

Error correction and clustering algorithms for next generation sequencing

by

Xiao Yang

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Major: Bioinformatics and Computational Biology

Program of Study Committee:
Srinivas Aluru, Co-major Professor
Patrick Schnable, Co-major Professor
Steven Cannon
Karin Dorman
David Fernández-Baca

Iowa State University

Ames, Iowa

2011

Copyright © Xiao Yang, 2011. All rights reserved.

TABLE OF CONTENTS

LIST OF FIGURES	iv
LIST OF TABLES	vi
ABSTRACT	ix
CHAPTER 1. OVERVIEW	1
1.1 Next-generation Sequencing and Applications	4
1.2 Error Detection and Correction in Next-generation Sequencing Data	9
1.3 Environmental Sample Classification and Clustering	16
1.3.1 MapReduce Framework for Distributed Computing	20
CHAPTER 2. Error Correction for Low-Repetitive Genomes	23
2.1 Notations	23
2.2 Preliminaries	24
2.3 The Algorithm	27
2.4 Results	37
2.5 Discussions	45
2.6 Our Contributions	46
CHAPTER 3. Repeat-aware Sequencing Error Detection and Correction .	48
3.1 Preliminaries	49

3.2	Error Model	51
3.3	Error Detection and Correction	54
3.4	Results	55
3.4.1	Dataset Preparation	55
3.4.2	Error Detection and Correction Results	57
3.5	Discussion	63
3.6	Our Contributions	66
3.7	Detailed Threshold Inference	67
CHAPTER 4.	MapReduce Framework for Read Clustering in Metagenomics	71
4.1	Problem Formulation and Computational Modeling	74
4.2	Proposed Algorithmic Approach	78
4.3	Algorithms for Metagenomic Sequence Clustering	79
4.3.1	Phase I: Edge Construction and Validation	79
4.3.2	Phase II: Incremental Quasi-clique Enumeration	83
4.4	Map-Reduce Implementation	85
4.4.1	Edge Construction	86
4.4.2	Incremental Quasi-clique Enumeration	89
4.5	Experimental Results	91
4.5.1	Test Data and Experimental Environment	91
4.5.2	Experiments	92
4.6	Conclusions	98
CHAPTER 5.	Summary and Future Directions	100
BIBLIOGRAPHY		104

ACKNOWLEDGEMENTS	118
----------------------------	-----

LIST OF FIGURES

- 2.1 G is the target genome, shown as a bold line; r_i 's ($0 \leq i \leq 8$) represent reads, shown as thin lines; α_j ($0 \leq j \leq 8$), α'_2 and α''_2 are k mer instances in the reads, shown as rectangles. Every read is drawn aligned to its origin of sequencing position on the target genome. The bases at two positions in the k mers α_2 , α'_2 , and α''_2 are shown. All other positions in these k mers match across all three variants. 25
- 2.2 An illustration of choosing a tiling of a read for $d = 1$. A read is represented on top by a concatenation of rectangles, where each rectangle denotes a k mer. Each tile is represented by a concatenation of two adjacent arrows, which denote its k mer composition. For simplicity of illustration, we choose the read length to be divisible by k and each tile is a 0-concatenation of two adjacent k mers. X 's denote sequencing errors. Each bold arrow, a_i ($1 \leq i \leq 4$), denotes tile with insufficient evidence for correction. The placement of an alternative tile is indicated by a dotted arrow. 35

2.3	Gain and Sensitivity vs. Different Parameter Choices for $D3$. The first 11 sample points use parameters $k = 11, d = 1, t = 22$, and (C_m, Q_c) values $(14,60), (12,60), (10,60), (10,55), (8,60), (8, 55), (8, 50), (8, 45), (7, 45), (6, 45), (5, 45)$, respectively. The last sample point uses parameters $(k = 12, d = 2, t = 24, C_m = 8, Q_c = 45)$	43
3.1	A k mer neighborhood. The neighborhood of trimer AAA is the collection of trimers in R^3 that have a nonvanishing chance of being misread as AAA, in this case trimers with at most one substitution.	51
3.2	Plots of $\log(\text{FP} + \text{FN})$ vs. threshold for all datasets. In each plot, we compare the results by applying thresholds to \mathbf{Y} and to \mathbf{T} estimated by our model using the tIED, wIED, tUED and wUED error distributions.	60
3.3	Histogram of estimated T_l for <i>E. coli</i> dataset.	67

LIST OF TABLES

1.1	Properties of data produced by major Next-generation Sequencing platforms as of May, 2011.	5
1.2	A comparison between MPI and Hadoop	21
2.1	Experimental Datasets	38
2.2	Results of mapping each dataset to the corresponding genome using RMAP.	39
2.3	Results of Reptile and SHREC on Illumina Sequenced Short Reads. . .	41
2.4	Quality of ambiguous base correction using Reptile	45
3.1	Experimental datasets.	55
3.2	Estimated error probabilities $q_i(\cdot, \cdot)$, position $i = 11$	57
3.3	A comparison of minimum error rates. A Comparison of the minimum number of wrong predictions achieved by applying optimum thresholds to observed occurrences \mathbf{Y} , and our model with each of the error distributions tested. Bold numbers indicate that our model outperforms.	58
3.4	Error correction results.	62
4.1	Characteristics of the data used for experiments.	91

4.2	Quantity of the data generated in different stages when analyzing the three input datasets.	94
4.3	Run time in seconds for different stages of the CLOSET algorithm ex- ecuted on our 32 node Hadoop cluster.	95
4.4	Contingency Table for calculating ARI	97

ABSTRACT

Next generation sequencing (NGS) has revolutionized genomic data generation by enabling high-throughput parallel sequencing. This makes it possible to sequence new genomes or re-sequence individual genomes at a manifold cheaper cost and in an order of magnitude lesser time than traditional Sanger sequencing. Using NGS technologies, ambitious genomic sequencing projects target many organisms rather than a few, and large scale studies of sequence variation become feasible. Because of this revolution, the data analysis methodologies are changing, exemplified by different applications: de Bruijn or string graph based approach is replacing traditional overlap-layout-consensus paradigm in genome assembly, computational pipelines consisting of locating and counting short reads per gene location on the reference genome are replacing microarrays in gene expression analysis, and so on.

In this context, efficient analysis for large scale datasets is one of the most challenging problems. In this thesis work, we design efficient algorithms to improve the read quality for next generation sequencing and explore the emerging cloud computing techniques to cluster a large amount of metagenomic reads. First, we develop an efficient algorithm that uses a flexible read decomposition method to improve accuracy of error correction, and demonstrate its applicability using standard runs of Illumina sequencing. We further propose a statistical framework to differentiate infrequently observed subreads from sequencing errors when genomic repeats are prevalent. To differentiate between valid and invalid substrings based on their genomic

frequency, we propose a statistical approach to estimate a frequency related threshold based on the dataset under study. Lastly, we formalize the task to quantify microbial organisms in environmental samples as a sequence clustering problem and develop a parallel solution integrating sketching, quasi-clique enumeration and MapReduce techniques. The implementation is carried out using Hadoop – a MapReduce framework for cloud computing.

CHAPTER 1. OVERVIEW

We assume the reader has basic knowledge of molecular biology, such as familiarity with DNA (deoxyribonucleic acid) and RNA (ribonucleic acid) molecules, chromosomes, and genomes. In the following, we will only introduce some relevant concepts from computer science perspective, which we use to develop and describe our algorithms.

DNA is a double stranded molecule. Computationally, we treat each strand of a DNA molecule as a string over the alphabet $\Sigma = \{A, C, G, T\}$, where $c \in \Sigma$ is termed a nucleotide, or a base representing (A) Adenine, (C) Cytosine, (G) Guanine, and (T) Thymine. RNA, on the other hand, is typically single-stranded and can be represented as a string over the alphabet $\Sigma' = \{A, C, G, U\}$, where Thymine (T) in Σ is replaced by uracil (U). The length of a DNA or RNA sequence is measured in the number of *nucleotides* (nt), or *base pairs* (bp) when referring to the double stranded molecule. A DNA sequence can range from thousands to billions of nts in length, whereas, an RNA sequence is typically in the range of hundreds to thousands of nts (nt and bp are used interchangeably).

Sequencing refers to the process of determining the sequence of characters of one strand of a DNA sequence in genome sequencing or RNA sequence in transcriptome sequencing. Due to the limitations of sequencing techniques, each attempt to read a target sequence typically results in a short fragment, termed a *read*, where its length is in the range of tens to hundreds of base pairs. In a typical sequencing experiment, thousands to tens of millions of reading attempts are

made, which produce a large number of reads. Although we may be able to read every single position in the target sequence, deriving this complete sequence is a non-trivial task due to the loss of the following information during sequencing: the sampling location and the strand orientation of individual read with respect to the target sequence and the relative ordering of reads. Furthermore, adding to the complexity of this problem are a variety of errors that can be introduced in the reads during sequencing: insertions, deletions, and substitutions. For instance, let $S = \dots ACG\underline{TT}G \dots$ be the target DNA sequence, while reading the underlined nucleotide T , different types of errors can occur: AT may replace T as the result of an insertion error, T may not be read as the result of a deletion error, and G may replace T as the result of a substitution error. In addition, when the sequencer is uncertain about a base at a certain position, a universal character ‘ N ’ is used instead. Therefore, read errors lead to the enriched alphabet $\Sigma = \{A, C, G, T, N\}$ for DNA sequences.

Sequencing technique was initially developed in the 1970s by Frederick Sanger, and is known as Sanger sequencing [Sanger et al., 1977]. This technology consists of four components: dideoxy-based termination chemistry, fluorescent labeling, capillary separation, and automated computer reading [Men et al., 2008]. This has been the only widely used technology for three decades since the 1970s. Only within the past few years, a number of high-throughput sequencing technologies, known as next-generation sequencing technologies, have been developed and widely adopted [Perkel, 2009]. Understanding the properties of the reads produced by each of these technologies is important for using and analyzing such data in various applications. For instance, Sanger read based assembly methods [Batzoglou, 2002, Huang et al., 2003, Huson et al., 2002, Kalyanaraman et al., 2006] differ largely from short read assembly pipelines [Butler et al., 2008, Chaisson and Pevzner, 2008, Chaisson et al., 2009, Jackson et al.,

2010, Simpson et al., 2009, Simpson and Durbin, 2010, Zerbino and Birney, 2008] because the overlap-layout-consensus paradigm is more suitable for assembling long Sanger reads where genomic co-location can be reliably inferred for any pair of reads with significant alignments, whereas, among short reads, significant alignments arise by chance more frequently. In the following, we will ignore the details of the biological/chemical process behind each sequencing technique and focus only on the read properties, such as length and error rates, which have significant impact on algorithm design.

