* (via LAP) and *read-pair coverage* (via REAPR), assembler developers will spend less time deciphering changes in assembly quality, allowing them to focus on algorithm improvements and other bug fixes. If a significant change in assembly quality is detected, the inclusion of REAPR provides developers with an additional breakdown of potential error locations in the assembly. Comparing the error locations between version histories may aid in tracking down sources of potential error within the code.

MetAMOS greatly simplifies the assembler regression testing process. Although all *de novo* assemblers require a set of reads as input, it is difficult to automate the assembly process utilizing multiple assemblers, since different parameters are used by different assemblers. Assembler parameters can change across software versions. It is common in large-scale assembly projects to combine the results of multiple assemblers in order to achieve what they believe is the best assembly. MetAMOS provides a standard generic assembler format for developers where they can specify how an assembler should run given a set of predefined keywords, such as *k-mer* length. This gives users the ability to specify a single parameter in MetAMOS, which is then automatically translated to the appropriate runtime parameter for the corresponding assemblers.

A frequently used strategy to test software without an oracle is to run the program on a simplified dataset for which the correctness of the result can be accurately determined [84]. In general, software is often tested this way, since exhausting testing is not practical. A simplified dataset may uncover some easy to find faults, but the complex cases are usually more error-prone. The Minimus test cases, Influenza-A and zebrafish gene, contain only 151 and 153 reads, respectively. However, typical assembly datasets consist of millions of reads, such as the *S. aureus* and *R. shpaeroides* datasets presented in Section 4.4. Two of the faults we seeded do not affect the assembly quality of Influenza-A dataset, but do affect the zebrafish gene assembly quality. The zebrafish gene dataset executes 0.8% and 0.2% and more code in the `hash-overlap` and `tigger` modules, respectively. Although the faults are inserted into shared sections of code, it is difficult to determine how much more complex the state of the assembler becomes due to the increase code execution.

Ideally, once a fault is discovered, a test case is added that exercises the code path containing the fault. Unit tests are one of way testing the method containing the fault, but the state of an assembler is often very large with many complex interactions. Thus, assemblers heavily rely upon end-to-end testing. However, it is difficult to modify existing test cases to exercise the fixed fault. Modifying the reads could have unforeseen side effects. Altered reads could produce new overlaps, changing the execution path of the code and potentially skipping the fixed fault.

Unlike Minimus, SOAPdenovo does not come packaged with a test set of sequences and assemblies. The changelist for the three versions following the initial release of SOAPdenovo only states that they, "*fixed some bugs.*" The details of

these bugs are not given, nor their affect on assembly quality using their in-house test data. In addition, users may employ different software/hardware configurations than those tested by the developers. It is crucial for the user to have the means to verify that they have correctly installed said software and are able to verify that the software is operating as the developers intended. Otherwise, results published from these users could lead to incorrect biological conclusions and misguided future studies.

4.6 Conclusion

Assembler developers face a difficult task: iteratively improving their assemblers to handle the exponential increases in biological data, while ensuring that changes at each step do not introduce any bugs. Traditional methods of comparing the text outputs of assemblers are unable to detect trivial differences in assemblies that are the result of using multi-core systems (a *requirement* to process increasingly large datasets) or the circular nature of bacterial genomes. We have developed a regression testing framework for genome assembly software that leverages existing assembly tools to accurately evaluate changes in assembly quality that traditional regression testing methods do not. We have examined the change in assembly quality over the version history of the popular assembler, SOAPdenovo. Lastly, our regression testing framework was able to detect manually inserted faults into the Minimus assembler.

Future work includes the addition of interactive visual analytics tools for

genome assembly to our regression testing framework. In cases where our framework detects non-trivial changes in assembly quality, it could be easier for the user to understand the differences if the assemblies were displayed visually.

4.7 Availability

Software to calculate the LAP is available for download at `assembly-eval.sourceforge.net`. REAPR is available for download at `http://www.sanger.ac.uk/resources/software/reapr/`. Both tools are available for automatic installation with MetAMOS (`https://github.com/treangen/metAMOS`). Sequence libraries are available GAGE assembler competition at `http://gage.cbcb.umd.edu/data/index.html`.

Chapter 5: Finding Metagenomic Mis-Assemblies

5.1 Introduction

Genome assembly of single organisms is made difficult due to the presence of sequencing errors and repeats. This difficulty is compounded in metagenomic samples due to the addition of varying organism abundances, intrapopulational variations, and conserved genomic regions between closely-related species. Since many downstream applications rely on these assembled genomes, it is critical that the assembly is error-free. Existing methods for finding mis-assemblies have primarily focused on single genome assembly and fall into two categories: reference-based and *de novo* evaluation.

Reference-based methods rely on having a collection of, often manually curated, reference genomes, while *de novo* methods look for inconsistencies between characteristics of the data generation process and the resulting assembly. QUAST [90] is a tool that can identify mis-assemblies and structural variants when provided with a reference genome. QUAST leverages existing methods (Plantagora [91], GeneMark.hmm [92], GlimmerHMM [93]) and quality metrics (GAGE [20]). QUAST uses the Plantagoras definition of a mis-assembly, i.e., a mis-assembly breakpoint is defined as a position in the assembled contigs where (1) the left flanking sequence

aligns over 1kb away from right flanking sequence on the reference, or (2) the sequences overlap by over 1kb, or (3) the right flanking sequence aligns on opposite strands or different chromosomes.

De novo techniques for detecting mis-assemblies in single genomes rely on looking for inconsistencies between the sequence generation process and the resulting assembly. In other words, given a model of the sequencing process, could the sequences have been generated if the assembly was the truth (inserted into the sequencing machine)? Regions of the assembly that do not meet these assumptions are signatures of potential mis-assemblies. One assumption is that the sequence generation process is roughly uniform, i.e., sequences starting at any position have equal probability. Substantially divergent coverage may indicate mis-assembly. If the sequences are paired-end or mate-pair then additional constraints based on insert size can be used.

Amosvalidate [18] is a *de novo* pipeline for detecting mis-assemblies that incorporates the above constraints. As mentioned in the previous chapter, REAPR [29] is a tool that leverages insert size constraints and evaluates the accuracy of the assembly using read-pair coverage. REAPR determines the fragment coverage by first independently aligning the read-pairs to the assembly. A fragment is defined as the distance from the end points of proper read-pairs (those that are determined to have correct orientation and separated by the correct distance). REAPR is able to find base-level errors by comparing the fragment coverage of a given base with the theoretical coverage.

Although all the above-mentioned tools were designed to work on single genomes,

they do not function correctly on metagenomic assemblies. QUAST relies on the existence of reference genomes, which are simply not available for most metagenomic samples. Furthermore, if the *correct* reference strain is not available, then QUAST may erroneously flag correct and biologically novel sequence as mis-assembled. The *de novo* tools REAPR and amosvalidate rely on global uniform sequence coverage to flag regions. In previous chapters, we have shown that contigs within the metagenomic assemblers vary widely in abundances. Assuming uniform coverage will cause these tools to erroneously flag regions as mis-assembled. In this chapter, we detail how to modify the constraints of existing tools to allow them to work with metagenomic assemblies. The result is VALET, the first *de novo* pipeline for detecting mis-assemblies in metagenomic assemblies.

5.2 Methods

5.2.1 Types of mis-assemblies

The majority of mis-assemblies fall into two categories: (1) compression/expansion of repetitive sequence and (2) sequence rearrangements. The first category of mis-assembly results when an assembler is unable to determine the correct copy count of repeats, leading to additional or fewer copies. The second category results when an assembler erroneously links separate unique portions of the genome that lie adjacent to a repeat. The repeat acts as a bridge joining the two separate parts of the genome together. Each category of mis-assembly has its own signatures that can be used to identify potential mis-assemblies.

The sequencing process of randomly-sheared fragments follows a Poisson distribution [94]. Regions within the assembly that show high variance in **depth of coverage** are a potential signature of compressed/expanded repeats, chimeric contigs, and other types of contamination.

The reads given to the assembler should be **alignable** to the resulting assembly. In practice, however, a read may fail to align for a few reasons. In metagenomic samples, an unaligned read can be from a rare, low coverage organism, and was never assembled with any other reads. A read with a large amount of errors will be unable to align within a specified similarity to the assembly. A read can be sequenced from a unfiltered contaminant or primer. If a read does not fall into one of the above categories, then it may be a sign of a potential mis-assembly.

Another signature that is used to find mis-assemblies relies on finding regions of the assembly that violate mate-pair **insert size constraints**. Certain sequencing technology allows researchers to sequence from the ends of a DNA fragment of a known insert size. Although the sequence technology can only give the raw sequence of the first couple hundred basepairs from the ends of the fragment, the distance between the ends of the sequences can be used to aid in resolving repeats, and orienting and scaffolding contigs. Regions of an assembly containing a disproportionate number of mate-pairs (reads from the same fragment) with incorrect insert sizes may be a potential mis-assembly.

VALET flags regions of the assembly based on (1) sampling depth, (2) alignability of the sequences, and (3) insert size constraints.

5.2.2 Estimating contig abundances using k-mers

An important part of most metagenomic pipelines is determining the relative abundance of each contig. The presence of repeats among different species poses a serious problem for estimating abundances. Short-read alignment tools such as Bowtie2 often randomly assign sequences that align equally well to multiple positions ignoring the relative abundances of the underlying contigs. This poses a *chicken or the egg* type problem because the sequence alignments are used to determine the contig abundances. Here we solve this problem by using the uniquely alignable sequences to establish an initial contig abundance. Then when we encounter a sequence that can align multiple locations, we randomly assign it based on the relative abundance of the corresponding contigs. We update the contig abundances and repeat the above step for a given number of iterations (30 by default). This approach is similar in spirit to that of Sailfish [95] which performs alignment-free abundance estimation of RNA-seq reads.

5.2.3 Depth of coverage analysis

In order to find regions of unexpectedly high/low coverage, we first learn the distribution of per-base coverages across a given contig. Using this distribution, bases are marked if their coverage falls below or above a certain threshold. We set the lower cutoff as the first quartile minus 1.5 $\times$ the interquartile range (IQR), which is the difference between the first and third quartile. 1.5 $\times$ IQR is the value used by Tukeys box plot [96]. Regions whose coverage is greater than the third quartile

plus 1.5 × IQR are marked as high coverage.

Using the per-base coverages may result in a large number of regions erroneously marked as mis-assemblies due to the inherent noisiness of the data, so we also provide a sliding window approach to smooth out the per-base coverages. The larger the window, the fewer the regions marked as mis-assemblies. VALET uses a window size of 300 bp by default.

5.2.4 Insert size consistency

VALET relies on the REAPR [29] pipeline to identify mate-pair insert size inconsistencies. REAPR works by first sampling the fragment coverage across the genome to get average fragment length and depth of coverage. Using this information, REAPR scans the assembly for observed regions that differ from the expected fragment length distribution and orientations.

REAPR is designed to work with single genome assemblies, more specifically, assemblies with a global uniform coverage. Since the contig abundances can vary drastically in metagenomic assemblies, VALET first bins contigs by similar abundances and then executes the REAPR pipeline on the binned contigs.

5.2.5 Identifying assembly breakpoints

Possible breakpoints in the assembly are found by examining regions where a large number of partial reads are able to align. We evenly split each unaligned read into *sister* reads. The *sister* reads are then aligned independently back to the

reference genome. We partition the provided assembly into bins (100 bp by default) and record which bins correspond to the sister reads. If we find a pair of bins that contain at least two different pairs of *sister* reads, then we mark it as a breakpoint location.

5.2.6 Comparing multiple assemblies

We visualize the quality of an assembly by recording the number of errors accumulated as we add contigs in decreasing order of length. This allows us to visually compare a set of metagenomic assemblies.

5.2.7 VALET pipeline

VALET takes as input a metagenomic assembly FASTA file and a collection of paired and un-paired reads (Figure 5.1). Assembled contigs are first filtered out based on size (2x the average read length by default). Next the abundances of contigs are calculated using our k-mer-based approach described above. Contigs undergo an additional filtering step based on abundance (10x by default). Higher coverage and longer sequence provide a better baseline for detecting mis-assemblies.

Once filtering has finished, regions of the assembly are flagged based on the inconsistencies described above. In practice, most mis-assembly signatures have high false positive rates which can be reduced by looking at regions where multiple signatures agree. Therefore, any window of the assembly (2000 in length by default) that contains multiple mis-assembly signatures are marked as **suspicious**. The flagged

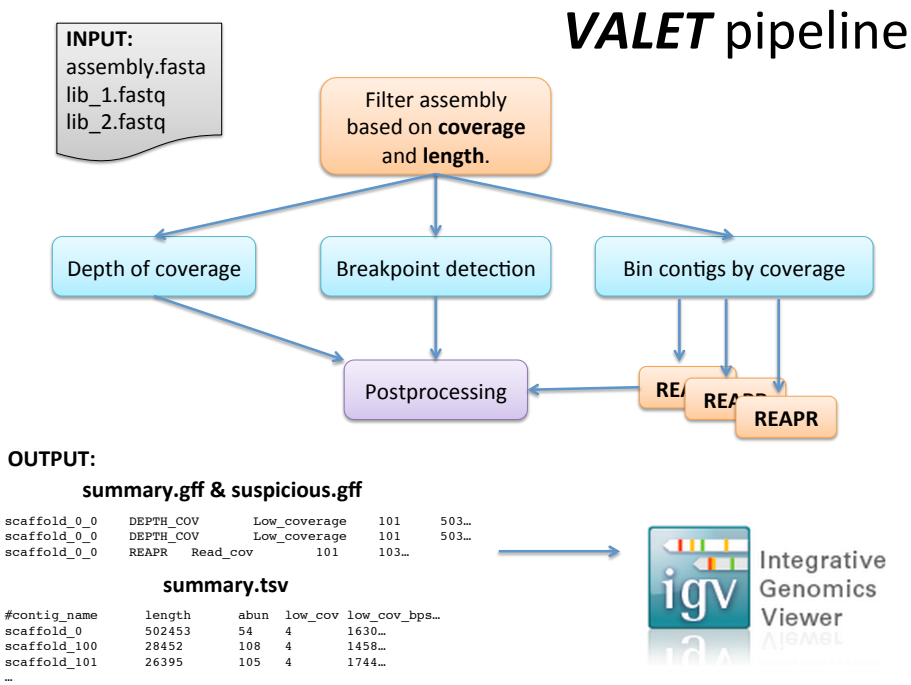


Figure 5.1: Overview of the VALET pipeline.

and suspicious regions are stored in a GFF file, which allows users to visualize the mis-assemblies using any genomic viewers, such as IGV [97].