Typical length of Sanger reads ranges from 450bp to over 1kbp, with an average length around 800bp. Due to its labor intensiveness and high cost, Sanger sequencing can only be applied to a few carefully chosen model organisms. Notably, the human genome project took over 10 years (1990 to 2001) and cost over 1 billion dollars in order to generate a draft human genome. Although with the improved 96-capillary platform, up to 0.5Mbp DNA data could be produced per day [Men et al., 2008], which brought down the cost of sequencing human sized genome to 10 to 15 million dollars, the cost is still high. The introduction of pyrosequencing technique in 2004 unveiled the era of next-generation sequencing, featured by a much cheaper cost and a much higher throughput. Thanks to this revolution, the cost to sequence a human genome has dropped to under 10,000 dollars, a price of two magnitude cheaper; and this trend is expected to continue in the upcoming years.

With a decrease in the cost and increase in the throughput, the next-generation sequencing techniques also brought in new computational challenges, from the need to store and process large quantities of data, to handling much shorter read lengths and higher error rates compared to Sanger sequencing. In the following, we will first describe the properties of reads produced by popular next-generation sequencing platforms. Then, we demonstrate the need for detecting

and correcting errors in the data obtained from such platforms. We analyze the existing methods for error detection and correction and point out their shortcomings leading to an a compelling need for developing new algorithms to address these problems. Lastly, we address an interesting problem of read clustering in metagenomics and show how it has been tackled by the existing methods. Due to the need to avoid the $O(n^2)$ time complexity required for pairwise comparison of a large number of reads and the need for developing parallel solutions, we explore the sketching technique (widely used in web document clustering) and quasi-clique enumeration solutions to address this problem in cloud environment.

1.1 Next-generation Sequencing and Applications

High throughput next generation sequencing has revolutionized genomics, making it possible to sequence new genomes or re-sequence individual genomes at a manifold cheaper cost and in an order of magnitude less time than traditional Sanger sequencing [Shendure and Ji, 2008]. With this technology, ambitious genomic sequencing projects target many organisms rather than a few, and large scale studies of sequence variation become feasible [Stratton, 2008]. Many next-gen sequencing technologies have been developed, including systems currently in wide use, such as the Illumina Genome Analyzer (also known as Solexa previously), Applied Biosystems SOLiD, as well as more recent and new offerings from companies such as Complete Genomics and Pacific Biosciences [Parker, 2009, Korlach et al., 2010]. Note that sequencing machines such as those invented by Pacific Biosciences are considered as belonging to the third generation sequencing platform, featured by the use of single molecule sequencing (SMS) technique [Schadt et al., 2010].

We categorize sequencing data according to the company manufacturing the instrumenta-

Table 1.1 Properties of data produced by major Next-generation Sequencing platforms as of May, 2011.

Company	Equipment	Read length	Throughput per run	Time per run	Insert size	Technique
454 Life Sciences	Roche GS Titanium	400bp (avg)	400- 600Mb	10 hrs	3, 8, 20kb	sequencing by synthesis
Illumina	HiSeq 2000	35bp, 50bp, 100bp	150 - 200Gb	~ 8.5 day	200bp - 5kb	reversible terminator
Applied Biosystems	5500 SOLiD™ System	35bp, 60bp 75bp	up to 90Gb	1d 1 lane 1×35bp	600bp - 10kb	sequencing by ligation
Complete Genomics		35bp			500bp	ligation based sequencing
Helicos BioSciences	HeliScope™ SMS	25 - 55bp	21-35Gb	> 1Gb/hr		SMS
Pacific iosciences	PacBio RS	1k – 10k		< 30min expected	250bp - 6kb	SMS

tion that produce them since it is a convention to relate the data to the name of the sequencing platforms invented by a specific company. The properties of data produced by major platforms – 454 LifeSciences [Margulies et al., 2005], Illumina [Bentley et al., 2008], Applied Biosystems SOLiD Systems [McKernan et al., 2009], Helicos HeliScope [Shendure and Ji, 2008], Complete Genomics [Drmanac et al., 2010], and the emerging Pacific BioSciences – are described in Table 1.1, as per May, 2011. These properties are expected to change frequently.

During Sanger sequencing, reads are grown on the template strand by incorporating new bases with the help of a polymerase enzyme. Special types of bases, called terminators, are designed such that they are coded with fluorescent signals. Once a terminator has been incorporated, a read will stop growing. Terminators are introduced concurrently with normal nucleotides but with a lower concentration. Reads elongate and incorporate terminators at different lengths on different copies of the same type of template strand. By reading the last base from all of these reads (sorted by length), its sequence can be identified. This whole process is referred as the *chain termination sequencing*. In Illumina Genome Analyzer or 454 Roche GS

systems, different types of terminators are used such that they stop read elongation after being incorporated but can be washed away using special chemicals. Using this property, *sequencing by synthesis* (SBS) is carried out by repeating the following procedures: a terminator is incorporated, its signal is measured, and it is washed away. During this process, a terminator can be incorporated with the help of different enzymes. For instance, polymerase in Illumina Genome Analyzer or ligase in SOLiD™ 4 system. Due to the reason that sequencing by synthesis can be carried out in parallel on different types of templates, it achieves ultra high-throughput. A third type of sequencing technique is based on single molecule sequencing, which has been used by Helicos BioSciences and Pacific Biosciences. However, the chemical/physical process in SMS may differ largely. But these technologies are expected to have a much higher throughput compared to SBS based methods.

New technology inevitably comes with challenges. For many next generation sequencers, the advantage of deeper and cheaper coverage comes at the cost of shorter reads with higher error rates compared to the Sanger sequencing they replace. Although some technologies such as the 454 produce reads with an average length of 400bp, as shown in Table 1.1, most of these high throughput next generation sequencing systems produce short reads, ranging from 25bp to 150bp. Short read technologies have been widely used to initiate new applications and sometimes, to replace the existing ones. Some of these applications include genome sequencing, re-sequencing, and metagenomics. We will briefly describe these applications and challenges below.

The algorithms for genome assembly, the *de novo* inference of a genomic sequence, are different for Sanger reads as compared to short reads. Sanger reads are long enough that genomic co-location of a pair of reads can be reliably inferred if they share significant overlap.

This is the basic strategy used in the overlap-layout-consensus assembly paradigm [Batzoglou, 2002, Havlak et al., 2004, Huang et al., 2003, Myers et al., 2000]. However, with short reads from next generation sequencing, de Bruijn graph [Idury and Waterman, 1995] and string graph [Myers, 2005] based formulations that reconstruct the genome as a path in a graph perform better due to their more global analysis and ability to naturally accommodate paired read information. As a result, they have become de facto models for building short read genome assemblers, *e.g.*, ALLPATHS [Butler et al., 2008], Euler-SR [Chaisson and Pevzner, 2008], Velvet [Zerbino and Birney, 2008], ABySS [Simpson et al., 2009], and Yaga [Jackson et al., 2010]. In these algorithms, the graph size is bounded by the number of total k mers in the reference genome in theory. However, it becomes the limiting factor for scaling to large genomes due to the large quantity of reads produced and a relatively high error rate, resulting in an overwhelming number of spurious k mers that do not belong to the target genome. In addition, these artifacts lead to a higher chance of mis-assemblies. Therefore, detecting or correcting errors in the data pre-assembly becomes indispensable.

Re-sequencing refers to the sequencing of part of an individual's genome when a representative genome for the species is known. Next generation sequencing enabled the study of sequence variations in a high-throughput manner [Stratton, 2008]. Sequence variation means anything from standard population-level genetic polymorphisms [Tassell et al., 2008] to epigenomic changes among cells of the same organism [Jacinto et al., 2008], to DNA binding site variation in the same genome [Johnson et al., 2007]. Mapping short reads to the reference genome is a critical component in re-sequencing. The quality of mapping deteriorates severely in the presence of sequencing errors and genomic repeats. Reads from repetitive regions will map to multiple locations on the reference genome, but reads with sequencing errors may also

map to multiple locations, or sometimes nowhere at all. It becomes difficult to distinguish repetition from errors among such reads. Numerous mapping methods have been developed [Au et al., 2010, Clement et al., 2010, Frith et al., 2010, Griffith et al., 2010, Langmead et al., 2009, Li et al., 2009, Lin et al., 2008, McKenna et al., 2010, Rumble et al., 2009, Smith et al., 2008]. Most of them allow up to two mismatches per read at full sensitivity and can handle unique read mapping efficiently. However, uniquely assigning reads that map to multiple locations of the reference genome is still an actively pursued problem.

Metagenomics refers to the study of microbial organisms from an environmental sample, *e.g.*, sea water, soil, human gut. The emergence of next-generation sequencing technologies garnered tremendous interest in exploring the unknowns – deciphering genetic code, drafting full length genomes, studying symbiosis relationship between the microbial communities and their host [Adams et al., 2009, Blow, 2008, Chan et al., 2008, Cole et al., 2009, Diaz et al., 2009, Huang et al., 2010, McKenna et al., 2010]. A major focus of the exiting methods in analyzing metagenomic data is to classify microbial organisms in a sample according to what has previously been documented in a database. However, as sequencing becomes a routine task, studies of environmental samples shift to explore mostly undocumented species. Therefore, *de novo* clustering or binning reads becomes a more important task.

Many other applications in next-generation sequencing [Ansorge, 2009, Janitz, 2008, Marguerat et al., 2008] that have not been described here include whole transcriptome analysis, genome methylation analysis, Chromatin Immunoprecipitation for transcription factor binding sites detection, MicroRNA discovery and so on.

1.2 Error Detection and Correction in Next-generation Sequencing Data

Error detection and correction has long been recognized as a critical and difficult part of graph based short read assemblers and a necessary preprocessing step in many other applications such as re-sequencing. It has been implemented either as a component in genome assemblers [Butler et al., 2008, Chaisson et al., 2004, Jackson et al., 2010, Simpson et al., 2009, Zerbino and Birney, 2008] or as a standalone program [Gajer et al., 2004, Kelley et al., 2010, Qu et al., 2009, Schröder et al., 2009, Shi et al., 2009, Tammi et al., 2003, Wijaya et al., 2009] to improve read quality before the data will be used for other applications.

Sanger read assemblers rely on high quality overlaps between reads, which is readily achievable thanks to their long average read length (~ 800 bp). Although these reads contain errors, trimming the 3' end of each read effectively eliminates most errors without compromising overlap quality. However, to produce a high quality finishing sequence, it is necessary to refine the consensus sequence by re-aligning the reads to the consensus, and sometimes, re-checking the chromatogram (the raw signal to decide a sequencing base). MisEd [Tammi et al., 2003] and AutoEditor [Gajer et al., 2004] were designed for such a purpose. Common to both methods is the identification of sequencing errors via a majority voting rule: a less frequently observed base in an alignment column is considered an error. However, MisEd carried out multiple sequence alignment on its own whereas AutoEditor relies on alignment outputs of genome assemblers. AutoEditor further confirms the sequencing errors by comparing the consensus bases in the alignment to the chromatogram signals that were used for base calling. However, when the reference genomic sequences are not available or when the reads become too short, above approaches will no longer be applicable because carrying out multiple sequence alignments for a large number of reads would be computationally infeasible.

To address these difficulties, Spectrum Alignment Problem (SAP) formulation was proposed [Pevzner and Tang, 2001], where an exact solution was given in [Chaisson et al., 2004].

SAP-formulation: in a given dataset, a k mer is considered to be *solid* if it occurs over M number of times, and *weak* otherwise, where k and M are input parameters. Reads containing insolid k mers are converted using a minimum number of edit operations so that they contain only solid k mers post-correction.

Dynamic programming solution [Chaisson et al., 2004]: given M , let T be the set of all solid k mers in R . Define S to be a T -string if $S^k \subseteq T$. For every read $r \in R$, define the $score(i, t)$ to be the minimum edit distance between the prefix $r[0 : i - 1]$ of r and a T -string S with suffix t , where $t \in T$. Then, the score function can be recursively calculated as the minimum values of $score(i - 1, S[|S| - k - 1 : |S| - 2]) + hd(r[i - 1], t[k - 1])$ capturing substitution error, $score(i - 1, t) + 1$ capturing insertion error, and $score(i, S[|S| - k - 1 : |S| - 2]) + 1$ capturing deletion error. The minimum of $score(|r|, t)$ for all $t \in T$ corresponds to the minimum cost to convert r to a read that only contains solid k mers. The calculation of the recursive scoring function was carried out using a graph structure, where the vertex in the graph denotes a prefix length i of r and a k mer $t \in T$, and an edge exists between two vertices if they are related via one of the recursive functions.

The implementation of this dynamic programming solution will take $O(L|4^k|)$ space, hence, not conducive to large datasets due to memory limitations. Hence, heuristics were used to start with a T -string with a prescribed minimum length in r and to constrain the total number of edit distance allowed for converting r . After observing that errors in short reads such as Illumina reads are dominantly caused by substitutions, SAP formulation was adapted to consider only Hamming distance [Chaisson et al., 2009] and heuristics were applied in the following manner:

in each read, if a base change can increase the solid k mers to a prescribed amount, then it is applied. All reads that contain a prefix that is a T -string are considered fixable, otherwise, unfixable.

Similar approaches have been adapted and used by others [Butler et al., 2008]. However, this approach is still too time consuming to be applicable towards larger datasets. To expedite this process, Shi et al. [2009] implemented SAP-problem on GPUs.

Though SAP-formulation is computationally convenient, the error correction results need significant improvement. For this purpose, SHREC [Schröder et al., 2009] was proposed. SHREC constructs a generalized suffix trie (same as suffix tree except that edge labels are single characters) using both forward and reverse complementary strands of input reads. For each internal node u , the concatenation of edge labels from the root to u spells a substring s_u that occurs in the input, and the number of times s_u occurs equals the number of leaves of the subtree rooted at u . The expected occurrence of s_u can be computed analytically assuming the reference genome G to be a random string, G is uniformly sampled, and that sequencing errors uniformly occur at every read position. With these assumptions, the sampling of s_u can be considered as a collection of Bernoulli trials, where the mean $e = np$ with $p = \frac{l-|s_u|+1}{|G|}$, and the variance $\delta = np(1-p)$. Then, if the observed occurrence of s_u is less than $e - \alpha\delta$, s_u is considered as containing a sequencing error in the last base. The error correction results differ greatly with different α , which is a user specified parameter, but unfortunately, it is unclear how it should be chosen. In order to identify the potential correction for s_u , the subtree rooted at u is compared with the subtree rooted at v , where u and v are siblings and s_v passed the aforementioned frequency test. If the two subtrees are identical, then u can be merged to v and relevant changes in the suffix trie are made for all reads that were previously leaves under

u . Otherwise, there might be more than one error in a read containing s_u . Then, every read r under u is individually inspected and when applicable, $r[|s_u| - 1]$ is changed and all relevant suffixes of r are updated in the tree. For read with a high error rate, the above procedures could be applied for a fixed number of iterations to identify and correct multiple errors.

SHREC was further extended to capture insertion/deletion errors that may occur as frequently as substitution errors in 454 reads [Salmela, 2010]. Observe that if the last base of s_u is an insertion, then in the suffix trie, u should be compared with the siblings of the parent of u ; on the other hand, if a deletion occur, u should be compared with the children of its sibling. Same as SHREC, this extension is able to capture up to one insertion/deletion error in a read per given iteration. Validation on simulated and real datasets seemingly provided a convincing results that this strategy improves correction results. However, a closer look reveal some pitfalls. For real datasets, the accuracy of correction was measured by aligning the reads to the reference genome, where an increased percentage of reads that can be aligned is used as an indicator of improved correction results. This criterion is risky since an improved percentage of aligned reads may be due to excessive mis-corrections, which changes a read to part of the genome where it does not belong. As can be seen in the reported correction results, after correction, around 18% of the total genome was potentially lost in RC30x dataset (Table 6 [Salmela, 2010]) considering contigs with length $\geq 100bp$.

To improve memory usage, a suffix array based method named HiTEC [Ilie et al., 2011] was introduced. The basic ideas of HiTEC consist of the following: assume in read r , substring $r[i : i + k - 1]$ contains no errors whereas $r[i + k]$ is an erroneous base, where $0 \leq i \leq l - k - 1$. If we observe a substring s such that $|s| = k + 1$, $s[0 : k - 1] = r[i : i + k - 1]$, $s[k] \neq r[i + k]$, and s occurs over a preset value M times in R , then $s[k]$ is likely to be the real base attempted

for sequencing and $r[i+k]$ should be corrected to $s[k]$. In other words, any erroneous base can be corrected by HiTEC only if this base is preceded by an error free k mer. Then the goal is to derive k and M and using these values to achieve a satisfactory error correction result.

Aforementioned methods do not consider quality scores, which provide valuable guidance on sequence base reliabilities. Quake [Kelley et al., 2010] is a SAP-based approach that makes use of both the base quality score information and nucleotide specific miscall rates. The workflow consists of the following steps: 1) detect solid and insolid k mers – 1a) the overall weight of a k mer equals the weight summation of all of its instances in the dataset. The weight of every instance is calculated as the product of the probability that each base in this k mer is correct as defined by the corresponding quality score value; 1b) the histogram of the k mer weights is modeled as a mixture of Gaussian distribution and Zeta distribution for solid k mers and Gamma distribution for insolid k mers. A cutoff that differentiates the solid from insolid k mers is chosen (from the paper, it is unclear how the threshold is chosen). 2) convert reads containing insolid k mers to insolid k mer free reads – 2a) heuristically locate erroneous region in a read either by the union or the intersection of all insolid k mers. If some insolid k mers cover the 3' end, trimming is applied; 2b) greedily choose low quality score base for correction and pick the mutation with the highest likelihood until all k mers are solid. Abandon the read if no such choices exist. Although Quake has been applied to simulated datasets and achieved good results, experiments on real datasets are missing. Moreover, there exists a fundamental flaw in the data simulation process: every base in an Illumina read is simulated using the probability defined by the quality score. This strategy make it ideal for q -mer counting and cutoff selection strategies used in Quake.

Unlike these general purpose error correction methods, FreClu [Qu et al., 2009] targets

transcriptome data generated using Illumina small RNA protocol. FreClu groups input reads into a hierarchical tree such that 1) a parent node differs from any of its children by exactly one base, 2) children are less frequent than their parents, and 3) the transition probability from any parent to child, calculated using quality score information, is high enough to make error a likely explanation for child occurrences. The single base difference between any parent and child is considered a sequencing error in the child. For each tree, the root is considered as the real sequence and all other nodes are considered as the erroneous reads that originated from the root. Using this technique, up to 5% more reads can be mapped to the reference sequence. The tree structure accounts for frequency differences in RNA molecules and may be adaptable to sequencing of repetitive DNA. Indeed, we use related ideas later in our approach “Redeem”, but take a more global perspective, where multiple parents may give rise to the same erroneous sequence.

Not all sequencing errors can be accurately corrected among the ones that have been identified [Schröder et al., 2009]. However, the error correction results depends on correctly identifying of erroneous bases. By observing that SAP-based formulation requires a unique threshold M to determine if a k mer contains any errors, Chin et al. [2009] developed a statistical approach to derive the optimum M analytically so that the total number of errors left over in the datasets (including uncorrected errors and errors introduced by the algorithm) are minimized. Unfortunately, the formulation is based solely on the unrealistic assumptions of uniform genome sampling and uniform error distribution in the reads. It remains an open question if optimum M could be derived analytically without these assumptions, and whether multiple values of M should be used.

Although the aforementioned methods have addressed error detection or correction problem

from different aspects, the results are far from perfect. We list in the following several major issues and open questions to be resolved:

1. To handle a large amount of data. Next-generation sequencing techniques result in the production of as many as billions of reads in each experiment, and meaningfully analyzing these data is computationally demanding. For instance, SHREC [Schröder et al., 2009], one of the more efficient error correction methods, could barely handle a typical run generated by the more recent Illumina Genome Analyzer II platform (refer to section 2.4). Efficient algorithms that require moderate memory usage and tolerable run-time are needed.
2. To identify errors in reads that corresponding to repetitive regions of the reference sequence. A simplified version of this problem is to distinguish sequencing errors from biological polymorphisms.
3. To define a set of proper validation measures. Existing methods tend to validate error correction results by showing an improvement of assembly or SNP detection results post-correction over pre-correction [Kelley et al., 2010, Schröder et al., 2009]. Although a good error correction result can lead to an improvement over assembly or SNP detection, the vice versa, it is not necessarily true.
4. To detect and correct insertion or deletion errors.

In addition, developing quantitative methods and models to estimate errors for the various sequencing platforms is another challenging problem, which can improve the accuracy in identifying and correcting sequencing errors for different read types.

To address these problems, we proposed an error correction method, Reptile (Chapter 2), for genomes with low content of repeats and a repeat-aware error detection and correction

method, Redeem (Chapter 3).

1.3 Environmental Sample Classification and Clustering

Metagenomics is the study of genetic material sampled directly from environments. Typically, organisms in these samples are not easily separated, cultivated, and sequenced individually. Studies of metagenomics can lead to the discovery of new species [Venter et al., 2004, Poinar et al., 2006], increase our understanding of micro-organism communities, and facilitate in resolving environmental problems such as pollution.

One important problem in metagenomics is to profile the organism diversity in an environmental sample, by discovering 16S ribosome RNA (rRNA) genes (around 1500bp) of different organisms. This involves comparing the sequenced pool of 16S rRNA fragments from the sample with known databases [Liu et al., 2008, DeSantis et al., 2006a,b, Diaz et al., 2009, Huang et al., 2010, Huson et al., 2007, McHardy et al., 2007, Meyer et al., 2008, Wang et al., 2007]. However, many identified 16S rRNA sequences do not belong to any cultured species, indicating numerous new organisms in these samples [Blow, 2008, Meyer et al., 2008, TM]. In earlier studies, sequences of these metagenomes are sampled using the traditional shotgun sequencing, such as the pilot project of Sargasso Sea sampling [Venter et al., 2004]. Cloning biases and high cost of Shotgun sequencing prevent the discovery of species that are hard to be cultured and species with low percentage of presence in the sample. This limitation has been partly overcome by next-generation sequencing techniques [Edwards et al., 2006]. Among these techniques, 454 pyrosequencing has been more widely adopted than other types such as Illumina sequencing, because 454 reads are relatively longer using which a better separation of reads associated with different species can be achieved [Liu et al., 2008]. In addition, single molecule

sequencing technologies (so called next-next or third generation sequencing) that may produce over a few thousand kb reads can play a more important role in microbiome projects [Blow, 2008].