5.3 Results

5.3.1 VALET achieves high sensitivity on a simulated metagenomic community

We examine the sensitivity of VALET on a toy simulated metagenomic community consisting of four bacteria at varying abundances: *Bacteroides vulgatus* (80x), *Bacillus cereus* (60x), *Actinomyces odontolyticus* (40x), and *Acinetobacter baumannii* (20x). Approximately four million reads are simulated using WGSIM [98] with

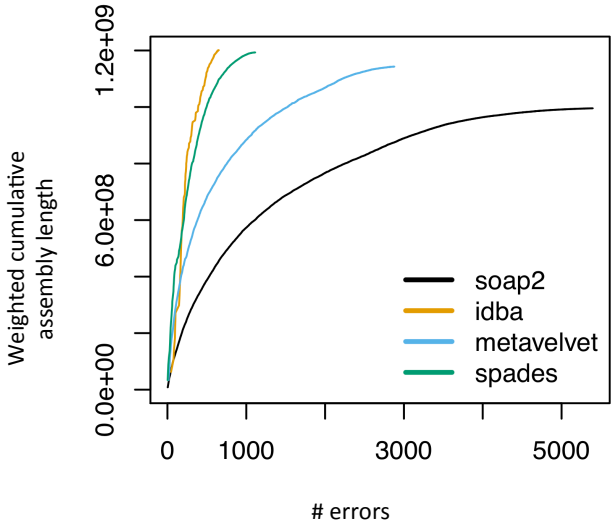


Figure 5.2: FRC plot provided by VALET of a simulated mock community.

default parameters. The dataset was assembled using IDBA-UD [99], MetaVelvet [50], SPAdes [100], and SOAPdenovo2 [30]. We ran VALET on the assemblies and compared the errors found with the reference-based mis-assemblies detected by QUAST (Table 5.1 and Figure 5.2). If any part of a region flagged by VALET overlaps with a mis-assembled region reported by QUAST, we consider it a true positive (mis-assembly correctly identified by our method).

Across all assemblers, VALET detects greater than 80% of mis-assemblies detected by QUAST.IDBA-UD has the greatest N50 after breaking the assembly at regions marked by QUAST (206.7 Kbp), followed by SPAdes (128.1 Kbp), MetaVelvet (29.5 Kbp), and SOAPdenovo2 (10.8 Kbp). These rankings match those provided by VALET (Figure 5.2).

5.3.2 VALET accurately evaluates assemblies of a synthetic metagenomic community

A major challenge of evaluating assemblies of environmental datasets is that a sizeable portion of the organsisms are unknown or lack a draft genome to compare against. In silico simulations often lack the complexity and sequencing biases present in real environmental samples. Fortunately, Shakya et al. provide a *gold standard* synthetic metagenomic dataset containing a mixture of 64 organisms (16 members of Archaea and 48 organisms from 18 Bacteria phyla) with complete or high-quality draft genomes and 200-fold differences in abundances [101]. The dataset consists of 53.4 million reads (101 bp in length). Due to the greater size and complexity of this dataset compared to the previous simulation, we assemble the dataset using two recent, fast metagenomic assemblers: Omega [102] and MEGAHIT [103]. We run VALET on the assemblies and compare the errors found with those reference-based mis-assemblies detected by QUAST (Table 5.3.2).

While the MEGAHIT and Omega assemblies are close in total size (192.3 Mbp vs. 194 Mbp, respectively), MEGAHIT has nearly twice as many contigs as Omega (19,145 vs. 10,284). QUAST detects far more mis-assemblies in the Omega assembly compared to MEGAHIT (56,917 vs. 770, respectively). VALET detects 34.80% and 96.10% of these mis-assemblies found by QUAST in the MEGAHIT and Omega assemblies, respectively. While Omega has a higher N50 than MEGAHIT (44.1 Kbp vs. 38.9 Kbp), after breaking at mis-assemblies, the N50 drops well below MEGAHIT's (11.9 Kbp vs. 33.5 Kbp), illustrating why the N50 metric is not always

a good indicator of assembly quality. VALET is able to accurately assess the quality of the two assemblies without using the reference genomes.

We investigate the high false positive rate by examining a small number of regions flagged by VALET, but not marked by QUAST within the MEGAHIT assembly. One contig, roughly 25 Kbp in size, had a 5 Kbp region at the start of the contig marked as high coverage (Figure 5.3). This region was roughly 4x the median coverage of the remaining contig. Using NCBI's BLAST [104] and reference database, the region aligned to the organism *Nostoc* sp. PCC 7120. Upon closer inspection, this region contained 16S, 23S, and 5S rRNA genes and was found at *four* locations in *Nostoc* sp. PCC 7120. This region was only found once in the assembly, so all the sequences from the repeats aligned to this region, inflating the coverage. This noticeable and consistent increase in coverage caused VALET to mark it as mis-assembled. Unsurprisingly, QUAST did not mark this as a mis-assembly because the actual sequence within this region matched the reference.

						Mis-assembly signatures			Suspicious regions			
Assembler	Len (Mbp)	Ctgs	N50 (Kbp)	NA50	Errs	Num	Valid	Sens	Num	Valid	Sens	Mismatches per Kbp
IDBA-UD	16.5	200	208.8	206.7	36	804	36	100.00%	25	8	22.20%	23.95
MetaVelvet	16.3	1,117	29.5	29.5	21	2,802	19	90.50%	4	2	9.50%	35.52
SPAdes	16.4	330	130.9	128.1	37	1,117	31	83.80%	17	4	10.80%	22.43
Soapdenovo2	12.3	2,161	10.8	10.8	1	4,983	1	100.00%	2	0	0%	13.37

Table 5.1: VALET results for simulated mock community consisting of four bacteria at varying abundances: *Bacteroides vulgatus* (80x), *Bacillus cereus* (60x), *Actinomyces odontolyticus* (40x), and *Acinetobacter baumannii* (20x). Reads were assembled using the four provided assemblers. General assembly statistics include length in Mbp (Len), number of contigs (Ctgs), N50 contig size (N50), N50 of contigs after broken at mis-assemblies (NA50), number of errors detected by QUAST (Errs), number of flagged regions by VALET (Num), number of flagged regions that overlap an error found by QUAST (Valid), sensitivity (Sens), and mismatches per Kbp.

						Mis-assembly signatures			Suspicious regions			
Assembler	Len (Mbp)	Ctgs	N50 (Kbp)	NA50 (Kbp)	Errs	Num	Valid	Sens	Num	Valid	Sens	Mismatches per Kbp
MEGAHIT	192.3	19,145	38.9	33.5	770	30,377	268	34.80%	2,239	100	13.00%	92.24
Omega	194	10,284	44.1	11.9	56,917	1,425,127	55,108	96.10%	17,758	13,935	96.80%	98.55

Table 5.2: VALET results for assemblies of the Shakya et al. [101] dataset. General assembly statistics include length in Mbp (Len), number of contigs (Ctgs), N50 contig size (N50), N50 of contigs after broken at mis-assemblies (NA50), number of errors detected by QUAST (Errs), number of flagged regions by VALET (Num), number of flagged regions that overlap an error found by QUAST (Valid), sensitivity (Sens), and mismatches per Kbp.

5.4 Discussion

In practice, VALET has high sensitivity for mis-assembly detection, but also a high false positive rate. While we can tune parameters, such as window size, to trade-off between the measures, the high false positive rate still remains fairly prevalent. Some of the false positives can be explained as the assembler deduplicates repetitive regions of the genome, e.g., the ribosomal genes. This highlights an important issue prevalent in metagenomic assemblers. In the Shakya et al. dataset, a more *correct* *Nostoc* sp. PCC 7120 assembly would include an additional contig consisting solely of the ribosomal genes. Then during the abundance estimation step of VALET, a quarter of the sequences would align to the original contig due to the flanking unique region and the remaining three quarters would align solely to the new contig containing the ribosomal genes. VALET would no longer mark this region in the original contig.

Assemblathon1 [19] has stated that assemblers have trouble with polymorphism and heterozygosity. This problem is compounded in metagenomic assemblies due to closely-related strains having uneven abundances. MetaCompass [105], a reference-based metagenomic assembler, was used to assemble the HMP Sample SRS024655 (retroauricular crease of a male). A 25 Kbp region was flagged as having low coverage by VALET, but not reported by QUAST (Figure 5.4). After further investigation, the 25 Kbp region belonged exclusively to the one of the reference genomes chosen by MetaCompass: *Propionibacterium acnes* KPA171202. The higher coverage flanking regions aligned to both *Propionibacterium acnes* KPA171202

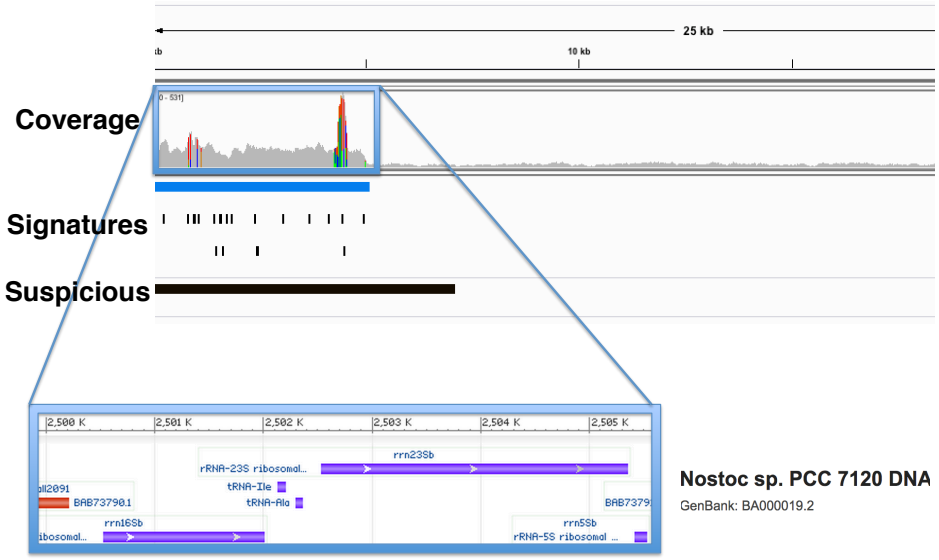


Figure 5.3: A closer examination of a region flagged by VALET, but no mis-assembly reported by QUAST. This region contained 16S, 23S, and 5S rRNA genes and was found at *four* locations in the *Nostoc* sp. PCC 7120 genome.

and *Propionibacterium acnes* ATCC 11828. *Propionibacterium acnes* KPA171202 contains nearly 70 Kbp of insertions. Despite being found at a lower abundance, the KPA171202 strain of *Propionibacterium acnes* was chosen for the reference-guided assembly because all reads that were align to the ATCC 11828 strain also aligned to the KPA171202 strain. Since the KPA171202 strain was actually found in the dataset, QUAST detected no structural errors. A more *correct* assembly would include both complete genomes.

5.5 Conclusion

VALET is the first *de novo* pipeline for detecting mis-assemblies in metagenomic datasets. VALET searches for regions of the assembly that are statistically inconsistent with characteristics of the data generation process. VALET finds mis-assemblies on a simulated and synthetic metagenomic mock community.

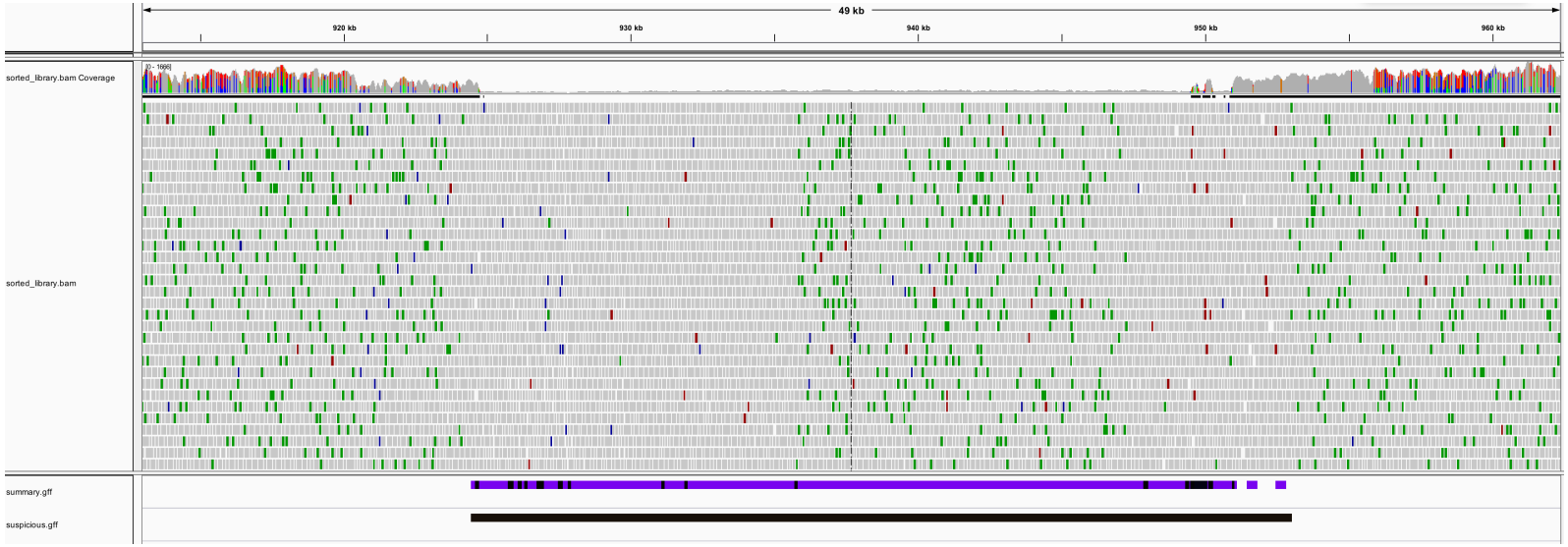


Figure 5.4: A 25 Kbp low coverage region flagged by VALET, but no mis-assembly reported by QUAST. The low coverage region was due to MetaCompass selecting only a single strain of *Propionibacterium acnes* to use for assembly instead of both.

Chapter 6: Additional Contributions

During my time at the University of Maryland, I have had the privilege to work on a wide array of interesting problems in key areas of bioinformatics. The ever-increasing amount of sequencing data poses a challenge to commodity hardware both in terms of storage and analysis. In this chapter, I describe my contributions to the fields of lossy compression and clustering. In Section 6.1, I show how we use lossy compression algorithms to greatly reduce the required storage for next generation sequencing data with little effect on downstream analyses. In Section 6.2, I show how we leverage the power of cloud computing to cluster sequencing data and speedup sequence alignment. I mentored two undergraduate students to complete this project.

6.1 Lossy Compression of DNA Sequence Quality Values

6.1.1 Abstract

The FASTQ file format has become the *de facto* standard for storing next-generation sequencing data, containing nucleotide information along with a quantitative measure of the reliability of individual base calls. As the cost of sequencing

continues to decrease, the rate of sequencing data production is increasing, requiring efficient ways of storing and transferring this vast amount of data. Most methods on sequencing data compression focus on compressing nucleotide information without any loss of information. Quality data, however, have different properties than nucleotide data, and methods compressing nucleotide sequences efficiently do not perform as well on quality sequences. Furthermore, while lossless representation is necessary for nucleotide sequences, it is not an essential requirement for quality values.