Motivated by the above problem, many methods have been developed to classify (via comparison with known databases for taxonomy assignment) or cluster metagenomic data. NAST [DeSantis et al., 2006a] provides a taxonomy assignment for each of the 77,363 16S rRNA reads in the NCBI database by comparing them with 16S rRNA reference sequences in RDP database [Cole et al., 2009]. The work flow of NAST consists of pre- and post-processing. Pre-processing includes three steps: 1) remove all reads with length $< 300\text{bp}$, 2) BLAST each read against all references, and reject the alignment with length smaller than 300bp, and 3) exclude any read that equally well matches more than one RDP terminal tree branch. During post-processing, every read is aligned to the most similar reference sequence (pairwise alignment) or alignments using a 4182-character template. The newly formed pairwise alignment is adjusted to maintain the intactness of the reference sequence in its original alignment downloaded from RDP. If the adjustment results in any gap with length over 10bp, then the read is excluded from further consideration. In this way, the algorithm outputs a set of multiple sequence alignments. For each alignment, only positions between 68 to 3689 are considered and all the reads that have length over 600bp are clustered into one Operational Taxonomic Units (OTU), where their pairwise similarities are measured using BLAST. Finally, taxonomy assignment for each OTU is derived according to RDP annotations.

Cd-hit [Li and Godzik, 2006, Huang et al., 2010] is a tool developed for protein sequence clustering. The work flow consists of the following steps: 1) Sort input sequences in the order of decreasing length. 2) Pick the longest sequence to be the reference, then, use word-counting

similarity metric (based on statistical analysis) to identify all similar sequences to the reference. Together, these sequences form a cluster. 3) Remove all sequences in the cluster formed in step 2, and repeat all three steps until sequences in the input dataset are all considered. Cd-hit suffers from a major drawback that the clustering process is biased towards longer sequences. This is because at a given similarity cutoff, a sequence in the input is more likely to be clustered with a longer sequence even though it has a higher similarity with a shorter sequence. Although the worst case performance of this method is still $O(N^2)$, where N is the total number of sequences in the input, in practice, the worst case performance is unlikely to happen, and in fact Cd-hit is one of the fastest clustering methods.

To partially avoid $O(N^2)$ comparisons, [Cameron et al. \[2007\]](#) adapted the fingerprinting technique widely used for webpage clustering [[Broder et al., 1997](#)]. Each read is first converted to a set of shingles (for instance, k mers) [[Broder et al., 1997](#)], then a subset of these shingles, termed sketches, are selected to represent this read. Reads are aligned only if they share a large percentage of common shingles. Direct application of shingling technique can cause problems of the same shingle being shared among too many reads due to the small alphabet size of DNA sequences, making it computationally overwhelming. Therefore, [Cameron et al. \[2007\]](#) converted each read to a set of non-overlapping k mers through a biased sampling technique, which however, may fail to identify all highly similar reads. The quality of the clustering could not be evaluated unless pairwise comparisons were carried out. In addition, this approach can only be used to cluster reads with very high similarities.

MEGAN [[Huson et al., 2007, 2009](#)] is a tool developed to use the BLAST program for classifying input reads to NCBI taxonomic tree structure. The input reads are initially BLAST-ed against the databases (*e.g.*, NCBI-NR) of NCBI, where each read is assigned to the lowest

common ancestor of the set of taxa it hit. [Huson et al. \[2007\]](#) demonstrated that short reads can be used for taxonomic assignment but at the cost of under-prediction – anywhere from 10% to 90% of reads may fail to produce valid hits to known databases using BLAST. Moreover, 454 reads with an average of 200bp may serve as a good tradeoff between under prediction and the production cost.

PhyloPythia, a SVM classifier trained using known genomic sequences, was proposed by [McHardy et al. \[2007\]](#). A flexible selection of genomic signatures such as *k*mer composition can be used for training the SVM, which avoids the dependence on gene sequences (via sequence assembly) for classification.

Different from PhyloPythia that trained using completely assembled genomes, [Chan et al. \[2008\]](#) used a seeded growing self-organized map (GSOM) trained using sequences flanking 16S rRNAs to classify metagenomics contigs. To avoid erroneous classification due to chimeric contigs, assembled sequences with minimum length $\geq 8kb$ were used. Firstly, the seed sequences (flanking regions of 16S rRNAs) of known genomes are clustered along with the unknown sequences using GSOM algorithm. Then, all seed sequences are identified. Finally, the neighboring sequences (defined by a clustering percentage parameter) of every seed node are assigned to the same class as the seed sequence.

Like PhyloPythia, TACO [\[Diaz et al., 2009\]](#) is a classification method trained using completely sequenced genomes. Every genomic sequence is converted to a vector storing the ratio of observed oligonucleotide to its expected frequency. The query vector is then compared to the reference using a defined distance function. The major difference between TACO and other existing machine learning approaches is the function used for sequence comparison. Consistent with all other composition-based methods, accurate classification can be made to long

sequences with at least a few thousand bases.

Several important issues are still remaining:

1. Clustering a large number of reads generated using next-generation sequencing techniques is still a computationally time consuming and memory demanding problem. None of the existing methods can be readily scaled to handle high-throughput sequencing datasets.
2. Environmental sample sequencing results in a large number of reads with unknown taxonomic assignment. In this scenario, read clustering rather than classification becomes a more important task, and this issue has not been adequately addressed.
3. Evaluation of clustering and taxonomic assignment are still not well defined.

To address these issues, we propose and develop a parallel read clustering framework in Chapter 4. Since we will be using MapReduce techniques implemented in Hadoop framework, we will first introduce it here.

1.3.1 MapReduce Framework for Distributed Computing

Hadoop is a programming framework that supports the processing of large data sets in a distributed computing environment. It was originally conceived based on Google's MapReduce [Dean and Ghemawat, 2008] concept. Recently, it gains popularity in communities beyond parallel computing, such as computational biology.

What makes Hadoop popular: Hadoop versus MPI.

MPI stands for message passing interface, which is an application programming interface that allows processes to communicate with each other. It is the main framework used for implementing parallel algorithms. The pros and cons of both frameworks are summarized in Table 1.2. Due to the reason that Hadoop is accessible to researchers with little expertise

Table 1.2 A comparison between MPI and Hadoop

Framework	Pros	Cons
Hadoop	<ul style="list-style-type: none"> - hidden parallelization from users, easy to program, good code portability - excellent fault-tolerance (out of memory/disk space, physical failure, network failure/congestion) - flat scalability 	<ul style="list-style-type: none"> - no guarantee of efficiency - constrained application - currently, difficult to tune system parameters
MPI	<ul style="list-style-type: none"> - fully controllable dataflow, highly flexible in programming - highly efficient for implementing well designed efficient parallel algorithms - can implement highly efficient MapReduce framework 	<ul style="list-style-type: none"> - expertise knowledge required to program, low portability of code - no automatic fault tolerance

in parallel computing and provide a highly fault tolerant system to process large datasets, it is favored over MPI in many biological applications. However, very limited algorithms can be efficiently implemented using Hadoop framework, making it more suitable for applications requiring low intensive communications.

Hadoop Distributed File System (HDFS).

The traditional Network File System (NFS) is designed for allowing remote access to files residing in other machines. These files are typically small and unique over the whole system. HDFS, on the other hand, is specialized for allowing remote access to very large files in a streaming fashion as compared to random access to a set of files and collect small amount of information. Every file in HDFS is divided into physical blocks, distributed among different nodes, termed *DataNode*. The metadata recording the block locations for each file is stored

in a *NameNode*, which typically reside in the main memory to allow fast access. To tolerate node failure, file blocks are duplicated in the system and there can be redundant NameNodes to preserve metadata information.

MapReduce.

The work flow of a MapReduce task consists of the following steps: the input file is divided into blocks in the HDFS. Each block is read individually by a *RecordReader* function, then processed by a *Mapper* function defined by users. Next, the output from all the Mappers are partitioned by a pre-defined function (typically a hash function) and sent to different nodes. Data on the same node is stored in the format of <key, value> pairs. The pairs sharing the same keys are processed collectively by a user defined reducer function and output to hard drive, which will be processed by the next MapReduce iteration.

CHAPTER 2. Error Correction for Low-Repetitive Genomes

In this Chapter, we present Reptile (Representative Tiling for Error Correction), a scalable short read error correction method. We draw upon the k -spectrum approach pioneered in earlier results, but explore multiple alternative k mer decompositions of an erroneous read and use contextual information specified by neighboring k mers to infer appropriate corrections. Reptile also incorporates quality score information when available. We present algorithmic strategies to store k mer Hamming distance graphs and efficiently retrieve all graph neighbors of a k mer as candidates for correction. We compare our results with SHREC [Schröder et al., 2009], a high quality short read error correction method, and one of the more recent in a line of continuously improving error correction protocols. In all experiments with Solexa datasets, Reptile outperforms SHREC in percentage of errors corrected and accuracy of true base assignment. Furthermore, a significant reduction in memory usage and run time makes Reptile more applicable to larger datasets. As with most current approaches including SHREC, Reptile is targeted to short reads with substitution errors, assuming insertion and deletion errors are rarely produced by short read sequencing technology [Dohm et al., 2008].

2.1 Notations

Let G denote the reference genome that has been sequenced, and let $R = \{r_1, r_2, \dots, r_n\}$ be a collection of resulting short reads. For simplicity (but without loss of generality), we

assume each read r_i has a fixed length L . The coverage of the genome by the reads is given by $Cov = \frac{nL}{|G|}$, where $|G|$ denotes the genome length. Define the k -spectrum of a read r to be the set $r^k = \{r[i : i + k - 1] \mid 0 \leq i < L - k + 1\}$, where $r[i : j]$ denotes the substring from position i to j in r . We index the positions of a string starting from 0. The k -spectrum of R is given by $R^k = \bigcup_{i=1}^n r_i^k$.

Let α and β be two strings such that $\alpha[(|\alpha| - l) : (|\alpha| - 1)] = \beta[0 : (l - 1)]$ for some $0 \leq l < \min(|\alpha|, |\beta|)$. We define the l -concatenation of α and β , denoted $\alpha \parallel_l \beta$, to be the string γ of length $|\alpha| + |\beta| - l$ such that $\gamma[0 : (|\alpha| - 1)] = \alpha$ and $\gamma[(|\gamma| - |\beta|) : (|\gamma| - 1)] = \beta$.

The Hamming distance between two strings α_1 and α_2 for $|\alpha_1| = |\alpha_2|$, denoted $hd(\alpha_1, \alpha_2)$, is the number of positions at which they differ. For a k mer $\alpha_i \in R^k$, define its d -neighborhood $N_i^d = \{\alpha_j \in R^k \mid hd(\alpha_i, \alpha_j) \leq d\}$. Its complete d -neighborhood $N_{ci}^d = \{\alpha_j \mid hd(\alpha_i, \alpha_j) \leq d\}$ contains all k mers within Hamming distance d , whether or not they occur in R^k .