Existing methods for compressing quality sequences mostly focused on minimizing the loss of information with less emphasis on effects on subsequent analyses. In this chapter, we evaluate several different compression methods for quality values that compromise accuracy for better storage efficiency, and their resulting impact on common bioinformatic analyses using sequence read data.

Lossy compression of quality information can greatly decrease storage and memory requirements with little discernible effects on subsequent analysis results. The three compression strategies in this study were able to produce similar results to those obtained with uncompressed quality sequences in terms of quality control, genome assembly, and alignment of short read to a reference sequence.

6.1.2 Introduction

Read data from high-throughput sequencing constitutes the largest category of data in genomics research because of great redundancy, inclusion of quality values,

and read-level naming and metadata. Because of this abundance effective compression of read data has the potential for substantial improvement in data storage and transfer efficiency.

Quality values comprise a standard component of FASTQ files [106], a very common format for sequence read data. At the level of sequence read the probability of error for each base-call is typically represented by PHRED quality value, which is defined as $Q = -10\,log_{10}P$ [107]. Depending on the sequencing technology these quality values can range from 0 to 93, and are represented with the ASCII characters 33 to 126 (with some offset). There is a single quality value per base-call for Illumina sequence reads.

Quality values can be used throughout bioinformatics pipelines. Among the most fundamental uses of sequence quality values is as part of the quality assessment and quality control (QA/QC) processes prior to subsequent analysis steps. Quality control based on quality values generally includes two operations: *i.* filtering, the elimination of reads that on the whole do not meet arbitrary quality standards, which reduces the total number of reads; and *ii.* trimming of low quality base-calls from reads, which reduces the number total number of bases. Quality values can be used by genome assembly software to produce better assemblies [108, 109]. Short-read alignment software, such as Bowtie2 [38], use quality values to weight mismatches between read and reference sequences. Software for detecting single nucleotide polymorphisms (SNPs) can use quality values [110], and identified SNPs with high-quality calls are deemed more reliable than those with low-quality calls, particularly in low coverage regions.

Previous literature on sequence data compression has largely focused on lossless compression of base calls [111–120]. Among the several challenges for compression of read data is dealing with different error profiles resulting from differences in underlying chemistries, signal detection and processing mechanisms, inherent biases, and other idiosyncratic properties of distinct high-throughput sequencing technologies. Sequence reads generated by instruments such as an Illumina HiSeq, the focus of this research, are characterized by having relatively few insertion and deletion errors, but substitution (miscall) errors are much more frequent and have context-specific patterns. These errors are non-uniformly distributed over the read length (e.g., error rates increase up to $\sim$16$\times$ at the 3$'$ end, and 32.8 – 67.9% of reads have low quality tails at the 3$'$ end [121]).

Although we recognize the need for lossless compression for some purposes and contexts (e.g., archiving, provenance), our perspective is largely pragmatic with a focus on the use of quality values in subsequent analyses. From this perspective some loss of information is deemed acceptable if the inferences from analyses are relatively unaffected. Here we describe our research investigating lossy compression of sequence read quality values, specifically those associated with Illumina instruments, with the objective to provide some perspective on several strategies rather than to develop a robust high-quality software for use. Recognizing these properties of Illumina sequence reads motivates our exploration of three general classes of lossy compression methods – binning, modeling, and profiling – and consider an exemplar of each class. [112] and [114] evaluated the effects of lossy compression on identifying variants within a dataset. We build on these prior works and access the

effects of quality value information loss resulting from compression on additional subsequent genomic analyses including read preprocessing (filtering and trimming), genome assembly, and read mapping.

6.1.3 Methods

6.1.3.1 Compression strategy: binning

Quality values can be binned, and the minimum number of bins that allows for a any distinction among quality values is two, i.e., two categories "good" and "bad" quality. We implement 2-bin encoding by setting a quality value threshold empirically determined by the distribution of quality values across reads. Base-calls are marked "bad" if their quality value falls below the first quartile minus 1.5 $\times$ the interquartile range (IQR), which is the difference between the first and third quartile. 1.5 $\times$ IQR is the value used by Tukey's box plot [96]. The main benefit of this approach is that it is completely data-dependent, and no assumptions regarding the distribution of the quality values need to be made.

With 2-bin encoding binary encoding is possible, allowing us to use a single bit to represent the quality of a base instead of the standard 8 bits used to store quality values in ASCII. An additional benefit of 2-bin encoding is the potential for increased adjacency of identical values and repeating patterns, properties that may increase effectiveness of subsequent compression using established algorithms [122–124].

The economic costs of memory use for binning, in general terms, include no fixed costs, and marginal costs that are a function of the number of base-call quality

values times the cost of the encoding.

[118] provide three similar lossy compression strategies based on binning the base error probability distribution: UniBinning, Truncating, and LogBinning. UniBinning evenly splits the error probability distribution into a user-defined number of partitions. Truncating treats a user-defined number of highest quality values as a single bin. LogBinning works similar to UniBinning, except with the *log* of the error probability distribution, which effectively bins the ASCII quality values evenly. Our 2-bin encoding is a combination of LogBinning and Truncating in that we are placing the highest quality values (as defined above) of the log of the error probability distribution into a single bin.

6.1.3.2 Compression strategy: modeling

If quality values are modeled, compression is conceivably possible by replacing the original quality values by a representation of the model. For example, quality values can be conceptualized as bivariate data with the ordered nucleotides (1 to read length) representing the abscissa, and quality values representing the ordinate. In this research we model read quality values as polynomial functions obtained with least-squares fitting, as one approach to compression read quality values by modeling.

Despite the fact that polynomial functions have significantly lower number of parameters (i.e. one to six coefficients) than a read-length string of raw quality values, the necessity of using floating point numbers to store coefficients greatly limits

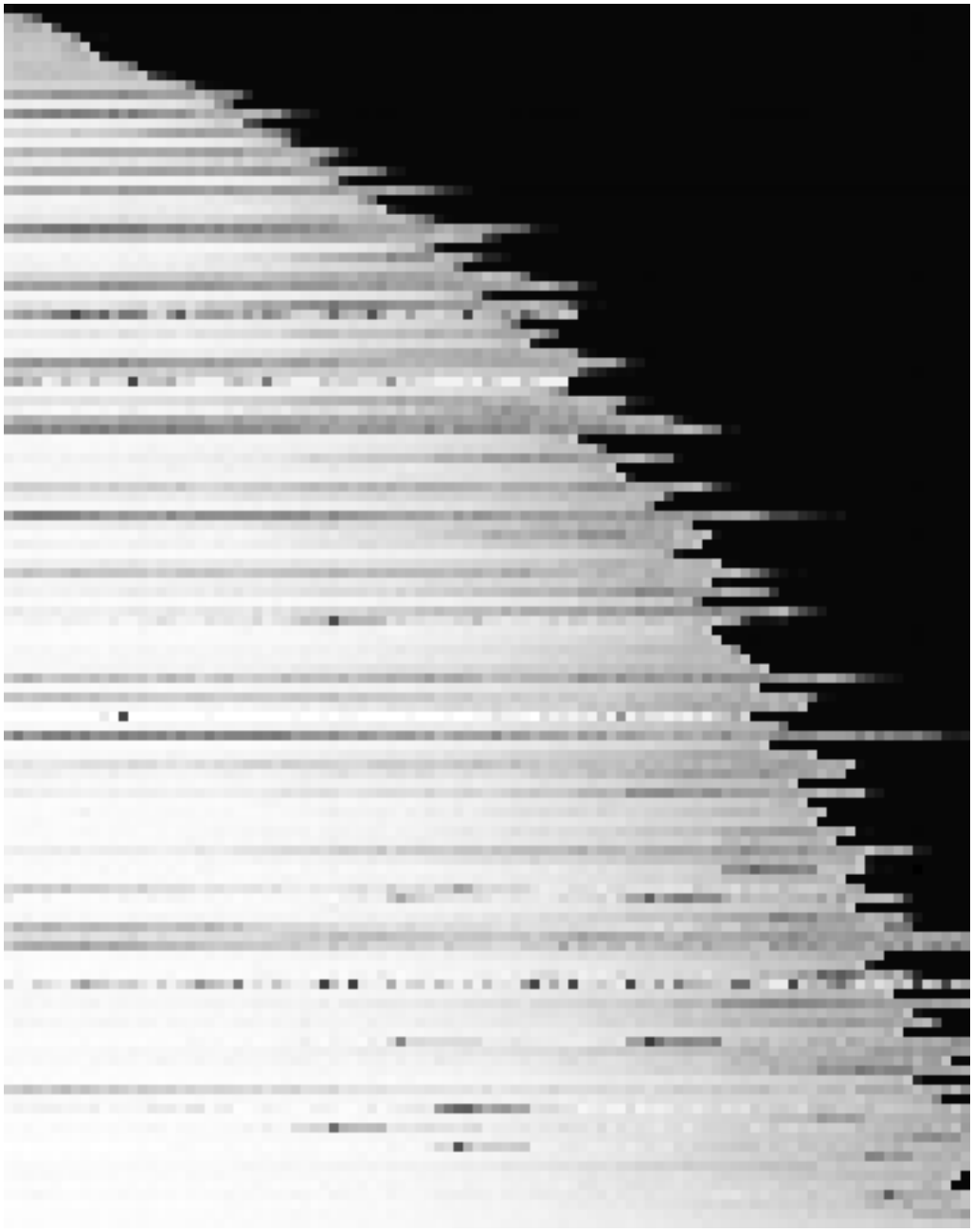


Figure 6.1: Quality profiles obtained by k-means clustering on the fragment library from *Rhodobacter sphaeroides* 2.4.1 data set using $k = 128$, with each row corresponding to a quality profile. Dark to light colors represent low to high quality values. It is readily visible that the two most distinctive features of quality profiles is their drop-off position and average overall quality. One can also see sporadic low-position values in a handful of profiles, likely capturing intermittent problems in the sequencing process affecting thousands of reads at a time.

the compression potential of the method. In order to get meaningful compression on single-precision four-byte floating point numbers, one would have to relax on the least-squares approximation constraint to obtain compressible values on the byte level which is outside the scope of this study.

The economic costs of memory use for model-based compression, in general terms, include no fixed costs, and marginal costs that are a function of the number of reads times the cost representing the model parameters.

QualComp is a lossy compression tool that models quality values as a multivariate Gaussian distribution, computing the mean and covariance for each position in the read [116]. Once the model parameters are calculated, they are stored by the decoder to later reconstruct a *representative* quality value. QualComp takes as input a user-specified rate (bits per read) and then poses an optimization problem of how to allot these bits for a given position while minimizing the MSE. The quality values can be clustered before hand to produce more accurate models.

6.1.3.3 Compression strategy: profiling

As large sets of quality strings show similar trends of quality over their length, it makes sense to identify such common patterns in the data and use them as reference profiles to approximate individual sequences of quality values. Such patterns can be readily determined by clustering data points (i.e. quality vectors) and using the resulting cluster centers as representative profiles.

k-means clustering is a vector quantization method, partitioning a set of sam-

ples into k sets that minimize within-cluster sum of squares [125]. Using a random subset of read quality values, a compression method can use the computed cluster centers as read quality profiles. As the problem is NP-hard, we use a heuristic iterative refinement approach by quickly converging to a locally optimal minimum provided by R [126].

First, the method samples an adjustable number of reads randomly from the file to be used as a training set. Quality values are represented by vectors containing their PHRED scores corresponding to each position along the read. Subsequently, k-means clustering is performed on the training set until convergence. The obtained cluster centers will be the quality profile prototypes for the dataset.

Once the k quality profiles are determined, all reads are passed through the trained k-means predictor, with the nearest quality profile in Euclidean space being assigned to every read as their compressed representation.

The compressed quality file therefore consists of an index enumerating the k quality profiles, and a binary part containing the assigned quality profile index for each read.

Although this approach is not able to capture randomly occurring outlier quality values, it ensures that the overall trends in quality value patterns are retained. Quality profiles capture different overall qualities and different drop-off positions and gradients. An example of 128 quality profiles are shown on Figure 6.1.

The economic costs of memory use for profile-based compression, in general terms, include fixed costs associated with representing the profiles, which is a function of the number of profiles times the cost of encoding them, and these fixed costs

are amortized over the entire set of reads to which they apply. Additionally there are marginal costs that are a function of the number of reads encoded.

6.1.3.4 Datasets

We used several Illumina sequence read datasets in this research, which are taken from data from the GAGE (Genome Assembly Gold-Standard Evaluations) [**?**] except as noted. These datasets are as follows.

Rhodobacter sphaeroides 2.4.1, which are generated from a fragment library (insert size of 180 nt; 2,050,868 paired-end reads) and short-jump library (insert size of 3,500 nt; 2,050,868 reads). The corresponding reference sequence was obtained from the NCBI RefSeq database (NC_007488.1, NC_007489.1, NC_007490.1, NC_007493.1, NC_007494.1, NC_009007.1, NC_009008.1).

Homo sapiens chromosome 14 data, which are generated from a fragment library (insert size of 155 nt; 36,504,800 paired-end reads) and short-jump library (insert sizes ranging from 2283-2803 nt; 22,669,408 reads). The corresponding reference sequence was obtained from the NCBI RefSeq database (NC_000014.8).

Escherichia coli str. K-12 MG1655 MiSeq data was downloaded from `http://www.illumina.com/systems/miseq/scientific_data.html`, which are generated from a fragment library (insert size of 180 nt; 1,145,8940 paired-end reads). The corresponding reference sequence was obtained from the NCBI RefSeq database (NC_000913.2).

Mus musculus data was downloaded from `http://trace.ddbj.nig.ac.jp/`

DRASearch/run?acc=SRR032209, which consisted of 18,828,274 reads of length 36.

6.1.3.5 Performance evaluation

As a measure of compression effectiveness we use bits/base-call, and define it as the size of the compressed representation of quality values (in bits) divided by the number of quality values represented. As a measure of information loss we use mean squared error (MSE) as a loss function, and define it as $\frac{1}{n}\sum_{i=1}^{n}(Q'_i - Q_i)^2$, where n is the number of sequences, Q'_i is the compressed/decompressed quality value, and Q_i is the original quality value associated with sequence position i.

We evaluate effects of information loss from quality value compression on quality control steps of read filtering and trimming, which were performed using Sickle [127], and make comparison to uncompressed data.

We evaluate effects of information loss from quality value compression on *de novo* genome assembly performance using contiguity statistics, log average read probability (LAP) [28], and a collection of reference-based metrics. The contiguity statistics include N50, which is defined as the median contig size (the length of largest contig c such that the total size of the contigs larger than c exceeds half of the assembly size) and corrected N50, which is the recalculated N50 size after the contigs are broken apart at locations of errors. The LAP score can be viewed as a log likelihood score, where a value closer to 0 is better. We use a script provided by GAGE reference-based evaluation to count single nucleotide polymorphisms (SNPs), relocations, translations, and inversions. The reference-based metrics are normalized

by the length of the assembly to facilitate comparison. For the genome assembly we used software that makes use quality values in the assembly process: ALLPATHS-LG [109] version r50191 with default settings and 32 threads.