2.2 Preliminaries

The success of any error correction method relies on an adequate coverage of the target genome. If we know the genomic location of every read, we could layout all reads that contain a specific genomic position into a multiple alignment (Fig. 2.1) and correct all erroneous bases to the consensus base under the reasonable assumption that errors are infrequent and independent. For instance, base T in r_3 would be considered a sequencing error to be corrected to the consensus base A .

The main idea underlying Reptile is to create approximate multiple alignments, with the possibility of substitutions, in the absence of location information. Multiple alignments with substitutions could be created by considering all reads with pairwise Hamming distance less

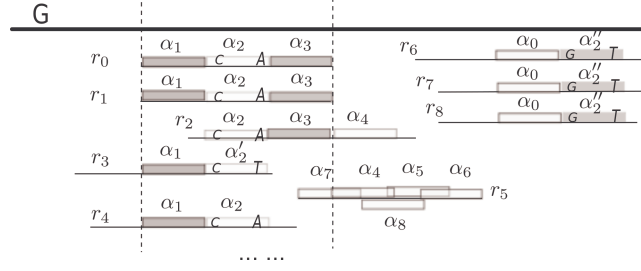


Figure 2.1 G is the target genome, shown as a bold line; r_i 's ($0 \leq i \leq 8$) represent reads, shown as thin lines; α_j ($0 \leq j \leq 8$), α'_2 and α''_2 are k mer instances in the reads, shown as rectangles. Every read is drawn aligned to its origin of sequencing position on the target genome. The bases at two positions in the k mers α_2 , α'_2 , and α''_2 are shown. All other positions in these k mers match across all three variants.

than some threshold, but such alignments are already hard [Manthey and Reischuk, 2005] and even in high coverage situations, the occurrence of many exactly coincident reads, e.g. r_0 and r_1 in Fig. 2.1, are rare. We therefore resort to alignments on subreads, the substrings of a read.

Storing R , let alone all its subreads, could be memory intensive, not to mention the memory required to store information required for error correction. Inspired by the idea for bounding memory usage with de Bruijn graphs in short read assembly, we work with k mer subreads of input data, where the memory of storing the k -spectrum R^k is bounded by $O(\min(4^k, n(L - k + 1)))$. Typically, k is chosen so that the expected number of occurrences of any k mer in the genome should be no more than one, i.e., $4^k > |G|$. Therefore, choosing $10 \leq k \leq 16$ is sufficient for microbial or human genomes, in which case the k -spectrum would fit within 4GB RAM regardless of input size.

Focusing on reasonably short k mers has several advantages. First, we expect an adequate number of k mers to align to the same position along the genome even with relatively low coverage (e.g. 40x). High local coverage is needed to identify erroneous bases. For instance, in Fig. 2.1, there exist five subreads, four copies of α_2 and one copy of α'_2 , aligning to the

same starting position in the genome, but this number reduces to three for the longer subread $\alpha_2 ||_0 \alpha_3$. Second, it is less compute-intensive to identify N_i^d when k is small, since there are fewer ways to select d out of k positions. Last, sequencing errors in k mers are much less frequent compared to full length reads, so d need not be large.

Nevertheless, relying solely on short k mers can easily lead to ambiguities when resolving erroneous bases. For instance, in Fig. 2.1, without knowing the alignment, it is unclear if α'_2 should be corrected to α_2 or α''_2 , since both $hd(\alpha'_2, \alpha_2) = hd(\alpha'_2, \alpha''_2) = 1$ and α_2 and α''_2 have a similar higher frequency. However, the contextual information of α'_2 available from read r_3 (in this case, α_1) uniquely identifies α_2 as the right correction. We seek a way to use contextual information to help resolve errors *without* increasing k and lowering local coverage.

Contextual information is provided by noting which k mers coexist in observed reads. For instance, $\alpha_1 ||_0 \alpha_2$ exists in r_0 , r_1 or r_4 , and $\alpha_5 ||_{.5k} \alpha_6$ exists in r_5 in Fig. 2.1. The following definitions formalize the notion of contextual information.

Definition 2.1 $t = \alpha_1 ||_l \alpha_2$ ($0 \leq l < k$) is a tile of read r if t is a substring of r , and $|\alpha_1| = |\alpha_2| = k$.

Definition 2.2 $t' = \alpha' ||_l \beta'$ is a d -mutant tile of $t = \alpha ||_l \beta$ if $hd(\alpha, \alpha') \leq d$ and $hd(\beta, \beta') \leq d$.

Definition 2.3 $T_r = (t_1, t_2, \dots, t_m)$ is a tiling of read r if $r = t_1 ||_{l_1} t_2 \dots ||_{l_{m-1}} t_m$ such that 1) t_i ($1 \leq i \leq m$) is a tile of r , and 2) $l_i \geq 1$ ($1 \leq i < m$).

Note that if t_i is specified as $r[j : j']$, then the overlaps between consecutive tiles can be inferred; i.e., l_i 's can be derived from t_i 's. Multiple tilings exist for any read. For example, both $((\alpha_7 ||_{.5k} \alpha_4), (\alpha_5 ||_{.5k} \alpha_6))$ and $((\alpha_7 ||_0 \alpha_8), (\alpha_8 ||_0 \alpha_6))$ are tilings of r_5 . We also extend the concept of d -mutant tiles to tilings. For instance, we can think of $((\alpha_1 ||_0 \alpha'_2), (\alpha'_2 ||_0 \alpha_3))$

as a 1-mutant tiling of $T_{r_0} = ((\alpha_1 ||_0 \alpha_2), (\alpha_2 ||_0 \alpha_3))$. Similarly, $((\alpha_1 ||_0 \alpha_2'')(\alpha_2'' ||_0 \alpha_3))$ is a 2-mutant tiling of T_{r_0} . Formally,

Definition 2.4 *A d -mutant tiling of $T_r = (t_1, t_2, \dots, t_m)$ of read r is a sequence of tiles $(t'_1, t'_2, \dots, t'_m)$ such that 1) $|t_i| = |t'_i|$ and $|t_i ||_{l_i} t_{i+1}| = |t'_i ||_{l_i} t'_{i+1}|$, where l_i is implicitly determined by r for $1 \leq i < m$, and 2) t'_i is a d -mutant tile of t_i for $1 \leq i \leq m$.*

If read r contains errors and T_r is a tiling of r , then we expect to find a tiling T_s of the true read s as one of the d -mutant tilings of T_r , where constituent tiles of T_s have higher coverage than those of T_r . However, in some cases, T_s will not be found among the d -mutant tilings.

2.3 The Algorithm

For ease of presentation, we assume R can fit in main memory (this requirement will be relaxed later). The algorithm consists of two major phases. We first provide a brief overview, and subsequently describe the algorithm in more detail and analyze its time and space complexity.

1. Information extraction.

- (a) Derive the k -spectrum R^k of R .
- (b) Derive Hamming graph $G_H = (V_H, E_H)$, where $v_i \in V_H$ represents $\alpha_i \in R^k$ and $\exists e_{ij} = (v_i, v_j) \in E_H \iff hd(v_i, v_j) \leq d$, for a given threshold d .
- (c) Compute tile occurrences.

2. Individual read error correction.

- (a) Place an initial tile t at the beginning of the read.

- (b) Identify d -mutant tiles of t .
- (c) Correct errors in t as applicable.
- (d) Adjust tile t placement and go to step 2b, until tile placement choices are exhausted.

Given a read $r \in R$, any of its constituent k mers α is a vertex v in the Hamming graph. The d -neighborhood of α is accessible via the edges incident to v . Hence, if α contains at most d substitution sequencing errors, the k mer it should be corrected to exists in its d -neighborhood. By building local, approximate alignments of tilings constructed from d -neighborhoods, our strategy identifies a tiling of the true read as a high frequency tiling.

Phase 1: Information Extraction

Constructing R^k involves one linear scan of each read in R . This takes $O(nL)$ time. We maintain R^k in sorted order using $O(|R^k| \log |R^k|)$ time. The space requirement for R^k is given by $|R^k| = O(\min(4^k, n(L-k+1)))$. Any non-*ACGT* characters (due to difficulty in base calling) are initially converted to *A*, which will be validated or corrected later by the algorithm.

During error correction, it is important to have fast access to the d -neighborhood of any k mer, ideally in constant time per neighboring k mer. One could do so by storing the entire Hamming graph G_H , but it would require large amount of memory. If we assume G as a random string, and errors accumulate independently with probability p_e , then the probability that a node is linked to another is $p_k = \sum_{e=1}^d \binom{k}{e} [0.25^{k-e} 0.75^e + (1-p_e)^{k-e} p_e^e]$, including the chance that another random k mer in the genome is within d Hamming distance of the current k mer and the chance that the current k mer contains up to d errors. Thus, the expected memory usage is $O\left(\binom{|R^k|}{2} p_k\right)$.

Alternatively, we could recover all edges associated with a given k mer α_i by checking

whether each k mer in its complete neighborhood, $\alpha_j \in N_{ci}^d$, exists in R^k . If $\alpha_j \in R^k$, then there is an edge between v_i and v_j in the Hamming graph. This takes $O(\log |R^k|)$ time using binary search. There are $\binom{k}{d} 4^d$ possible candidate mutant k mers for each $\alpha_i \in R^k$, so it takes $O\left(\binom{k}{d} 4^d |R^k| \log |R^k|\right)$ time to identify all edges.

To reduce the average time for inferring N_i^d of α_i , we replicate R^k in memory and sort each replicate on a different subset of positions in the k mer string, using the following strategy:

- a. Store indices 0 to $k - 1$ in a vector A .
- b. Divide A evenly into c ($d < c \leq k$) chunks, each of size $\lfloor k/c \rfloor$ or $\lceil k/c \rceil$.
- c. For every choice of $\binom{c}{d}$ chunks, sort R^k by masking the indices from selected chunks and store the sorted results separately.

To identify N_i^d of α_i , we query α_i against each sorted k -spectrum R_j^k ($0 \leq j < \binom{c}{d}$) by binary search considering only indices used for sorting R_j^k . All k mers that belong to the d -neighborhood of α_i are adjacent to α_i in at least one R_j^k . If sequencing errors accumulate independently, then the expected number of elements of N_i^d found in every R_j^k is $h = |R^k| / 4^{k-d\lceil k/c \rceil}$. Hence, we need approximately $\binom{c}{d} h |R^k| \log |R^k|$ expected time to recover all edges of the Hamming graph, i.e., $O(|R^k| \log |R^k|)$ time assuming both $\binom{c}{d}$ and h are constants. Typically, $|R^k| \ll 4^k$, therefore, choosing a larger c value will use more memory, but less expected run time. As an example, in a real *E. coli* dataset with 160x coverage, storing 13 copies of R^k required only $\sim 560MB$ memory, but the average number of hits per 13mer in each 13-spectrum was less than one. Therefore, identifying each element of N^1 for a 13mer took constant time on average.

The above method provides an exact solution for identifying all edges in the Hamming graph. Alternatively, a simpler recursive approximation derives N^d by inferring N^1 for every

element in N^{d-1} . This strategy might be more biologically meaningful [Qu et al., 2009], but is only an approximation since an edge between two vertices v_i and v_j could be recovered only if there exists a path connecting them such that adjacent vertices represent k mers that differ by exactly one position. In this case, choosing a smaller k , using a larger dataset, or having a higher sequencing error rate all improve the chance to identify all edges.

Tiles are l -concatenations of consecutive or overlapping k mers found in reads. Here, we use one fixed value of l but several different values of l can be used to consider tiles with different lengths. We compute the multiplicities of tiles by a linear scan of every read to record all tiles, followed by a sort of the collected tiles and one linear scan of the sorted list. This process takes $O(|R^{2k-l}| \log |R^{2k-l}|)$ time, where $|R^{2k-l}| = O(\min(4^{2k-l}, n(L - 2k + l + 1)))$. Meanwhile, we record the number of occurrences of each tile, where every position has a quality score exceeding some threshold Q_c . Typically, a quality score is associated with every base of a short read. The score indicates the probability p_e that the corresponding base is sequenced incorrectly. For instance, Illumina GenomeAnalyzer encodes the quality score as $Q = -10 \log_{10}(p_e/1 - p_e)$. A higher score indicates a more reliable base call.

To deal with the double strandedness of the target genome, we consider both the forward and reverse complementary strands of every read. Edge identification in the Hamming graph takes twice the time, but no additional memory is needed since R^k is already generated using both strands.

Phase 2: Error Correction

We use the contextual information in read r to identify sequencing errors through the process of choosing a tiling T_r and comparing it with its d -mutant tilings. In particular, if r contains x errors, and we choose any tiling T_r , then an error free tiling T_s belongs to the

collection of d -mutant tilings of T_r if $d \geq x$. Under the standard assumption of uniform coverage, the tiles of T_s should be substantially more abundant than at least some of the tiles of T_r with errors. After T_s is identified, the true read s can be readily inferred from T_s .

In practice, x could be large, and sequencing errors tend to cluster towards the 3' end of a read. Since we prefer d to be small to limit memory usage, run time and false error detection, it is entirely possible that T_s is not one of the d -mutant tilings of T_r . On the other hand, an alternate tiling Γ_r of r may lower the maximum number of mutations per k mer to below d such that Γ_s with high frequency tilings *is* one of the d -mutant tilings of Γ_r . In the case that there is no such tiling Γ_r , we examine a subset of constituent tiles in Γ_r . If a high coverage path of these selected tiles is present, the tiles are corrected. With some errors removed, a tiling may now exist that contains the true read among its d -mutant tilings.

These observations are sufficient to motivate the following procedure for identifying and correcting read errors. Place a tile t on r and attempt to correct t via comparisons with its d -mutant tiles (tile correction). If t is validated or corrected, move to the next tile in the standard tiling and repeat. If t cannot be corrected or validated, look for an alternative tiling, presumably one that avoids clusters of more than d errors that are thwarting attempts to find error-free tiles within the d neighborhoods. We first describe tile correction in Algorithm 3, then the overall procedure for read correction in Algorithm 2.

Tile Correction. For each tile in R , we have recorded its multiplicity O_c in R and the number O_g of those instances where the quality score of every base exceeds Q_c . If a short read dataset comes with unreliable or missing quality score information, we set $O_g = O_c$. Otherwise, O_g is a better estimate of the number of error-free occurrences of each tile.

The tile correction procedure is given in Algorithm 3. A decision to correct tile t is based

```

1: if  $O_g(t) \geq C_g$  then
2:    $t$  is valid; return.
3: end if
4: if  $t$  has no  $d$ -mutant tiles  $t'$  then
5:   if  $O_g(t) \geq C_m$  then
6:      $t$  is valid; return.
7:   end if
8:   return due to insufficient evidence.
9: end if
10: if  $O_g(t) \geq C_m$  then
11:   Select  $d$ -mutant tiles  $T = \{t' \mid \frac{O_g(t')}{O_g(t)} \geq C_r\}$ .
12:   If  $T = \emptyset$ ,  $t$  is valid; return.
13:   For every  $t' \in T$ , record those positions differed from  $t$  and corresponding quality
       scores.
14:   Correct  $t$  to  $t'$  and return if 1)  $hd(t, t') \leq hd(t, t'')$  for all  $t'' \in T$ . 2) at least one of the
       corrected bases has quality score less than  $Q_m$  in  $t$ .
15:   If  $t'$  is not unique, return due to insufficient evidence (ambiguities).
16: else
17:   if  $t$  has only one  $d$ -mutant tile  $t'$  s.t.  $O_g(t') \geq C_m$  then
18:     Correct  $t$  to  $t'$ ; return.
19:   else
20:     return due to insufficient evidence.
21:   end if
22: end if

```

Algorithm 1: Tile t Error Correction.

on a comparison of the high quality occurrence counts O_g of t compared to its d -mutant tiles. As a rule of thumb, there must be compelling evidence before a correction is made. Any tile is automatically validated if its occurrence count exceeds an upper threshold C_g (lines 1–2). A low occurrence tile with no d -mutant tiles is validated only if it occurs more than a low threshold C_m times (lines 4–6). When there exist d -mutant tiles of t with much higher frequencies (at least C_r times, $C_r > 1$) than t and which differ at low quality bases ($< Q_m$) in t , it is likely that t contains error(s) and the true tile is one of its d -mutant tiles. In such cases, a correction is possible only if there is a unique d -mutant tile with the closest Hamming distance, including a mutation at a low quality base in t (line 14). We will replace t with this tile. Otherwise, we choose not to modify t to avoid false corrections. Similar reasoning applies to t with very low multiplicity (lines 17–18). Since there exist a constant number of d -mutant tiles of t , and correcting t requires comparison of every base between t and $t' \in T$, Algorithm 3 takes constant time.

Read Correction. The overall procedure for correcting a read is given in Algorithm 2. In line 1, an initial tile is chosen and d_1 and d_2 , two parameters specifying the maximum Hamming distance allowed in the two constituent k mers while identifying mutant tiles, are initialized to d . The algorithm terminates when no more plausible tiles can be identified. Within the while loop, d -mutant tiles are identified for the current tile (line 3), which is validated or corrected using Algorithm 3 (line 4). The new placement of tile t_{next} is chosen according to which of the three decisions is made by Algorithm 3 (line 4). Repeated placement of t_{next} according to decisions, [D1] through [D3] (lines 6–8), gradually forms a validated or corrected tiling of read r , although some reads may never be fully validated.

To better understand how the rules in lines 6–8 choose a tiling, we illustrate with an example

- 1: Initialize: $t \leftarrow t_0$, where t_0 is a prefix of r ; $d_1, d_2 \leftarrow d$.
- 2: **while** $t \neq \emptyset$ **do**
- 3: Identify d -mutant tiles of $t = \alpha_1 \parallel_l \alpha_2$ as the set $\{t' = \alpha'_1 \parallel_l \alpha'_2 \mid (\alpha'_1, \alpha'_2) \in N_1^{d_1} \times N_2^{d_2}\}$.
- 4: Correct t (Algorithm 3).
- 5: Based on the decisions made on t in the former step, tile t_{next} of r will be chosen according to [D1]–[D3] as follows. If there is insufficient space to place a tile towards the end of a read r , t_{next} will be chosen as the suffix of r .
- 6: [D1] t is valid: select t_{next} such that the suffix-prefix overlap between t and t_{next} equals α_2 ; $d_1 \leftarrow 0$.
- 7: [D2] t was corrected to $t' = \alpha'_1 \parallel_l \alpha'_2$ and t is replaced with t' in r : select t_{next} such that the suffix-prefix overlap between t' and t_{next} equals α'_2 ; $d_1 \leftarrow 0$.
- 8: [D3] Insufficient evidence to correct t : set t_{next} to be one of the following. Let $r[i_1 : i_2]$ be a maximal validated or corrected region of r that overlaps with the $5'$ region of t , where $i_2 - i_1 + 1 \geq |t|$, and $r[i_2 + 1 : i_3]$ be a maximal uncorrected region from previous iterations due to insufficient evidence.
 - a. If $r[i_2 + 1 : i_3] = \emptyset$, then $t_{\text{next}} \leftarrow r[i_2 - |t| + 2, i_2 + 1]$ and $d_1 \leftarrow 1$.
 - b. If $r[i_2 + 1 : i_3] \neq \emptyset$, then $t_{\text{next}} \leftarrow r[i_3 + 1, i_3 + |t|]$.
- Similarly, if $r[i_1 : i_2]$ overlaps with $3'$ end of t and $r[i_3 : i_1 - 1]$ is a maximal uncorrected region from previous iterations, then if $r[i_3 : i_1 - 1] = \emptyset$, set $t_{\text{next}} \leftarrow r[i_1 - 1 : |t| + i_1 - 2]$ and $d_2 \leftarrow 1$.
- In above cases, t_{next} is valid only if it was not assigned with the same value in previous iterations (unless t_{next} became the suffix of r). If such a choice does not exist, $t_{\text{next}} \leftarrow \emptyset$.
- 9: $t \leftarrow t_{\text{next}}$.
- 10: **end while**

Algorithm 2: Read Error Correction.

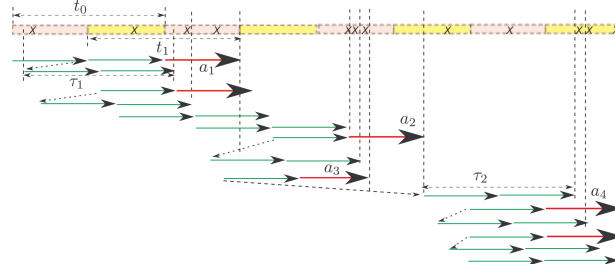


Figure 2.2 An illustration of choosing a tiling of a read for $d = 1$. A read is represented on top by a concatenation of rectangles, where each rectangle denotes a k mer. Each tile is represented by a concatenation of two adjacent arrows, which denote its k mer composition. For simplicity of illustration, we choose the read length to be divisible by k and each tile is a 0-concatenation of two adjacent k mers. X 's denote sequencing errors. Each bold arrow, a_i ($1 \leq i \leq 4$), denotes tile with insufficient evidence for correction. The placement of an alternative tile is indicated by a dotted arrow.

in Fig. 2.2 for $d = 1$. An initial tile t_0 is chosen as shown. Since there exists one error in each of the constituent k mers, t_0 can be corrected. Tile t_1 is chosen as the next tile according to [D2], but two sequencing errors within the second k mer of t_1 lead to an inconclusive decision. Hence, t_1 is not selected in the tiling and an alternative tile τ_1 is chosen according to [D3(a)]. The algorithm iterates and if tiles can be validated or corrected at every stage, we are able to complete a tiling moving from $5'$ to $3'$ along the read. Unfortunately, non-uniform coverage and the existence of more than one reasonable d -mutant tile can lead to an inconclusive decision in algorithm 3 regardless of tiling choice. In Fig. 2.2, the $5'$ to $3'$ tiling encounters an inconclusive dead-end at arrow a_3 . To move past dead-end tiles, a non-overlapping tile τ_2 is chosen by [D3(b)]. A small unvalidated gap is left in the middle of the $5'$ to $3'$ tiling of this read. The example shows only the tiling from $5'$ to $3'$. The same strategy is applied in the $3'$ to $5'$ direction.