6.1.4 Results

6.1.4.1 Compression effectiveness versus information loss

We compare the MSE versus bits/base-call of the *Rhodobacter sphaeroides* 2.4.1, *Homo sapiens* chromosome 14, *Escherichia coli* str. K-12 MG1655, and *Mus musculus* datasets (Figure 6.2). We only include the fragment libraries for the *Rhodobacter sphaeroides* 2.4.1, and *Homo sapiens* chromosome 14 data sets, but the additional short-jump library results are available in the Supplementary of the submitted manuscript. Storing the uncompressed quality values requires 1 byte per base-call because they are stored in ASCII format and is denoted by the dotted black asterisk in the figure. The lossless compression of each dataset using BZip2 ranges from 2.19 - 3.10 bits/base-call and is denoted by the colored asterisks on the abscissa. The compression methods tend to cluster together across the different datasets. Across all datasets, the 0-degree polynomial regression, profile encodings, and QualComp have the lowest bits/base-call.

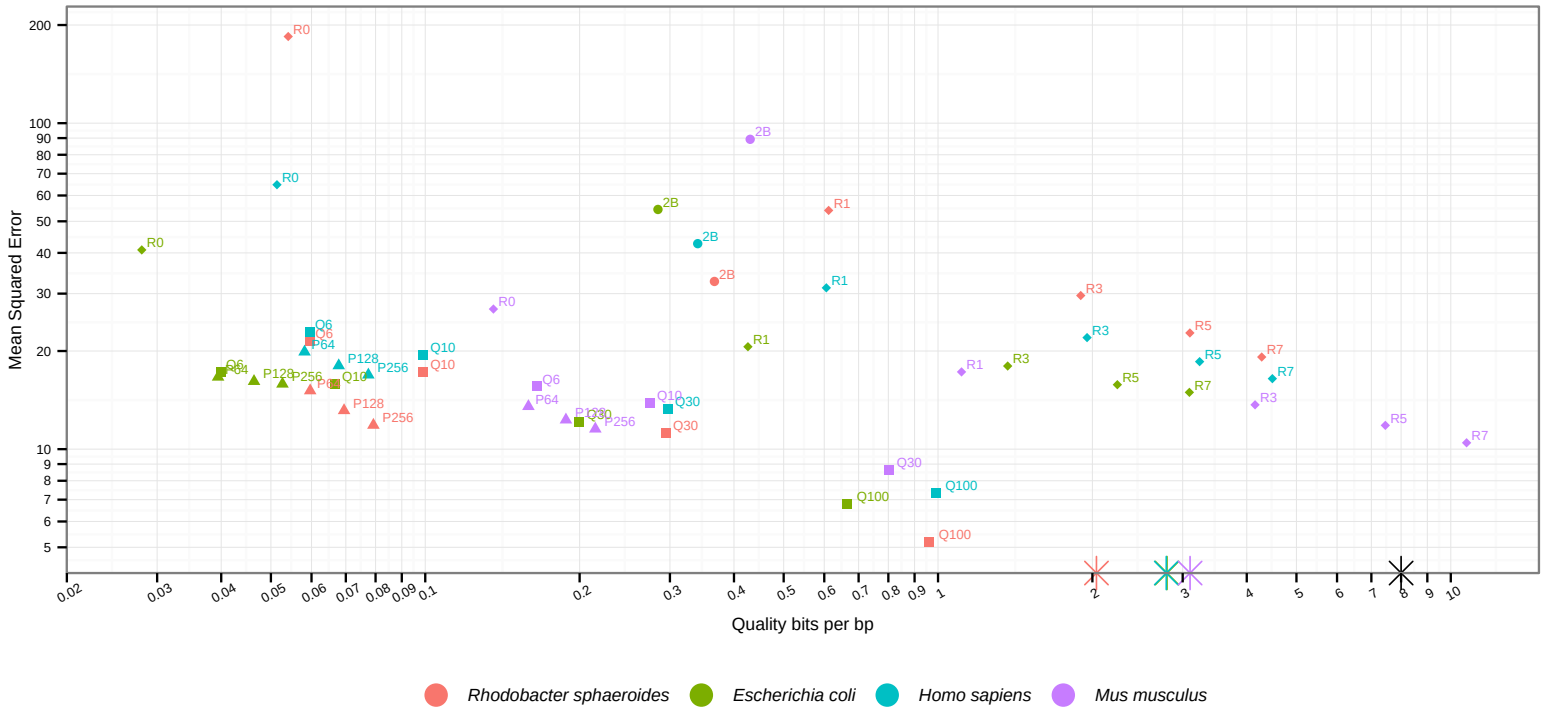


Figure 6.2: Mean squared error versus bits/base-call for different compression methods applied to the *Rhodobacter sphaeroides* 2.4.1, and *Homo sapiens* chromosome 14 fragment libraries, and *Escherichia coli* str. K-12 MG1655, and *Mus musculus* datasets. 2B — 2-bin encoding; Pn — profiling with n profiles; Rn — modeling with polynomial regression models of degree n; Qn — QualComp with rate parameter of n. Asterisks denote the corresponding lossless compression using BZip2, with the black asterisk corresponds to original uncompressed data.

QualComp with the rate parameter set to 100 bits/read has the lowest MSE, but requires 10-15x more storage than the profile encoding methods for only a 2-3x reduction in MSE. When QualComp's rate parameter is set to match the profile encoding methods, QualComp performs slightly worse in terms of MSE. In the *Rhodobacter sphaeroides* 2.4.1 fragment library, QualComp with rate 10 bits/read (0.099 bits/bp) has a MSE of 17.29. Using 256-profile encoding requires less storage (0.079 bits/bp) and has a lower MSE (11.85).

As the order of the polynomial increases, the bits/base-call increase and the MSE decreases at an exponential rate. The 7th-degree polynomial regression has the highest bits/base-call and in the *Mus musculus* dataset, and requires more storage than the ASCII original quality values. A 7th-degree polynomial requires storing eight floating point values, resulting in 32 bytes per sequence of quality values. The read length of the *Mus musculus* dataset is only 26, so the 7th-degree polynomial regression is storing six more bytes than necessary for lossless encoding the quality data.

6.1.4.2 Effects on sequence read preprocessing

The majority of compression methods retain more base-pairs after preprocessing than the uncompressed sequences (Figure 6.3). In general, as a given compression model increases in complexity, i.e., as the number of profiles, polynomial degrees, or rate increases, the amount of base-pairs kept approaches the number base-pairs kept using the uncompressed sequences. The compression methods on the *Mus musculus*

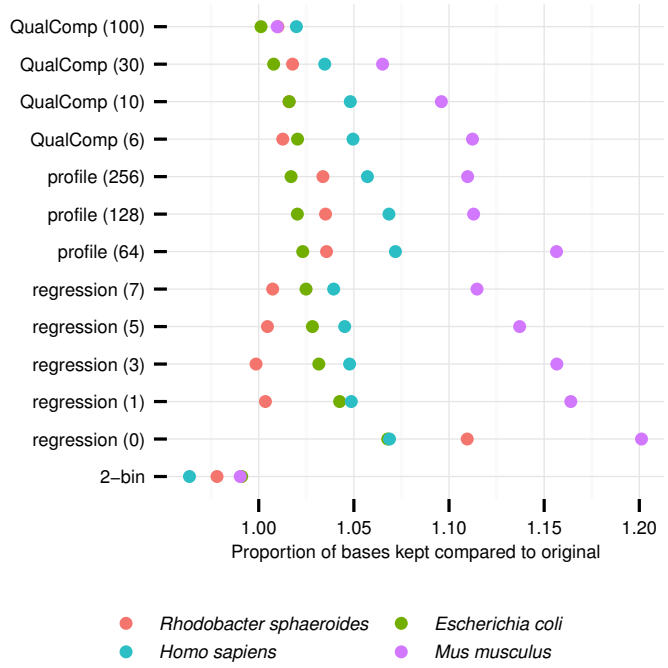


Figure 6.3: Preprocessing results of *Rhodobacter sphaeroides* 2.4.1, and *Homo sapiens* chromosome 14 fragment libraries, and *Escherichia coli* str. K-12 MG1655, and *Mus musculus* datasets. Sequences were trimmed using Sickle. The total amount of bases filtered by each compression method is compared with the amount of bases filtered using the uncompressed sequences.

dataset have the greatest proportion of retained base-pairs compared to the uncompressed sequences. The *Escherichia coli* str. K-12 MG1655 MiSeq dataset has the smallest range.

The 2-bin approach is the only compression method that results in a higher number of filtered base-pairs across all datasets. Sickle uses a sliding window approach to smooth the read quality values before it trims the sequence based on a specific quality threshold. In the 2-bin approach, there is not an even distribution of

values per bin. In other words, *bad* quality values may range from 0-33, whereas *good* values may only range from 34-40. Thus, mid-range quality values that are above the threshold (20 by default) are set below the quality threshold when compressed, resulting in an increased number of filtered bases.

The 0-degree polynomial regression results in the highest proportion of bases kept. If the mean quality value of the read is above the filtering threshold, then no base-pairs are trimmed. Only reads that are comprised of mainly low quality values will be filtered.

It is important to highlight that the even though a compression method may result in the same number of filtered base-pairs as the uncompressed sequences, it does not mean the *same* base-pairs were filtered. The 1st-degree and 5th-degree polynomial regression of the *Rhodobacter sphaeroides* fragment library filters nearly as many bases as each other. However, if we examine the specific reads filtered, the 5th-degree polynomial regression discards approximately two thirds less reads than the 1st-degree polynomial regression that are kept by the uncompressed reads (4,640 and 12,066 reads, respectively).

6.1.4.3 Effects on genome assembly

No assembly of the *Rhodobacter sphaeroides* 2.4.1 dataset outperforms all others in all metrics (Table 6.4). Among the compression methods, the 256-, 64-, 128-profile encoding had the highest ranks, followed by QualComp with rates 10 bits/read, 30 bits/read, 100 bits/read, then 7th-degree polynomial regression, fol-

lowed by the QualComp with rate 6 bits/read, the 2-bin encoding, and lastly, the 3rd-degree, 5th-degree, and 0-degree polynomials.

The lossy compression methods largely preserve the contiguity found in the assembly produced using the reads with unmodified quality values. All compression methods other than 0-degree polynomial regression produce an N50 ranging from 3.17–3.22 Mbps (see Supplementary of manuscript). Despite the similar contiguity statistics, the different compression methods vary noticeably in the amount of SNPs. The order of polynomial has an inverse relationship with the amount of SNPs detected. The 2-bin and profile methods detected the least amount of SNPs compared to the reference genome, outperforming the assembly using the original quality values. A more in-depth evaluation is needed to determine whether these compression methods are missing actual SNPs.

It is important to highlight that using uncompressed reads does not produce the best assembly in terms of any of the metrics. The uncompressed reads scores worse than the top overall assembly (256-profile encoding) in terms of assembled bases, missing reference bases, N50, SNPs, indels >5bp, and relocations. The assembly using uncompressed reads has an error rate of roughly 8.75 errors per 100 kb of assembled sequence, while the 256-profile encoding has an error rate of 8.02 errors per 100 kb.

In general, the greater the polynomial order, the better overall assembly; however, the 5th-degree polynomial regression performs slightly worse than the 3rd-degree polynomial. The respective ranks in terms of N50 and relocations are fairly distant, which lowers the overall ranking of the 5th-degree polynomial slightly below

that for 3rd-degree polynomial model. The 1st- and 0-degree polynomial regression methods perform poor in all metrics except assembled bases. One explanation is that the high error portions of reads are being marked as high quality, so ALLPATHS-LG is unable to trim or error correct the sequences. Assembled sequences that overlap maybe unable to align across the errors at the end of the reads, artificially inflating the assembled genome size.

Among the different QualComp rate parameters, the 10 bits/read rate ranked highest overall, outperforming the other rate parameters in terms of corrected N50, least missing reference bases, SNPs, and indels >5bp. With the exception of the 6 bits/read rate, the assemblies decrease in rank with the increase in the rate parameter in terms of corrected N50, and least missing reference bases. This trend runs counter to the the decrease in MSE of the different rates.

6.1.4.4 Effects on read mapping

Certain short read alignment tools use the quality value information when finding potential alignments. Bowtie2 (version 2.2.3) was used to evaluate the different decompressed FASTQ files. Bowtie2 uses quality values written in the FASTQ files when mapping reads against a reference genome. The original uncompressed and decompressed FASTQ files were mapped with Bowtie2 against *Rhodobacter sphaeroides* reference genome. The generated SAM file for each compressing approach were compared with the uncompressed SAM file. The total, shared and unique proportional numbers of mapped reads are calculated with respect to the uncompressed SAM

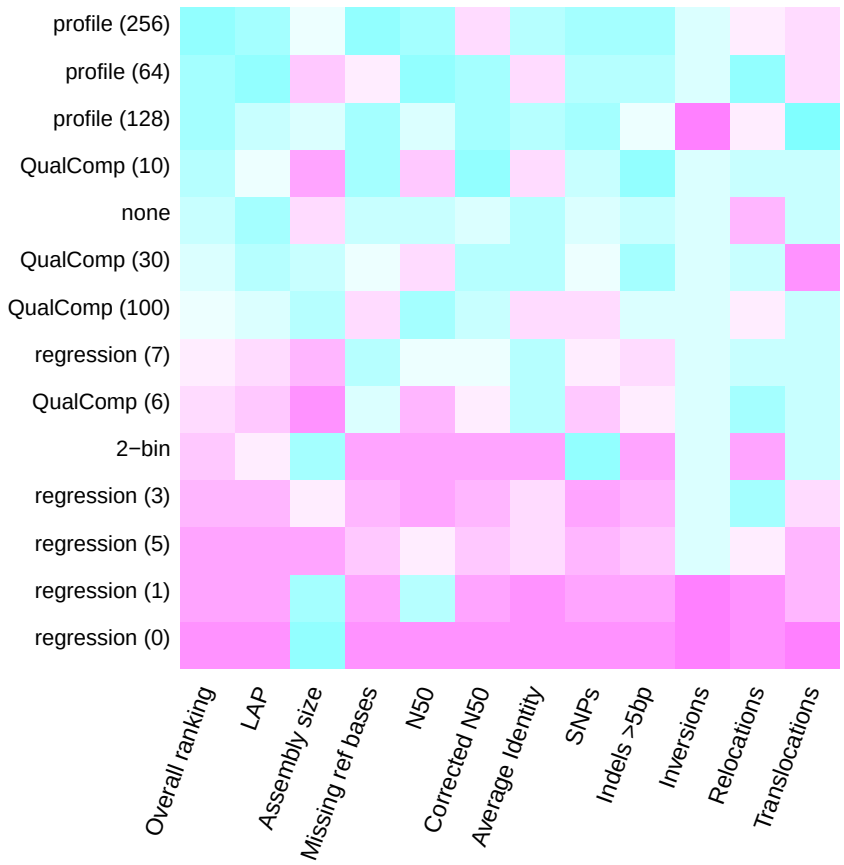


Figure 6.4: Rankings of compression methods based on *Rhodobacter sphaeroides* assembly attributes sorted by overall rank. Assemblies were constructed using ALLPATHS-LG. Rankings above the median value are in cyan, those below the median value in magenta.

matches as shown in table 6.1. Additionally, to monitor the effect of quality values on mapping in general, Bowtie2 was adjusted so that the maximum and minimum mismatch penalty were equivalent to maximum and minimum quality scores (with parameters: –mp 6,6 and –mp 2,2 respectively).