We briefly analyze the run time of Algorithm 2. Tiles are sorted, so tile information is accessed in $O(\log(nL))$ time. Therefore, line 3 requires $O(\log(nL))$ time. Once some tiles

have been corrected, the search space for new d -mutant tiles shrinks when d_1 or d_2 is set to 0 in line 5. The maximum number of non-overlapping tiles in a tiling is the constant $L/|t|$. Hence the time spent in correcting each read is $O(\log(nL))$, and the overall run time of Algorithm 2 is $O(nL \log(nL))$.

Choosing Parameters. Although analytical calculations like those adopted in existing methods can be used to choose parameters of Algorithm 3, we choose their values based on the input data to help avoid the unrealistic assumptions of uniformly distributed read errors and uniform genome coverage. Given short read data R , we examine the empirical distribution of quality scores and choose threshold Q_c such that a given percentage (e.g., 15% to 20%) of bases have quality score value below Q_c . This value could be adjusted to consider the error rates of the particular next generation sequencing equipment in use. Given Q_c and counts of high quality tile occurrences, we choose C_g so that only a small percentage (e.g., 1% to 3%) of tiles have high quality multiplicity greater than C_g . C_m is chosen so that a larger percentage (e.g., 4% to 6%) of tiles occur more than C_m times in R . As C_m value decreases, more errors are corrected at the cost of an increased risk of false error correction. The specific values chosen depend on the histogram of tile occurrences. By default, we set $C_r = 2$ such that a low frequency tile could only be corrected to a tile with at least twice the frequency. Increasing C_r improves the confidence in error correction. Finally, we choose $k = \lceil \log_4 |G| \rceil$ when an estimate of the length of the genome is available, otherwise, a number between 10 to 16 should work. Tile size is approximately $2k$, so k mer overlap is 0 to a few bases. The maximum Hamming distance d is set to one by default. But when k is chosen to be relatively large (e.g., 14 to 16), increasing d allows us to identify more sequencing errors but incurs a longer run time and increases the risk of false error prediction.

Overall Complexity

Combining the analysis for each step, the overall run time is $O(nL \log(nL))$ and the space usage is $O(|R^k| + |R^{l}|)$.

When the collection of input short reads R does not fit in main memory, we propose a divide and merge strategy where R is partitioned into chunks small enough to occupy just a portion of main memory. For each chunk, we stream through each read and record the k -spectrum and tile information, merging it with the data from previous chunks. Reads need not be stored in memory after they have been processed. A similar strategy is applied for error correction: R is reloaded into memory in chunks, tilings and d -mutant tilings are inferred for each read, and errors are corrected as warranted.

2.4 Results

We evaluated Reptile on several Illumina/Solexa datasets and compared the results with SHREC [Schröder et al., 2009] version 2.0, a recent high quality short read error correction method that is itself shown to give superior results over prior k -spectrum approaches. We omitted evaluation on simulated data because simulations with random errors or synthetic genomes do not accurately reflect actual short read sequencing errors [Dohm et al., 2008], and could even be misleading. Our test datasets are Illumina generated short reads of well-characterized, Sanger assembled bacterial genomes. Knowledge of the genomes is needed for determining the accuracy of the error correction methods. The six experimental datasets, downloaded from the sequence read archive at NCBI, are listed in Table 2.1. Datasets $D1$ (Accession Number: SRX000429), $D2$ (SRR001665_1), $D5$ (SRR022918_1) and $D6$ (SRR034509_1) are Illumina reads from the *E. coli str. K-12 substr* (NC_000913) genome (~ 4.64 Mbp); datasets

Table 2.1 Experimental Datasets

Data	Genome	Read Length	Number of Reads	Discarded Reads	Cov.	Error rate
<i>D1</i>	<i>E. coli</i>	36bp	20.8M	107.7K	160x	0.6%
<i>D2</i>	<i>E. coli</i>	36bp	10.4M	48.3K	80x	0.6%
<i>D3</i>	<i>A. sp.</i>	36bp	17.7M	456K	173x	1.5%
<i>D4</i>	<i>A. sp.</i>	36bp	4.0M	0	40x	1.5%
<i>D5</i>	<i>E. coli</i>	47bp	7.0M	32.7K	71x	3.3%
<i>D6</i>	<i>E. coli</i>	101bp	8.9M	1.44M	193x	2.2%

Error rate is estimated by mapping the reads to the corresponding genome using RMAP, and finding mismatches based on uniquely mapped reads.

D3 (SRR006332) and *D4* are Illumina reads from the *Acinetobacter sp. ADP1* (NC_005966) genome (~ 3.6 Mbp). The first four datasets are generated by Solexa 1G Genome Analyzer, where each read has the same length 36bp. The latter two datasets are generated using the more recent Illumina Genome Analyzer II, with read lengths of 47bp in *D5* and 101bp in *D6*. *D1* has high coverage and low error rate. *D2* has typical coverage and low error rate. *D3* has high coverage and high error rate. *D4* is derived from *D3* by randomly selecting short reads amounting to 40x coverage. This is done for evaluating performance on a low coverage, high error rate dataset. Both *D5* and *D6* have higher error rates. In addition, over 13.9% of the reads in *D6* contain ambiguous nucleotides, denoted by character *N*. Since SHREC cannot process non-*ACGT* characters, we eliminated all reads with ambiguous bases, even though Reptile has no such limitation. The number of discarded reads is indicated in column 5, Table 2.1.

Similar to Schröder et al. [2009], we evaluated error correction results with the aid of RMAP (v2.05) [Smith et al., 2008], which maps short reads to a known genome by minimizing mismatches. We allowed up to five mismatches per read in the first four datasets and allowed up to ten mismatches (default value of RMAP) in *D5* and fifteen mismatches in *D6* since the reads are longer in the latter two datasets. Reads that could not be mapped to the genome,

Table 2.2 Results of mapping each dataset to the corresponding genome using RMAP.

Data	Allowed mismatches	Number of reads*	Uniquely mapped reads	Ambiguously mapped reads
<i>D1</i>	5	20,708,709	96.5%	2.5%
<i>D2</i>	5	10,359,952	96.7%	2.5%
<i>D3</i>	5	17,675,271	79.9%	1.5%
<i>D4</i>	5	4,000,000	84.1%	1.6%
<i>D5</i>	10	7,049,153	62.5%	1.5%
<i>D6</i>	10	8,874,761	63.5%	1.2%
	15		68.8%	1.4%

*Number of reads containing no ambiguous bases.

or that map to multiple locations, are discarded. The mismatches between uniquely mapped reads and the genome are considered read errors. Quality of the datasets varied as shown in Table 2.2, with the percentage of reads that are uniquely mapped ranging from 62.5% to 96.7%. The large percentage of un-mappable reads, the higher error rates as well as the large percentage of reads with ambiguous bases indicate that *D5* and *D6* have lower quality than *D1* to *D4*.

Since the goal of error correction is to identify and correct each erroneous nucleotide, we assess the quality of error correction at the base level. A *True Positive* (TP) is any erroneous base that is changed to the true base, a *False Positive* (FP) is any true base changed wrongly, a *True Negative* (TN) is any true base left unchanged, and a *False Negative* (FN) is any erroneous base left unchanged. Then $Sensitivity = TP / (TP + FN)$ and $Specificity = TN / (TN + FP)$. Note that these definitions are different from those used by Schröder et al. [2009], which target read level error detection (whether a read is flagged as containing an error or not). This is a less stringent measure because any read containing errors was classified as TP provided at least one of its errors was detected and irrespective of whether they were accurately corrected

or not.

We propose two additional measures for assessing the quality of error correction:

- *Erroneous Base Assignment* (EBA): Let n_e denote the number of erroneous bases that are correctly identified but changed to a wrong base. Then, $EBA = n_e / (TP + n_e)$ reflects how well we are able to correct an erroneous base to the true base after a sequencing error has been identified. A lower value of EBA indicates a more accurate base assignment.
- *Gain*: $(TP - FP) / (TP + FN)$. This measures the percentage of errors effectively removed from the dataset, which is equivalent to the number of errors before correction minus the number of errors after correction divided by the number of errors before correction. Clearly, Gain should approach one for the best methods, but may be negative for methods that actually introduce more errors than they correct.

We regard these measures as important because they penalize failing to detect an erroneous base, correctly detecting an erroneous base but wrongly correcting it, and characterizing a correct base to be an erroneous base. In particular, we strongly advocate the Gain measure as it captures data quality post error correction compared to the quality prior to the correction.

The results of running Reptile and SHREC on the six datasets are summarized in Table 2.3. Due to the larger memory usage of the SHREC program, we were not able to obtain results for $D3$, $D5$ and $D6$. In all other cases, Reptile had higher Gain and lower EBA than SHREC. With other parameters fixed in Reptile, we varied maximum d value used for inferring Hamming graph in $D1$ and $D2$. As expected, the run time significantly increased as d increased, since the size of d -neighborhood for each k mer increased. Also, we see an increase in both TP and FP and four to five times higher EBA, indicating that when we increase the search space, we run the risk of false error detection and correction but increase the chance to identifying more

Table 2.3 Results of Reptile and SHREC on Illumina Sequenced Short Reads.

Data (Cov)	Method (<i>d</i>)	TP	FN	FP	TN	EBA (%)	Sensitivity	Specificity	Gain	CPU Hours	Memory (GB)
<i>D1</i> (160x)	SHREC	2819754	1183861	667435	740842474	1.794	70.4%	99.9%	53.8%	-	> 8
	Reptile (1)	3164394	839221	133558	741376351	0.007	79%	99.9%	75.7%	0.79	1.1
	Reptile (2)	3457717	545898	245417	741264492	0.028	86.4%	99.9%	80.2%	2.49	1.1
<i>D2</i> (79.5x)	SHREC	1303505	422337	251228	370981202	1.549	75.5%	99.9%	61.0%	3.6	7.1
	Reptile (1)	1169256	556586	44959	371187471	0.009	67.8%	99.9%	65.2%	0.35	0.84
	Reptile (2)	1315277	410565	91205	371141225	0.042	76.2%	99.9%	70.9%	1.23	0.84
<i>D3</i> (172.5x)	SHREC	-	-	-	-	-	-	-	-	-	-
	Reptile (1)	7138883	2361813	1138666	610144638	0.013	75.1%	99.8%	63.2%	1.66	2.2
<i>D4</i> (40x)	SHREC	1473252	530736	613921	141382091	1.306	73.5%	99.6%	42.9%	2.78	7.6
	Reptile (1)	1422949	581039	222218	141773794	0.091	71%	99.8%	59.9%	0.26	0.66
<i>D5</i> (71x)	SHREC	-	-	-	-	-	-	-	-	-	-
	Reptile (1)	3551764	3189748	985674	323583005	0.017	52.7%	99.7%	38.1%	0.94	1.9
<i>D6</i> (193x)	SHREC	-	-	-	-	-	-	-	-	-	-
	Reptile (1)	17158925	2947342	1298891	874945703	0.01	85.3%	99.9%	78.9%	2.76	4.6

errors.