We evaluate the compression methods using two approaches. In the first approach, we order the compression methods based on how similar the alignment

results are using the original uncompressed quality values, i.e., the amount of reads aligned by both the uncompressed and compressed methods plus the amount of reads unaligned by both methods minus the amount of reads uniquely aligned by the uncompressed and compression method. In the second approach, we order the compression methods by total proportion of aligned reads.

The best compression method in terms of similarity with the uncompressed reads is QualComp with rate 100 bits/read, followed by QualComp with rate 30 bits/read, 256-, and 128-profile encoding, then 2-bin encoding, 64-profile encoding, QualComp with rate 10 bits/read, 7th-degree polynomial regression, QualComp with rate 6 bits/read, and finally, 5th-degree through 0-degree polynomial regression.

Ranking the compression methods by overall alignment rate produces an identical ordering as above. Aside from 0-degree polynomial regression (83.1%), all other compression methods have an alignment rate between 87% and 86.1%. The alignment rate of the uncompressed reads is 87%.

Most of the compression methods did not vary greatly in terms of the number of reads that were mapped *only* by the compression method; however, there is a sizable difference in the amount of reads that are originally mapped, but unmapped by the compressed methods. QualComp with rate 100 bits/read results in the fewest missing original read alignments (159). Increasing the regression model polynomial degree results in a decreasing amount of reads that are originally mapped, but unmapped by the regression model (40,931 to 1,134 reads for 0-degree and 7th-degree, respectively). There is no such trend for reads that are mapped only by the regression model.

Setting all bases as minimum quality results in the highest proportion of mapped reads 88.1%. Conversely, setting all bases as maximum quality results in the lowest proportion of mapped reads 72.8%.

		MaxQual		MinQual		2-bin		Degree 0		Degree 1	
		mapped	unmapped	mapped	unmapped	mapped	unmapped	mapped	unmapped	mapped	unmapped
original	mapped	746716	145897	892613	0	891864	749	851682	40931	883390	9223
	unmapped	0	132821	10821	122000	186	132635	67	132754	55	132766
	proportion	0.728	0.272	0.881	0.119	0.870	0.130	0.831	0.169	0.862	0.138

		Degree 3		Degree 5		Degree 7		Profile (64)		Profile (128)	
		mapped	unmapped	mapped	unmapped	mapped	unmapped	mapped	unmapped	mapped	unmapped
original	mapped	889537	3076	891019	1594	891479	1134	891753	860	891952	661
	unmapped	117	132704	155	132666	154	132667	144	132677	143	132678
	proportion	0.868	0.132	0.869	0.131	0.870	0.130	0.870	0.130	0.870	0.130

		Profile (256)		QualComp (6)		QualComp (10)		QualComp (30)		QualComp (100)	
		mapped	unmapped	mapped	unmapped	mapped	unmapped	mapped	unmapped	mapped	unmapped
original	mapped	892051	562	891375	1238	891777	836	892233	380	892454	159
	unmapped	119	132702	304	132517	265	132556	220	132601	172	132649
	proportion	0.870	0.130	0.870	0.130	0.870	0.130	0.870	0.130	0.870	0.130

Table 6.1: Mapping results of decompressed FASTQ files against *Rhodobacter sphaeroides* reference genome using Bowtie2. Numbers corresponds to the proportion of mapped reads with respect to the uncompressed FASTQ. "Shared" denotes the percentage of mapped reads by both the uncompressed and decompressed data. "Uncompressed only" denotes the percentage of reads mapped from the uncompressed data that are not mapped after decompression. "Compressed only" denotes the percentage of reads mapped from the decompressed data that were not mapped before compression.

6.1.5 Discussion

6.1.5.1 Lossy compression acceptable for subsequent biological analyses

The primary concern of using lossy compression methods is naturally the extent of information loss, that we quantified by MSE in this study. MSE and compressibility provide information in the theoretical context to the methods, but they are not the end-all of evaluation criteria. The performance of compressed datasets in different subsequent analyses and applications are just as important. Our benchmarks showed that some of the compression methods with high error rates are still practical for certain kinds of applications. Many subsequent tools proved to have enough additional redundancy built-in to handle such loss in information. Passing the decompressed quality values through quality control software shows that most methods filter nearly as many bases as using original quality sequences. Assemblers performing sequence alignment use percent similarity scores that are typically robust to standard sequencing errors.

6.1.5.2 Extension of 2-bin encoding

2-bin encoding has the nice property of being simple to compute and has good bits/base-call values. The 2-bin encoding suffers from having a high MSE, but fortunately, we have shown that in the case of genome assembly, 2-bin encoding outperforms all polynomial regressions encodings with degree less than 3.

2-bin encoding of the fragment and short-jump libraries of *Rhodobacter sphaeroides* have MSEs of 2.42× and 10.76× the 3rd-degree polynomial regression encodings, respectively. This further highlights the importance of using additional contextual information of the subsequent analyses when evaluating compressed quality values.

A potential extension to 2-bin encoding is to incorporate an additional bin (*okay*). The *okay* value can be used where the base qualities fall within a 2-bin range. Because the distribution of quality values is skewed towards higher quality, we need to experiment with different cutoffs for the *okay* value and determine if the additional storage is noticeable in subsequent analyses.

6.1.5.3 Extension of polynomial regression

The downside of modeling quality sequences using polynomial regression is that the model often requires a high number of degrees to achieve the same MSE as the profile and QualComp methods. However, storing a high number of coefficients requires more storage than losslessly compressing the original data. In order to increase the compressibility of modeling, we can attempt to store the *profiles* of certain polynomial functions. In other words, we can use a spline (a function that is piecewise-defined by polynomial functions) to represent a given quality sequence. Similar to our profile-encoding method, the user can specify how many polynomial functions they wish to store. Then a quality sequence can be divided evenly into a given number of segments, and each segment can be annotated with a polynomial function profile that closely matches its quality sequence.

6.1.5.4 Potential for operations on compressed data

Perhaps one of the greatest potential benefits of compressing quality values is the potential to perform quality control and possibly other operations directly on the compressed representations of the data. This is easiest to to consider for profile-based compression. The k profiles can be evaluated for (pre-)processing operations such as filtering and trimming, and the operations transitively applied to the entire set of reads, thus saving substantial computation associated with evaluating the full set of reads.

6.1.5.5 Future of lossy compression in bioinformatics analyses

We have simply provided here the initial steps in analyzing the effect of lossy compression on quality values using a single, high-coverage bacterial dataset. More work needs to be done using additional biological datasets, such as human and mouse, along with different sequencing technologies. A more direct comparison against related lossy compression tools, such as SlimGene [115] and qualcomp [116], needs to be performed. Additionally, other types of sequencing data can be analyzed apart from the Illumina data examined here. For example, the PacBio sequencing instruments produce very long reads (with average read lengths on the order of 10 kbp), but with the trade-off of having a high error-rate ($\sim$15%). Unlike the class of quality values we have examined here, the distribution of erroneous bases is relatively uniform [128]. The assembly complexity of bacterial genomes can be greatly simplified, producing near complete genome assemblies, by utilizing a single run of

these long reads [129]. If long read sequencing technologies such as PacBio become more widely adopted, it would be of huge benefit to examine the potential of lossy compression algorithms on not only the quality values, but the biological sequencing data themselves.

6.1.6 Conclusion

In this chapter we have examined lossy compression on sequence quality values and their effect on subsequent analyses. Although most previous examinations on lossy compression primarily focused on information loss, we have shown that typically used bioinformatics software today have additional built-in sensitivity to handle significant loss of information in the compressed quality values.

6.2 K-mulus: Strategies for BLAST in the Cloud

6.2.1 Abstract

With the increased availability of next-generation sequencing technologies, researchers are gathering more data than they are able to process and analyze. One of the most widely performed analysis is identifying regions of similarity between DNA or protein sequences using the Basic Local Alignment Search Tool, or BLAST. Due to the large amount of sequencing data produced, parallel implementations of BLAST are needed to process the data in a timely manner. While these implementations have been designed for those researchers with access to computing grids, recent web-based services, such as Amazon's Elastic Compute Cloud, now offer scalable, pay-as-you-go computing. In this paper, we present K-mulus, an application that performs distributed BLAST queries via Hadoop MapReduce using a collection of established parallelization strategies. In addition, we provide a method to speedup BLAST by clustering the sequence database to reduce the search space for a given query. Our results show that users must take into account the size of the BLAST database and memory of the underlying hardware to efficiently carry out the BLAST queries in parallel. Finally, we show that while our database clustering and indexing approach offers a significant theoretical speedup, in practice the distribution of protein sequences prevents this potential from being realized.

6.2.2 Introduction

Identifying regions of similarity between DNA or protein sequences is one of the most widely studied problems in bioinformatics. These similarities can be the result of functional, structural, or evolutionary relationships between the sequences. As a result, many tools have been developed with the intention of efficiently searching for these similarities [130–132]. The most widely used application is the Basic Local Alignment Search Tool, or BLAST [130].

With the increased availability of next-generation sequencing technologies, researchers are gathering more data than ever before. This large influx of data has become a major issue as researchers have a difficult time processing and analyzing it. For this reason, optimizing the performance of BLAST and developing new alignment tools has been a well researched topic over the past few years. Take the example of environmental sequencing projects, in which the biodiversity of various environments, including the human microbiome, is analyzed and characterized to generate on the order of several terabytes of data [48]. One common way in which biologists use these massive quantities of data is by running BLAST on large sets of unprocessed, repetitive reads to identify putative genes [133, 134]. Unfortunately, performing this task in a timely manner while dealing with terabytes of data far exceeds the capabilities of most existing BLAST implementations.

As a result of this trend, large sequencing projects require the utilization of high-performance and distributed systems. BLAST implementations have been created for popular distributed platforms such as Condor [135] and MPI [136, 137].

Recently, MapReduce [138] has become one of the de-facto standards for distributed processing. There are a few advantages of using the MapReduce framework over other existing parallel processing frameworks. The entirety of the framework rests in two simple methods: a *map* and a *reduce* function. The underlying framework takes care of the communication between nodes in the cluster. By abstracting the communication between nodes, it allows software developers to quickly design software that can run in parallel over potentially thousands of processors. Although this makes it simple to program, without direct control of the communication, it may be more inefficient compared to other distributed platforms.

While these parallel implementations of BLAST were designed to work on large computing grids, most researchers do not have access to these types of clusters, due to their high cost and maintenance requirements. Fortunately, cloud computing offers a solution to this problem, allowing researchers to run their jobs on demand without the need of owning or managing any large infrastructure. Web-based services, such as Amazons Elastic Compute Cloud (EC2) [139], have risen in recent years to address the need for scalable, pay-as-you-go computing. These services allow users to select from a collection of pre-configured disk images and services, and also allow more fine-grained customization down to the number of CPUs, speed, and amount of memory in their rented cluster.

In this paper, we present K-mulus, a collection of Hadoop MapReduce tools for performing distributed BLAST queries. We show that a limitation to previous cloud BLAST implementations is their "one size fits all" solution to parallelizing BLAST queries. We provide several different strategies for parallelizing BLAST

depending on the underlying cloud architecture and sequencing data, including: (1) parallelizing on the input queries, (2) parallelizing on the database, and then a (3) hybrid, query and database parallelization approach. Finally, we describe a k-mer indexing heuristic to achieve speedups by generating database clusters which results in a reduction of the search space during query execution.

6.2.3 Methods

6.2.3.1 MapReduce

The MapReduce framework was created by Google to support large-scale parallel execution of data intensive applications using commodity hardware [138]. Unlike other parallel programming framework where developers must explicitly handle inter-process communication, MapReduce developers only have to focus on two major functions, called *map* and *reduce*.

Prior to running a MapReduce program, the data must be first stored in the Hadoop Distributed File System (HDFS). The user then specifies a *map* function that will run on the chunks of the input data in parallel. MapReduce is "data aware," performing computation at the nodes containing the required data instead of transferring the data across the network. The *map* function processes the input in a particular way according to the developers specifications, and outputs a series of key-value pairs. Once all nodes have finished outputting their key-value pairs, all the values for a given key are aggregated into a list (via Hadoop's internal shuffle and sort mechanisms), and sent to the assigned reducer. During the reduce phase,

the (key, list of values) pairs are processed. This list of values is used to compute the final result according to the applications needs. For more details and examples, please see [138].

6.2.3.2 Parallelization strategies

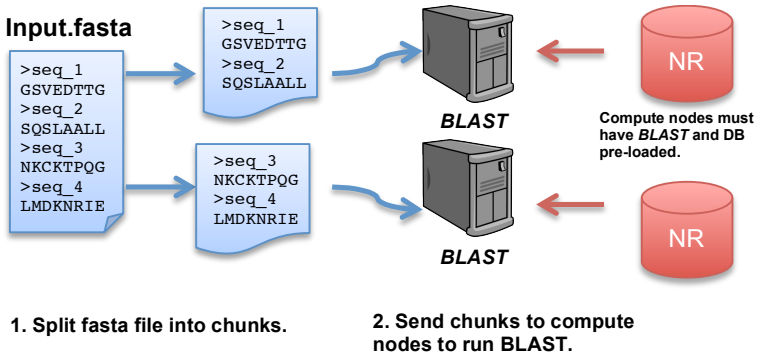


Figure 6.5: Query segmentation approach for parallelizing BLAST.

K-mulus uses three main strategies to perform distributed BLAST queries using Hadoop MapReduce. As we will show, the efficacy of these strategies are all dependent on the underlying hardware and data being used.

6.2.3.3 Query segmentation.

Arguably the simplest way to parallelize an application using MapReduce is to set the *map* function to the given application and execute it on subsets of the input. The individual results of the *map* functions are then aggregated by a single *reducer*. This query segmentation is the default implementation of CloudBLAST [140], a

popular MapReduce implementation of BLAST. Instead of writing custom *map* and *reduce* functions, CloudBLAST takes advantage of Hadoop's streaming extension that allows seamless, parallel execution of existing software on Hadoop without having to modify the underlying application.

The first step of the query segmentation approach is to partition the query file into a predetermined number of chunks (usually the number of computing nodes) and send them to random nodes (Fig. 6.5). This partitioning of the query sequences can be done automatically as the sequence files are uploaded to the HDFS. The user must pay special attention to the size of their query sequence file because the block sizes for the HDFS are 128MB by default. It is possible to underutilize the Hadoop cluster, since the *map* functions are often assigned to blocks of the input data. If a user uploads a 128MB sequence file to HDFS and uses Hadoop's streaming extension, then despite the number of nodes they request, BLAST will be performed only using the node containing the block of the data.

During runtime, the *map* function receives as input a block of FASTA-formatted sequences. Each *map* function simply executes the included BLAST binary against the included database sequences and outputs the results directly to disk. Although there is no need to use a reducing step for this strategy, one reducer can be used to aggregate the results.

6.2.3.4 Database segmentation.

Instead of segmenting the query, we can segment the database into a predetermined number of chunks. By segmenting the database, we can reduce the overhead of disk I/O for databases that do not fit completely into memory. Otherwise, as soon as the database grows larger than the amount of main memory, the runtime increases by orders of magnitude [136]. Therefore, it is important to examine the underlying hardware limitations and database size before using the default query segmentation approach.

During runtime, the query sequences are uploaded to the HDFS and sent to all nodes using the DistributedCache feature of Hadoop. The DistributedCache feature ensures that all nodes involved in the MapReduce have access to the same files. The *map* function is only responsible for passing the path of the database chunks to the reducer. Each *reduce* function executes BLAST on the complete set of input sequences.

Since BLAST takes into account the size of the database when computing alignment statistics, the individual BLAST results must have their scores adjusted for the database segmentation. Fortunately, BLAST provides the user an option to specify the effective length of the complete database.

6.2.3.5 Hybrid approach.

One potential problem with the database segmentation approach is that if we evenly partition the database across all nodes in our cluster, then the database

chunks may only fill up a small portion of the available memory. In this case, we must use a hybrid approach, where we segment the database into the least number of chunks that can fit entirely into memory. Afterwards, we replicate the database chunks across the remaining machines. During runtime, the query sequences are split and sent to the different the databases chunks, but only sent once to each of the database chunk replicates. This hybrid approach is also utilized by *mpiBLAST* [136], a widely-used distributed version of BLAST using MPI, which can yield super-linear speed-up over running BLAST on a single node.

During runtime, each *map* function receives a chunk of the query sequences and is responsible for sending out the chunk to each database partition. For each database partition i, the *map* function randomly selects a replicate to send the query chunk to in the form of a tuple ($db_{i,\text{replicate\_num}}$, query chunk). The reducer receives a collection of query chunks for a given database partition and replicate and BLASTs the query chunk against the database partition.

6.2.3.6 K-mer indexing

One of the original algorithms that BLAST uses is "seed and extend" alignment. This approach requires that there be at least one k-mer (sequence sub-string of length k) match between query and database sequence before running the expensive alignment algorithm between them [130]. Using this rule, BLAST can bypass any database sequence which does not share any common k-mers with the query. Using this heuristic, we can design a distributed version of BLAST using the MapReduce

model. One aspect of BLAST which we take advantage of is the database indexing of k-mers. While some versions of BLAST have adopted database k-mer indexing for DNA databases, it seems that this approach has not been feasibly scaled to protein databases [141]. For this reason, BLAST iterates through nearly every database sequence to find k-mer hits. Here we describe an approach for K-mulus that attempts to optimize this process by using lightweight database indexing to allow query iteration to bypass certain partitions of the database.

In order to cluster the database, for each sequence, we first create a vector of bits in which the value at each position indicates the presence of a specific sequence k-mer. The index of each k-mer in the vector is trivial to compute. We then cluster these bit vectors using a collection of clustering algorithms: k-means [126], and k-medoid [142]. Our algorithms perform clustering with respect to the presence vectors of each input sequence. For each cluster, a center presence vector is computed as the union of all sequence presence vectors in the cluster. The distance between clusters is taken as the Hamming distance, or number of bitwise differences, between these cluster centers. This design choice creates a tighter correspondence between the clustering algorithm and the metrics for success of the results, which depend entirely on the cluster presence vectors as computed above. We also keep track of the centers for each cluster as they play the crucial role of identifying membership to a cluster.

After the database has been clustered, we compare the input query sequences to all centers. The key idea is that by comparing the input query sequence to the cluster centers, we can determine whether a potential match is present in a given

cluster. If this is the case, we run the BLAST algorithm on the query sequence and the database clusters that we determined as relevant for the query.

6.2.4 Results

6.2.4.1 Comparison of parallelization approaches on a modest size cluster

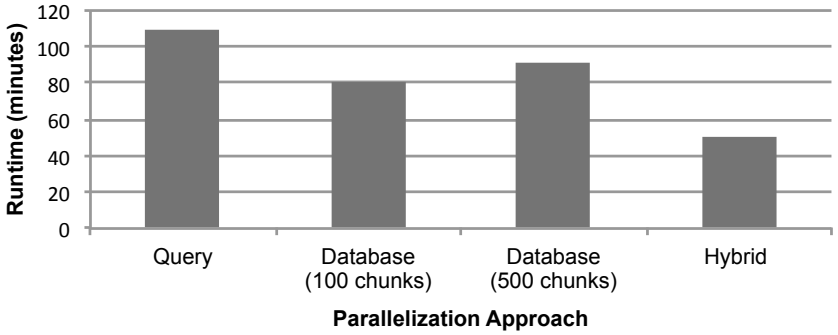


Figure 6.6: Runtimes of different BLAST parallelization approaches.

We evaluated the different parallelization approaches of protein BLAST on 30,000 translated protein sequences randomly chosen from the Human Microbiome Project [48] (Fig. 6.6). The sequences were BLAST against NCBI's non-redundant (*nr*) protein database (containing 3,429,135 sequences). For our analyses we used a 46 node Hadoop (version 0.20.2) cluster. Each node had 2 map/reduce tasks and 2GB of memory, reproducing a typical cloud cluster.

The *nr* database used was 9GB in size and unable to completely fit into the memory of a single node in our cluster. We segmented the database into 100 and 500

chunks to test our database segmentation approach. With 100 database chunks, the database will be roughly split across each reduce task. We included a partitioning of 500 database chunks to show the effects of over-partitioning the database.

Segmenting the database into 100 and 500 partitions resulted in a 26% and 16% decrease in runtime compared to the query segmentation approach, respectively. Although using a smaller number of database partitions was faster, there are still advantages for using more database partitions. Assuming an even distribution of query workload, if a node fails near the end of its BLAST execution, then that task must be restarted and the overall runtime is essentially doubled. Over-partitioning the database allows for a failed task to restart and complete faster.

Our hybrid query and database segmentation approach resulted in a 44% decrease in runtime compared to only query segmentation. Considering that the memory of each node in our cluster was 2GB, and the *nr* database was 9GB, we partitioned the database into 5 chunks, each roughly 2GB in size. This allows the databases to fit completely into memory at each node.

6.2.4.2 Analysis of database k-mer index

Using our clustering and k-mer index approach, we show noticeable speedups on well clustered data. To demonstrate this we simulated an ideal dataset of 1,000 sequences, where the sequences were composed of one of two disjoint sets of 3-mers. The database sequences were clustered into two even-size clusters. The sample query was 10,000 sequences, also comprising one of two disjoint sets of 3-mers. Figure

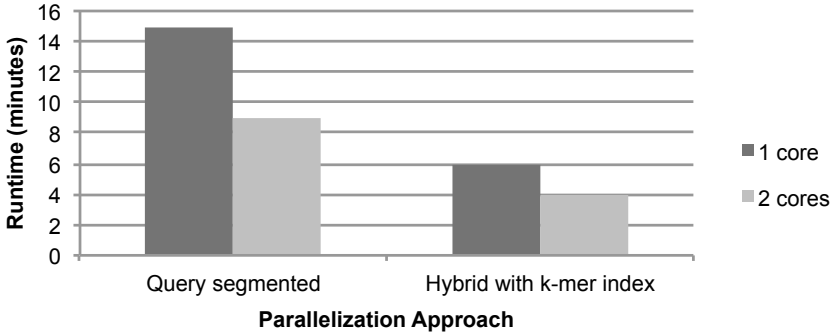


Figure 6.7: Runtimes of database segmentation with k-mer index approach.

6.7 shows the result of running BLAST on the query using Hadoop's streaming extension with query segmentation (the method used by CloudBLAST to execute BLAST queries) and K-mulus. K-mulus running on 2 cores with 2 databases yields a 56% decrease in runtime over BLAST using Hadoop's streaming extension on 2 cores. In practice, this degree of separability is nearly impossible to replicate, but this model allows us to set a practical upper bound for the speedup contributed by clustering and search space reduction.

For a more practical BLAST query using the *nr* database, our database and k-mer indexing approach took 2.75x as long compared to the naive Hadoop streaming method using a realistic query of 30,000 sequences from the HMP project. The poor performance is due to the very high k-mer overlap between clusters and uneven cluster sizes. Due to the high k-mer overlap, each query sequence is being replicated and compared against nearly all clusters.

K-mulus' database clustering and k-mer indexing approach shows poor per-

formance due entirely to noisy, overlapping clusters. In the worst case, K-mulus will map every query to every cluster and devolve to a naive parallelized BLAST on database segments, while also including some overhead due to database indexing. This is close to the behavior we observed when running our clustering and k-mer index experiments on the *nr* database. In order to describe the best possible clusters we could have generated from a database, we considered a lower limit on the exact k-mer overlap between single sequences in the *nr* database (Fig. 6.8). We generated this plot by taking 50 random samples of 3000 *nr* sequences each, computing the pairwise k-mer intersubsection between them, and plotting a histogram of the magnitude of pairwise k-mer overlap. This shows that there are very few sequences in the *nr* database which have no k-mer overlap which makes the generation of disjoint clusters impossible. Furthermore, this plot is optimistic in that it does not include BLASTs neighboring words, nor does it illustrate comparisons against cluster centers which will have intersubsection greater than or equal to that of a single sequence.

One strategy to improve the separability of the clusters and reduce the k-mer intersubsection between clusters is to use repeat masking software. In order to show the improvement offered by repeat masking, we ran SEG [143] on the sequences before computing the intersubsection (Fig. 6.8). On average, SEG resulted in a 6% reduction in the number of exact k-mer overlap between two given sequences. Repeat masking caused a significant, favorable shift in k-mer intersubsection and would clearly improve clustering results. However, the *nr* database had so much existing k-mer overlap that using SEG preprocessing would have almost no effect

on the speed of K-mulus' clustering and k-mer index approach.

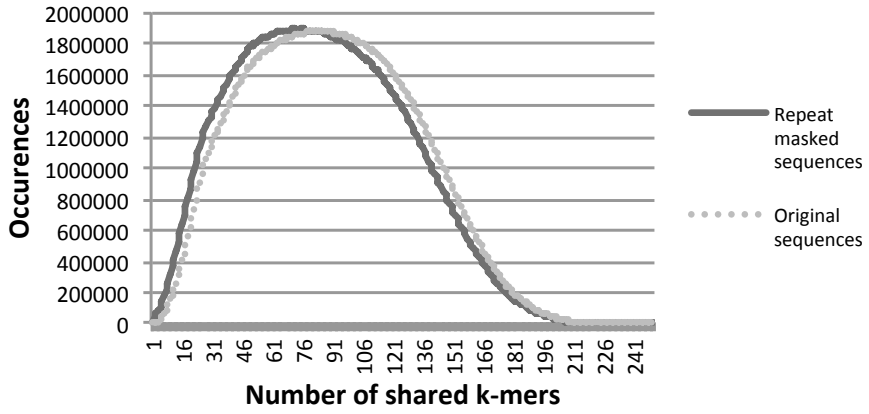


Figure 6.8: Pair-wise k-mer intersubsection of 50 random samples of 3000 original and repeat-masked *nr* sequences.

6.2.5 Discussion

With Amazon EC2 and other cloud platforms supporting Hadoop, developers should not make assumptions about the underlying hardware. Here we have provided K-mulus, which gives users the versatility to handle the common ways to perform distributed BLAST queries in the cloud without making assumptions of the underlying hardware and data. The default approach of most Hadoop implementations of BLAST is to segment the query sequences and run BLAST on the chunks in parallel. This approach works best when the entire BLAST database can fit into memory of a single machine, but as sequencing becomes cheaper and faster, this will become less likely. Computing clusters provided by services such as EC2 often contain commodity hardware with low memory, which we have shown makes

the default query segmentation approach poor in practice. The query segmentation approach works quite well on more powerful clusters that are able to load the entire database into memory. By providing users with the different parallel strategies, they are free to choose the one that is most effective with their data and hardware.

We have also provided a way to speed up BLAST queries by clustering and indexing the database using MapReduce. The speedup potential is largely dependent on the clusterability of the data. Protein sequences lie in high-dimensional non-Euclidean space, so by comparing them, we encounter the curse of dimensionality, where almost all pairs of sequences are equally far away from one another. This problem maybe slightly alleviated if we are trying to cluster multiple datasets of highly redundant sequences (multiple deep coverage whole genome sequencing projects with distinct, non-intersecting k-mer spectra). Future work includes clustering and indexing the query sequences, which may have higher redundancy than the database sequences.

Although our clustering and indexing approach was used on protein sequences, the logical next step is to include nucleotide database indexing, which has historically had more success in speeding up sequence alignment [132]. With a four character alphabet and simplified substitution rules, nucleotides are easier to work with than amino acids, and allow for much more efficient hashing by avoiding of the ambiguity inherent in amino acids.

It should be noted that the parallelization strategies presented here would also benefit other commonly used bioinformatics tools. Short read alignment tools (such as Bowtie2 [38]) can be parallelized by partitioning the reference index as well as the

query sequences. More work needs to be done to determine the best parallelization strategies for these tools running on commodity clusters.

Chapter 7: Conclusion

The genome is the blueprint for building an organism and helps researchers better understand the organism's function and evolution. Since its initial publication in 2001, researchers have periodically corrected mistakes in the human reference genome [1]. Determining what parts of the genome are missing or mistakes is a difficult task. The biological problem of genome assembly can be formulated as the computer science problem of reconstructing a text (*genome*) from a collection of randomly sampled word fragments with errors (*sequence reads*). The focus of this dissertation was to develop the theory and computational methods to compare and evaluate the reconstructed texts (*assemblies*).

I have developed computational tools that use the characteristics of the sequence data generation process to reproduce evaluations conducted by assembly experts without the use of reference genomes. I extended our likelihood-based framework and show that by taking into account abundances of assembled sequences, I can accurately compare different metagenomic assemblies. Lastly, I introduced VALET, the first *de novo* pipeline that flags regions in metagenomic assemblies that are statistically inconsistent with the data generation process. VALET has detected mis-assemblies in publicly available datasets and highlights shortcomings

in currently available metagenomic assemblers.

By providing the computational tools for researchers to accurately evaluate their assemblies, I decrease the chance of incorrect biological conclusions and misguided future studies.

Bibliography

[1] Monya Baker. De novo genome assembly: what every biologist should know. *Nat Meth*, 9(4):333–337, 04 2012.

[2] Philip Green. Against a whole-genome shotgun. *Genome Research*, 7(5):410–417, 1997.

[3] James L Weber and Eugene W Myers. Human whole-genome shotgun sequencing. *Genome Research*, 7(5):401–409, 1997.

[4] Phillip EC Compeau, Pavel A Pevzner, and Glenn Tesler. How to apply de Bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987–991, 2011.

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

[6] Paul Medvedev, Konstantinos Georgiou, Gene Myers, and Michael Brudno. Computability of models for sequence assembly. pages 289–301, 2007.

[7] Niranjan Nagarajan and Mihai Pop. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *Journal of computational biology*, 16(7):897–908, 2009.

[8] Carl Kingsford, Michael C Schatz, and Mihai Pop. Assembly complexity of prokaryotic genomes using short reads. *BMC bioinformatics*, 11(1):21, 2010.

[9] Jared T Simpson, Kim Wong, Shaun D Jackman, Jacqueline E Schein, Steven JM Jones, and Inanç Birol. ABySS: a parallel assembler for short read sequence data. *Genome research*, 19(6):1117–1123, 2009.

[10] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.

[11] Ruiqiang Li, Hongmei Zhu, Jue Ruan, Wubin Qian, Xiaodong Fang, Zhongbin Shi, Yingrui Li, Shengting Li, Gao Shan, Karsten Kristiansen, et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20(2):265–272, 2010.

[12] Steven L. Salzberg and James A. Yorke. Beware of mis-assembled genomes. *Bioinformatics*, 21(24):4320–4321, 2005.

[13] Niranjan Nagarajan, Christopher Cook, MariaPia Di Bonaventura, Hong Ge, Allen Richards, Kimberly A Bishop-Lilly, Robert DeSalle, Timothy D Read, and Mihai Pop. Finishing genomes with limited resources: lessons from an ensemble of microbial genomes. *BMC genomics*, 11(1):242, 2010.

[14] Claire M Fraser, Jonathan A Eisen, Karen E Nelson, Ian T Paulsen, and Steven L Salzberg. The value of complete microbial genome sequencing (you get what you pay for). *Journal of bacteriology*, 184(23):6403–6405, 2002.

[15] Elbert Branscomb and Paul Predki. On the high value of low standards. *Journal of bacteriology*, 184(23):6406–6409, 2002.

[16] GSC Consortia, HMP Jumpstart Consortia, PSG Chain, DV Grafham, RS Fulton, MG FitzGerald, J Hostetler, D Muzny, JC Detter, J Ali, et al. Genome project standards in a new era of sequencing. *Science (New York, NY)*, 326(5950), 2009.

[17] Jonathan Laserson, Vladimir Jojic, and Daphne Koller. Genovo: de novo assembly for metagenomes. *Journal of Computational Biology*, 18(3):429–443, 2011.

[18] Adam M Phillippy, Michael C Schatz, and Mihai Pop. Genome assembly forensics: finding the elusive mis-assembly. *Genome Biology*, 9(3):R55, 2008.

[19] Dent Earl, Keith Bradnam, John St John, Aaron Darling, Dawei Lin, Joseph Fass, Hung On Ken Yu, Vince Buffalo, Daniel R Zerbino, Mark Diekhans, et al. Assemblathon 1: a competitive assessment of de novo short read assembly methods. *Genome research*, 21(12):2224–2241, 2011.

[20] Steven L Salzberg, Adam M Phillippy, Aleksey Zimin, Daniela Puiu, Tanja Magoc, Sergey Koren, Todd J Treangen, Michael C Schatz, Arthur L Delcher, Michael Roberts, et al. GAGE: A critical evaluation of genome assemblies and assembly algorithms. *Genome research*, 22(3):557–567, 2012.

[21] Shiguo Zhou, Michael C Bechner, Michael Place, Chris P Churas, Louise Pape, Sally A Leong, Rod Runnheim, Dan K Forrest, Steve Goldstein, Miron Livny, et al. Validation of rice genome sequence by optical mapping. *BMC genomics*, 8(1):278, 2007.

[22] Catherine Adamidi, Yongbo Wang, Dominic Gruen, Guido Mastrobuoni, Xintian You, Dominic Tolle, Matthias Dodt, Sebastian D Mackowiak, Andreas Gogol-Doering, Pinar Oenal, et al. De novo assembly and validation of planaria transcriptome by massive parallel sequencing and shotgun proteomics. *Genome research*, 21(7):1193–1200, 2011.

[23] Giuseppe Narzisi and Bud Mishra. Comparing de novo genome assembly: the long and short of it. *PloS one*, 6(4):e19175, 2011.

[24] Francesco Vezzi, Giuseppe Narzisi, and Bud Mishra. Feature-by-feature–evaluating de novo sequence assembly. *PloS one*, 7(2):e31002, 2012.

[25] Makedonka Mitreva et al. Structure, function and diversity of the healthy human microbiome. *Nature*, 486:207–214, 2012.

[26] Barbara A Methé, Karen E Nelson, Mihai Pop, Heather H Creasy, Michelle G Giglio, Curtis Huttenhower, Dirk Gevers, Joseph F Petrosino, Sahar Abubucker, Jonathan H Badger, et al. A framework for human microbiome research. *Nature*, 486(7402):215–221, 2012.

[27] Todd J Treangen, Sergey Koren, Daniel D Sommer, Bo Liu, Irina Astrovskaya, Brian Ondov, Aaron E Darling, Adam M Phillippy, and Mihai Pop. Metamos: a modular and open source metagenomic assembly and analysis pipeline. *Genome biology*, 14(1):R2, 2013.

[28] Mohammadreza Ghodsi, Christopher M Hill, Irina Astrovskaya, Henry Lin, Dan D Sommer, Sergey Koren, and Mihai Pop. De novo likelihood-based measures for comparing genome assemblies. *BMC research notes*, 6(1):334, 2013.

[29] Martin Hunt, Taisei Kikuchi, Mandy Sanders, Chris Newbold, Matthew Berriman, and Thomas D Otto. Reapr: a universal tool for genome assembly evaluation. *Genome biology*, 14(5):R47, 2013.

[30] Ruibang Luo, Binghang Liu, Yinlong Xie, Zhenyu Li, Weihua Huang, Jianying Yuan, Guangzhu He, Yanxiang Chen, Qi Pan, Yunjie Liu, et al. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *Gigascience*, 1(1):18, 2012.

[31] Daniel D Sommer, Arthur L Delcher, Steven L Salzberg, and Mihai Pop. Minimus: a fast, lightweight genome assembler. *BMC bioinformatics*, 8(1):64, 2007.

[32] Eugene W Myers, Granger G Sutton, Art L Delcher, Ian M Dew, Dan P Fasulo, Michael J Flanigan, Saul A Kravitz, Clark M Mobarry, Knut HJ Reinert, Karin A Remington, et al. A whole-genome assembly of drosophila. *Science*, 287(5461):2196–2204, 2000.

[33] Osvaldo Zagordi, Arnab Bhattacharya, Nicholas Eriksson, and Niko Beerenwinkel. ShoRAH: estimating the genetic diversity of a mixed sample from next-generation sequencing data. *BMC bioinformatics*, 12(1):119, 2011.

[34] Irina Astrovskaya, Bassam Tork, Serghei Mangul, Kelly Westbrooks, Ion Măndoiu, Peter Balfe, and Alex Zelikovsky. Inferring viral quasispecies spectra from 454 pyrosequencing reads. *BMC bioinformatics*, 12(Suppl 6):S1, 2011.

[35] Paul Medvedev and Michael Brudno. Maximum likelihood genome assembly. *Journal of computational Biology*, 16(8):1101–1116, 2009.

[36] Scott Clark, Rob Egan, Peter I Frazier, and Zhong Wang. ALE: a generic assembly likelihood evaluation framework for assessing the accuracy of genome and metagenome assemblies. *Bioinformatics*, page bts723, 2013.

[37] Atif Rahman and Lior Pachter. CGAL: computing genome assembly likelihoods. *Genome Biology*, 14(1):R8, 2013.

[38] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357–359, 2012.

[39] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, et al. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[40] David A Rasko, Patricia L Worsham, Terry G Abshire, Scott T Stanley, Jason D Bannan, Mark R Wilson, Richard J Langham, R Scott Decker, Lingxia Jiang, Timothy D Read, et al. Bacillus anthracis comparative genome analysis in support of the amerithrax investigation. *Proceedings of the National Academy of Sciences*, 108(12):5027–5032, 2011.

[41] Faraz Hach, Fereydoun Hormozdiari, Can Alkan, Farhad Hormozdiari, Inanc Birol, Evan E Eichler, and S Cenk Sahinalp. mrsFAST: a cache-oblivious algorithm for short-read mapping. *Nature methods*, 7(8):576–577, 2010.

[42] Douglas B Rusch, Aaron L Halpern, Granger Sutton, Karla B Heidelberg, Shannon Williamson, Shibu Yooseph, Dongying Wu, Jonathan A Eisen, Jeff M Hoffman, Karin Remington, et al. The sorcerer ii global ocean sampling expedition: northwest atlantic through eastern tropical pacific. *PLoS biology*, 5(3):e77, 2007.

[43] Dongying Wu, Martin Wu, Aaron Halpern, Douglas B Rusch, Shibu Yooseph, Marvin Frazier, J Craig Venter, and Jonathan A Eisen. Stalking the fourth domain in metagenomic data: searching for, discovering, and interpreting novel, deep branches in marker gene phylogenetic trees. *PLoS One*, 6(3):e18011, 2011.

[44] Shibu Yooseph, Granger Sutton, Douglas B Rusch, Aaron L Halpern, Shannon J Williamson, Karin Remington, Jonathan A Eisen, Karla B Heidelberg, Gerard Manning, Weizhong Li, et al. The sorcerer ii global ocean sampling expedition: expanding the universe of protein families. *PLoS biology*, 5(3):e16, 2007.

[45] Thibault Varin, Connie Lovejoy, Anne D Jungblut, Warwick F Vincent, and Jacques Corbeil. Metagenomic analysis of stress genes in microbial mat communities from antarctica and the high arctic. *Applied and environmental microbiology*, 78(2):549–559, 2012.

[46] Shaomei He, Natalia Ivanova, Edward Kirton, Martin Allgaier, Claudia Bergin, Rudolf H Scheffrahn, Nikos C Kyrpides, Falk Warnecke, Susannah G Tringe, and Philip Hugenholtz. Comparative metagenomic and metatranscriptomic analysis of hindgut paunch microbiota in wood-and dung-feeding higher termites. *PloS one*, 8(4):e61126, 2013.

[47] Steven R Gill, Mihai Pop, Robert T DeBoy, Paul B Eckburg, Peter J Turnbaugh, Buck S Samuel, Jeffrey I Gordon, David A Relman, Claire M Fraser-Liggett, and Karen E Nelson. Metagenomic analysis of the human distal gut microbiome. *science*, 312(5778):1355–1359, 2006.

[48] Jane Peterson, Susan Garges, Maria Giovanni, Pamela McInnes, Lu Wang, Jeffery A Schloss, Vivien Bonazzi, Jean E McEwen, Kris A Wetterstrand, Carolyn Deal, et al. The nih human microbiome project. *Genome research*, 19(12):2317–2323, 2009.

[49] Yu Peng, Henry CM Leung, Siu-Ming Yiu, and Francis YL Chin. Meta-idba: a de novo assembler for metagenomic data. *Bioinformatics*, 27(13):i94–i101, 2011.

[50] Toshiaki Namiki, Tsuyoshi Hachiya, Hideaki Tanaka, and Yasubumi Sakakibara. MetaVelvet: an extension of velvet assembler to de novo metagenome assembly from short sequence reads. *Nucleic acids research*, 40(20):e155–e155, 2012.

[51] Sébastien Boisvert, Frédéric Raymond, Élénie Godzaridis, François Laviolette, Jacques Corbeil, et al. Ray meta: scalable de novo metagenome assembly and profiling. *Genome biology*, 13(12):R122, 2012.

[52] Keith R Bradnam, Joseph N Fass, Anton Alexandrov, Paul Baranay, Michael Bechner, İnanç Birol, Sébastien Boisvert10, Jarrod A Chapman, Guillaume Chapuis, Rayan Chikhi, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *arXiv preprint arXiv:1301.5406*, 2013.

[53] Florent E Angly, Ben Felts, Mya Breitbart, Peter Salamon, Robert A Edwards, Craig Carlson, Amy M Chan, Matthew Haynes, Scott Kelley, Hong Liu, et al. The marine viromes of four oceanic regions. *PLoS biology*, 4(11):e368, 2006.

[54] Elizabeth A Dinsdale, Robert A Edwards, Dana Hall, Florent Angly, Mya Breitbart, Jennifer M Brulc, Mike Furlan, Christelle Desnues, Matthew Haynes, Linlin Li, et al. Functional metagenomic profiling of nine biomes. *Nature*, 452(7187):629–632, 2008.

[55] Cora Carrigg, Olivia Rice, Siobhán Kavanagh, Gavin Collins, and Vincent OFlaherty. Dna extraction method affects microbial community profiles from soils and sediment. *Applied microbiology and biotechnology*, 77(4):955–964, 2007.

[56] M Krsek and EMH Wellington. Comparison of different methods for the isolation and purification of total community dna from soil. *Journal of Microbiological Methods*, 39(1):1–16, 1999.

[57] Jenna L Morgan, Aaron E Darling, and Jonathan A Eisen. Metagenomic sequencing of an in vitro-simulated microbial community. *PLoS One*, 5(4):e10209, 2010.

[58] Ben Temperton, Dawn Field, Anna Oliver, Bela Tiwari, Martin Mühling, Ian Joint, and Jack A Gilbert. Bias in assessments of marine microbial biodiversity in fosmid libraries as evaluated by pyrosequencing. *The ISME journal*, 3(7):792–796, 2009.

[59] Sergey Koren, Todd J Treangen, Christopher M Hill, Mihai Pop, and Adam M Phillippy. Automated ensemble assembly and validation of microbial genomes. *BMC bioinformatics*, 15(1):126, 2014.

[60] A.M. Phillippy, M.C. Schatz, and M. Pop. Genome assembly forensics: finding the elusive mis-assembly. *Genome Biology*, 9(3):R55, 2008.

[61] Arthur L Delcher, Steven L Salzberg, and Adam M Phillippy. Using mummer to identify similar regions in large sequence sets. *Current Protocols in Bioinformatics*, pages 10–3, 2003.

[62] Rayan Chikhi and Paul Medvedev. Informed and automated k-mer size selection for genome assembly. *arXiv preprint arXiv:1304.5665*, 2013.

[63] Martin D Davis and Elaine J Weyuker. Pseudo-oracles for non-testable programs. In *Proceedings of the ACM '81 conference*, ACM '81, pages 254–257, New York, NY, USA, 1981. ACM.

[64] Phil McMinn. Search-based failure discovery using testability transformations to generate pseudo-oracles. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 1689–1696, New York, NY, USA, 2009. ACM.

[65] Christian Murphy. *Metamorphic testing techniques to detect defects in applications without test oracles*. PhD thesis, New York, NY, USA, 2010. AAI3420778.

[66] Christian Murphy, Kuang Shen, and Gail Kaiser. Automatic system testing of programs without test oracles. In *Proceedings of the eighteenth international symposium on Software testing and analysis*, ISSTA '09, pages 189–200, New York, NY, USA, 2009. ACM.

[67] Shin Yoo. Metamorphic testing of stochastic optimisation. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, ICSTW '10, pages 192–201, Washington, DC, USA, 2010. IEEE Computer Society.

[68] Wing Kwong Chan, Jeffrey CF Ho, and TH Tse. Finding failures from passed test cases: improving the pattern classification approach to the testing of mesh simplification programs. *Softw. Test. Verif. Reliab.*, 20(2):89–120, June 2010.

[69] Tsong Yueh Chen, TH Tse, and Zhiquan Zhou. Semi-proving: an integrated method based on global symbolic evaluation and metamorphic testing. In *Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis*, ISSTA '02, pages 191–195, New York, NY, USA, 2002. ACM.

[70] Ljubomir Lazić and Nikos Mastorakis. Applying modeling and simulation to the software testing process: one test oracle solution. In *Proceedings of the 7th WSEAS international conference on Automatic control, modeling and simulation*, ACMOS'05, pages 248–256, Stevens Point, Wisconsin, USA, 2005. World Scientific and Engineering Academy and Society (WSEAS).

[71] René Just and Franz Schweiggert. Automating software tests with partial oracles in integrated environments. In *Proceedings of the 5th Workshop on Automation of Software Test*, AST '10, pages 91–94, New York, NY, USA, 2010. ACM.

[72] René Just and Franz Schweiggert. Automating unit and integration testing with partial oracles. *Software Quality Control*, 19(4):753–769, December 2011.

[73] Daniel Hook and Diane Kelly. Testing for trustworthiness in scientific software. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, SECSE '09, pages 59–64, Washington, DC, USA, 2009. IEEE Computer Society.

[74] Xiaoyuan Xie, Joshua W. K. Ho, Christian Murphy, Gail Kaiser, Baowen Xu, and Tsong Yueh Chen. Testing and validating machine learning classifiers by metamorphic testing. *J. Syst. Softw.*, 84(4):544–558, April 2011.

[75] Sergio Segura, Robert M. Hierons, David Benavides, and Antonio Ruiz-Cortés. Automated metamorphic testing on the analyses of feature models. *Inf. Softw. Technol.*, 53(3):245–258, March 2011.

[76] Dae S. Kim-Park, Claudio de la Riva, and Javier Tuya. An automated test oracle for xml processing programs. In *Proceedings of the First International Workshop on Software Test Output Validation*, STOV '10, pages 5–12, New York, NY, USA, 2010. ACM.

[77] Kambiz Frounchi, Lionel C. Briand, Leo Grady, Yvan Labiche, and Rajesh Subramanyan. Automating image segmentation verification and validation by learning test oracles. *Inf. Softw. Technol.*, 53(12):1337–1348, December 2011.

[78] Wing Kwong Chan, Shing-Chi Cheung, Jeffrey CF Ho, and TH Tse. Pat: A pattern classification approach to automatic reference oracles for the testing of mesh simplification programs. *Journal of Systems and Software*, 82(3):422–434, 2009.

[79] Tsong Chen, JoshuaWK Ho, Huai Liu, and Xiaoyuan Xie. An innovative approach for testing bioinformatics programs using metamorphic testing. *BMC Bioinformatics*, 10:1–12, 2009.

[80] Alessandro Orso and Tao Xie. Bert: Behavioral regression testing. In *Proceedings of the 2008 international workshop on dynamic analysis: held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008)*, WODA '08, pages 36–42, New York, NY, USA, 2008. ACM.

[81] Tsong Y Chen, Joshua WK Ho, Huai Liu, and Xiaoyuan Xie. An innovative approach for testing bioinformatics programs using metamorphic testing. *BMC bioinformatics*, 10(1):24, 2009.

[82] Javier Delgado, S Masoud Sadjadi, Marlon Bright, Malek Adjouadi, Hector Duran-Limon, et al. Performance prediction of weather forecasting software on multicore systems. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010.

[83] J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang. The weather research and forecast model: Software architecture and performance. In *Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, volume 25, page 29. World Scientific, 2004.

[84] Elaine J Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[85] Tsong Y Chen, Shing C Cheung, and SM Yiu. Metamorphic testing: a new approach for generating next test cases. *Department of Computer Science, Hong Kong University of Science and Technology, Tech. Rep. HKUST-CS98-01*, 1998.

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

[87] Todd J Treangen, Sergey Koren, Irina Astrovskaya, Dan Sommer, Bo Liu, and Mihai Pop. Metamos: a metagenomic assembly and analysis pipeline for amos. *Genome Biology*, 12(1):1–27, 2011.

[88] David R Kelley, Michael C Schatz, Steven L Salzberg, et al. Quake: quality-aware detection and correction of sequencing errors. *Genome Biol*, 11(11):R116, 2010.

[89] Michael Roberts, Wayne Hayes, Brian R Hunt, Stephen M Mount, and James A Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004.

[90] Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.

[91] Roger Barthelson, Adam J McFarlin, Steven D Rounsley, and Sarah Young. Plantagora: modeling whole genome sequencing and assembly of plant genomes. *PLoS One*, 6(12):1–9, 2011.

[92] Alexander V Lukashin and Mark Borodovsky. GeneMark.HMM: new solutions for gene finding. *Nucleic acids research*, 26(4):1107–1115, 1998.

[93] William H Majoros, Mihaela Pertea, and Steven L Salzberg. TigrScan and GlimmerHMM: two open source ab initio eukaryotic gene-finders. *Bioinformatics*, 20(16):2878–2879, 2004.

[94] Eric S Lander and Michael S Waterman. Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics*, 2(3):231–239, 1988.

[95] Rob Patro, Stephen M Mount, and Carl Kingsford. Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms. *Nature biotechnology*, 32(5):462–464, 2014.

[96] Robert McGill, John W Tukey, and Wayne A Larsen. Variations of box plots. *The American Statistician*, 32(1):12–16, 1978.

[97] Helga Thorvaldsdóttir, James T Robinson, and Jill P Mesirov. Integrative genomics viewer (IGV): high-performance genomics data visualization and exploration. *Briefings in bioinformatics*, page bbs017, 2012.

[98] Heng Li. wgsim-read simulator for next generation sequencing, 2013.

[99] Yu Peng, Henry CM Leung, Siu-Ming Yiu, and Francis YL Chin. IDBA-UD: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, 28(11):1420–1428, 2012.

[100] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012.

[101] Migun Shakya, Christopher Quince, James H Campbell, Zamin K Yang, Christopher W Schadt, and Mircea Podar. Comparative metagenomic and rrna microbial diversity characterization using archaeal and bacterial synthetic communities. *Environmental microbiology*, 15(6):1882–1899, 2013.

[102] Bahlul Haider, Tae-Hyuk Ahn, Brian Bushnell, Juanjuan Chai, Alex Copeland, and Chongle Pan. Omega: an overlap-graph de novo assembler for metagenomics. *Bioinformatics*, page btu395, 2014.

[103] Dinghua Li, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak-Wah Lam. MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph. *Bioinformatics*, page btv033, 2015.

[104] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

[105] Bo Liu and Mihai. Pop. Metacompass: comparative assembly of metagenomic sequences, 2015.

[106] Peter JA Cock, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res*, 38(6):1767–71, Apr 2010.

[107] Brent Ewing and Phil Green. Base-calling of automated sequencer traces using Phred. II. Error probabilities. *Genome Research*, 8(3):186–94, Mar 1998.

[108] Douglas W Bryant, Weng-Keen Wong, and Todd C Mockler. QSRA: a quality-value guided de novo short read assembler. *BMC Bioinformatics*, 10:69, 2009.

[109] S Gnerre, I MacCallum, D Przybylski, FJ Ribeiro, JN Burton, BJ Walker, T Sharpe, G Hall, TP Shea, S Sykes, AM Berlin, D Aird, M Costello, R Daza,

L Williams, R Nicol, A Gnirke, C Nusbaum, ES Lander, and DB Jaffe. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc Natl Acad Sci USA*, 108:1513–1518, 2011.

[110] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, and Mark A DePristo. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res*, 20(9):1297–303, Sep 2010.

[111] Himanshu Asnani, Dinesh Bharadia, Mainak Chowdhury, Idoia Ochoa, Itai Sharon, and Tsachy Weissman. Lossy compression of quality values via rate distortion theory. *arXiv preprint arXiv:1207.5184*, 2012.

[112] Rodrigo Cánovas, Alistair Moffat, and Andrew Turpin. Lossy compression of quality scores in genomic data. *Bioinformatics*, 30(15):2130–6, Aug 2014.

[113] Faraz Hach, Ibrahim Numanagić, Can Alkan, and S Cenk Sahinalp. SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics*, 28(23):3051–7, Dec 2012.

[114] Lilian Janin, Giovanna Rosone, and Anthony J Cox. Adaptive reference-free compression of sequence quality scores. *Bioinformatics*, page btt257, 2013.

[115] Christos Kozanitis, Chris Saunders, Semyon Kruglyak, Vineet Bafna, and George Varghese. Compressing genomic sequence fragments using SlimGene. *J Comput Biol*, 18(3):401–13, Mar 2011.

[116] Idoia Ochoa, Himanshu Asnani, Dinesh Bharadia, Mainak Chowdhury, Tsachy Weissman, and Golan Yona. QualComp: a new lossy compressor for quality scores based on rate distortion theory. *BMC Bioinformatics*, 14:187, 2013.

[117] Waibhav Tembe, James Lowey, and Edward Suh. G-SQZ: compact encoding of genomic sequence and quality data. *Bioinformatics*, 26(17):2192–4, Sep 2010.

[118] Raymond Wan, Vo Ngoc Anh, and Kiyoshi Asai. Transformations for the compression of FASTQ quality scores of next-generation sequencing data. *Bioinformatics*, 28(5):628–35, Mar 2012.

[119] Y William Yu, Deniz Yörükoglu, and Bonnie Berger. Traversing the k-mer landscape of NGS read datasets for quality score sparsification. In Roded Sharan, editor, *Research in Computational Molecular Biology - 18th Annual International Conference, RECOMB 2014, Pittsburgh, PA, USA, April 2-5, 2014, Proceedings*, volume 8394 of *Lecture Notes in Computer Science*, pages 385–399. Springer, 2014.

[120] Jiarui Zhou, Zhen Ji, Zexuan Zhu, and Shan He. Compression of next-generation sequencing quality scores using memetic algorithm. *BMC bioinformatics*, 15(Suppl 15):S10, 2014.

[121] André E Minoche, Juliane C Dohm, Heinz Himmelbauer, et al. Evaluation of genomic high-throughput sequencing data generated on Illumina HiSeq and Genome Analyzer systems. *Genome Biology*, 12(11):R112, 2011.

[122] DA Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40:1098–1101, 1952.

[123] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[124] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

[125] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[126] John A Hartigan and Manchek A Wong. Algorithm AS 136: A k-means clustering algorithm. *Applied statistics*, pages 100–108, 1979.

[127] Joshi NA and Fass JN. Sickle: A sliding-window, adaptive, quality-based trimming tool for FastQ files (version 1.33), 2013.

[128] Marco Ferrarini, Marco Moretto, Judson A Ward, Nada Šurbanovski, Vladimir Stevanović, Lara Giongo, Roberto Viola, Duccio Cavalieri, Riccardo Velasco, Alessandro Cestaro, and Daniel J Sargent. An evaluation of the PacBio RS platform for sequencing and de novo assembly of a chloroplast genome. *BMC Genomics*, 14:670, 2013.

[129] Sergey Koren, Gregory P Harhay, Timothy P L Smith, James L Bono, Dayna M Harhay, Scott D Mcvey, Diana Radune, Nicholas H Bergman, and Adam M Phillippy. Reducing assembly complexity of microbial genomes with single-molecule sequencing. *Genome Biol*, 14(9):R101, 2013.

[130] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[131] Sean R Eddy et al. A new generation of homology search tools based on probabilistic inference. In *Genome Inform*, volume 23, pages 205–211. World Scientific, 2009.

[132] W James Kent. BLAT-the blast-like alignment tool. *Genome research*, 12(4):656–664, 2002.

[133] Ying Li, Hong-Mei Luo, Chao Sun, Jing-Yuan Song, Yong-Zhen Sun, Qiong Wu, Ning Wang, Hui Yao, André Steinmetz, and Shi-Lin Chen. Est analysis reveals putative genes involved in glycyrrhizin biosynthesis. *BMC genomics*, 11(1):268, 2010.

[134] J Murray, J Larsen, TE Michaels, A Schaafsma, CE Vallejos, and KP Pauls. Identification of putative genes in bean (*Phaseolus vulgaris*) genomic (Bng) rflp clones and their conversion to STSs. *Genome*, 45(6):1013–1024, 2002.

[135] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[136] Aaron Darling, Lucas Carey, and Wu-chun Feng. The design, implementation, and evaluation of mpiblast. *Proceedings of ClusterWorld*, 2003, 2003.

[137] Jack J Dongarra, Rolf Hempel, Anthony JG Hey, and David W Walker. A proposal for a user-level, message passing interface in a distributed memory environment. Technical report, Oak Ridge National Lab., TN (United States), 1993.

[138] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[139] Amazon Inc. *Amazon Elastic Compute Cloud (Amazon EC2)*. Amazon Inc., http://aws.amazon.com/ec2, 2008.

[140] Andréa Matsunaga, Maurício Tsugawa, and José Fortes. Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications. In *IEEE Fourth International Conference on eScience, 2008.*, pages 222–229. IEEE, 2008.

[141] Aleksandr Morgulis, George Coulouris, Yan Raytselis, Thomas L Madden, Richa Agarwala, and Alejandro A Schäffer. Database indexing for production megablast searches. *Bioinformatics*, 24(16):1757–1764, 2008.

[142] Mark Van der Laan, Katherine Pollard, and Jennifer Bryan. A new partitioning around medoids algorithm. *Journal of Statistical Computation and Simulation*, 73(8):575–584, 2003.

[143] John C Wootton and Scott Federhen. Statistics of local complexity in amino acid sequences and sequence databases. *Computers & chemistry*, 17(2):149–163, 1993.