An inherent difficulty in using any method is the challenge of choosing optimal parameters. The results reported in Table 2.3 are obtained when using the parameter choices suggested in the Methods section. To show that even better performance is possible, we applied a series of parameter choices to dataset *D3* (Fig. 2.3). Gain improved from 63% with the default parameters to as high as 72%. We chose to report on Reptile using the default parameters for all cases in Table 2.3 because it is unfair to choose optimal parameters for each individual case based on our knowledge of the genome, which would generally not be known. Similarly, we used the default parameter settings for SHREC. Using a different combination of parameters may vary the results of both SHREC and Reptile. In this chapter, we have presented a method to select parameters for Reptile based on known quantities such as kmer frequency and quality score histograms. A similar guidance is needed for the SHREC program and is beyond the scope of the paper. Note that we do not take into account improved results that can only be obtained by the knowledge of the genome (see Fig. 2.3). One can observe that our method of parameter estimation based on statistics from the dataset is performing better than analytical calculations based on the assumptions of uniform error distribution and uniform coverage of genome by reads.

In addition, we compared the run time and memory usage of SHREC and Reptile. SHREC is a multi-threaded program while the current release of Reptile can only use a single core. Hence, we report run times in total CPU hours. Both methods were run on a SUN Fire X2200 workstation with dual quad-core 2.3 GHz AMD Barcelona 3 processors with 8 GB RAM and 4 GB swap memory, running Debian GNU/Linux x86_64. Results in Table 2.3 show Reptile is three to ten times faster and uses eight to eleven times less memory than SHREC. As expected,

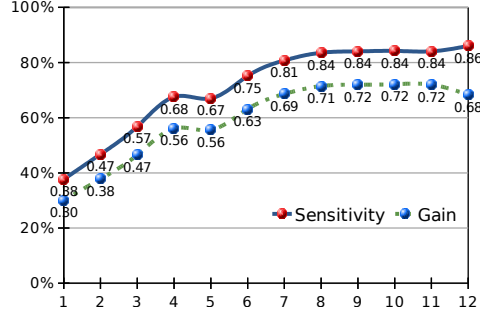


Figure 2.3 Gain and Sensitivity vs. Different Parameter Choices for $D3$. The first 11 sample points use parameters $k = 11, d = 1, |t| = 22$, and (C_m, Q_c) values $(14, 60), (12, 60), (10, 60), (10, 55), (8, 60), (8, 55), (8, 50), (8, 45), (7, 45), (6, 45), (5, 45)$, respectively. The last sample point uses parameters $(k = 12, d = 2, |t| = 24, C_m = 8, Q_c = 45)$.

memory usage of Reptile is associated with the length of the genome and the number of errors in the data (compare $D1$ and $D2$), while the memory usage of SHREC increased with the number of reads, genome length and sequencing errors. In addition, although $D1$ contains many more reads (20.8M) than $D3$ (17.7M), the higher error rate significantly increased memory usage in both methods.

To enable fair comparison with SHREC, the above experiments were carried out by excluding all reads containing ambiguous bases. However, Reptile does have functionality to deal with ambiguous bases, which is useful in the following cases: 1) if a read contains few ambiguous bases, the surrounding high quality regions provide sufficient information to infer correct bases by referencing the k -spectrum; 2) in some datasets, neglecting reads containing ambiguous bases leads to excessive loss of data, which further distorts uniformity of the sampling. For instance, as much as 13.9% reads in dataset $D6$ contain N 's.

Reptile attempts to correct ambiguous bases in regions where their density is low. If a read contains too many ambiguous bases, it is low quality and untrustworthy. Some reads may have ambiguous bases clustered in some region, e.g., the 3' region, while other parts may still be of

good quality. It is more meaningful to try correcting ambiguous bases in the latter parts alone, since a cluster of ambiguous nucleotides in a read makes it unlikely to pinpoint other reads that have the same genomic co-location. Formally, Reptile attempts to correct an ambiguous base b of read r , if in any substring $r[i : i + w - 1]$ that contains b , there are no more than d ambiguous bases. The ratio of d to w constrains the maximum density of ambiguous bases allowed in attempting error correction. By default, w is set to k (to equal k mer length), while d is set to the maximum Hamming distance allowed (see Methods section). To implement the above idea, all ambiguous bases satisfying the density constraint are changed to one of the bases from the set $\{A, C, G, T\}$ initially (default “A”), and will be validated or corrected later by the algorithm.

To test the accuracy of this procedure as well as study the effect of the choice of the default base, we conducted Reptile runs on the full datasets of $D2$ (36bp) and $D6$ (101bp) by setting the ambiguous bases to the chosen default. The results are presented in Table 2.4. The default base used is shown under Column “N”. *Accuracy* is defined to be the percentage of ambiguous bases that have been successfully corrected (again, only reads uniquely mapped by RMAP are considered as truth is unknown otherwise). The last four columns are as defined in Table 2.3. As can be observed from Table 2.4, 1) the accuracy of ambiguous base correction is high and consistent with the overall EBA rate, 2) changing the default base slightly influenced the results due to the resulting differences in k -spectrum composition, and 3) the sensitivity and Gain values are slightly lower than reported in Table 2.3, mainly because the ambiguous bases that were left uncorrected by Reptile could sometimes be uniquely mapped to the reference genome using RMAP, hence increasing the FN value.

Table 2.4 Quality of ambiguous base correction using Reptile

Data	N	Accuracy	Sensitivity	Specificity	Gain	EBA
<i>D2</i>	A	99.98%	66.4%	100%	63.7%	0.01%
	C	100%	66.4%	100%	63.8%	0.01%
	G	100%	66.4%	100%	63.7%	0.01%
	T	100%	66.3%	100%	63.7%	0.01%
<i>D6</i>	A	99.99%	85.1%	99.8%	78.5%	0.01%
	C	99.99%	85.2%	99.8%	78.4%	0.009%
	G	100%	85.3%	99.8%	78.5%	0.01%
	T	99.99%	85.3%	99.8%	78.5%	0.009%

2.5 Discussions

The proposed error correction algorithm is conservative because it avoids changing bases unless there is a compelling under-representation of a tile compared to its d -mutant tiles. Actual errors in read r cannot be corrected if r occurs in a very low coverage region of the genome or there exist multiple candidate d -mutant tiles, probably because of genome repetition. On the other hand, a tile may be miscorrected if it contains a minor variant of a highly repetitive element in the genome or it traverses a low coverage region that is similar to other regions with normal coverage. Our method is not unique in being challenged by nonuniform coverage on repetitive genomes. Error correction for highly repetitive genomes is essential for successfully assembling larger eukaryotic genomes but none of the existing methods successfully addresses this problem, including Reptile.

Short read mapping provides a reasonable method to evaluate error correction methods in well assembled, low repetition genomes. Nevertheless, it is not possible to unambiguously determine all errors. There are natural polymorphisms among bacterial lines, and some presumed polymorphisms may be unrecognized assembly errors. Furthermore, the mapping software chooses among alternative mappings by invoking parsimony, but there is some chance

that the true number of errors is less than the minimum. Lastly, mapping software cannot map reads that contain more than a constant number of substitutions, typically just two, with full sensitivity, although we considered five here and tested as many as fifteen with similar results. Despite these limitations, we believe that most errors are correctly identified, and this approach can provide a fair comparison of error correction methods.

We and others [Smith et al., 2008] have found that sequence quality scores provide valuable information. Our use of quality scores probably helped us account for the error patterns in next generation sequencing data [Dohm et al., 2008] without explicitly modeling them. However, it has been observed [Dohm et al., 2008] that high quality scores may be too optimistic and low quality scores too pessimistic in estimating sequencing errors in Solexa data. Since quality scores may not be precise measures of misread probabilities, the current version of Reptile uses quality score information in a very simple manner, but can be modified to make more sophisticated use of quality scores if warranted. Finally, although quality scores are needed to run Reptile, it can be run effectively without scores by setting all quality scores and the threshold Q_c to the same value.

2.6 Our Contributions

We developed a novel approach, termed Reptile, for error correction in reads generated using next generation sequencing techniques. Reptile is applicable to datasets where insertion and deletion errors occur rarely. This property has been shown to be the case for Illumina reads with length $\sim 36\text{bp}$ and $\sim 50\text{bp}$ by Dohm et al. [2008] and by our own experiments with a set of Illumina datasets. Reptile works with the spectrum of k mers from the input

reads, and corrects errors by simultaneously examining 1) Hamming distance based correction possibilities for potentially erroneous k mers, and 2) neighboring k mers from the same read for correct contextual information. By not needing to store input data, Reptile has the favorable property that it can handle data that does not fit in main memory. In addition to sequence data, Reptile can make use of available quality score information. Our experiments show that Reptile outperforms previous methods in the percentage of errors removed from the data and the accuracy in true base assignment. In addition, The time complexity of Reptile is $O(N \log N)$, where N is the total number of input characters. In practice, a significant reduction in run time and memory usage have been achieved compared to previous methods, making it more practical for short read error correction when sampling larger genomes.

The proposed method is made available through the software package Reptile at “<http://aluru-sun.ece.iastate.edu/doku.php?id=reptile>”.

CHAPTER 3. Repeat-aware Sequencing Error Detection and Correction

Repeats in genomes can lead to mishandling of errors in many ways. Nearly identical repeats can easily be mistaken to be sequencing errors. Even when errors are rare, an erroneous k mer may appear at a moderate frequency if it has few nucleotide differences from one or more valid k mers that have a high frequency of occurrence in the genome. The problem of detecting and correcting sequencing errors among reads in the presence of repeats has so far not been adequately addressed. Nevertheless, repeats are widely prevalent, even in some viral genomes such as *N. meningitidis*. Other genomes, like those of plants, are known for their high repeat content; for instance, an estimated 65-80% of the maize genome is spanned by repeats, which makes the assembly, mapping and error detection and correction tasks difficult. Although packages like FreClu [Qu et al., 2009] and Recount [Wijaya et al., 2009] could be potentially adapted to consider repeats, they are specifically designed for transcriptome data and correct read counts rather than identify and correct erroneous bases within reads. Moreover, insufficient replication of full length reads in genomic data prevents these methods from accurately estimating model parameters.

In this chapter, we address the problem of identifying and correcting sequencing errors in short reads from genomes with different levels of repetition, particularly for reads produced by the widely used Illumina Genome Analyzer platform. Similar to existing approaches, we decompose the input reads into k mer substrings and count the number of times Y_l each k mer

x_l occurs in the reads. However, instead of inferring erroneous k mers based on these observed occurrence frequencies [Chaisson et al., 2004, Schröder et al., 2009], we developed a maximum likelihood estimate of the expected number T_l of *attempts* to read x_l , including both attempts that resulted in error-free reads and erroneous reads. In addition, we propose a new method to choose the threshold, which can be used to identify erroneous k mers as those x_l 's for which T_l 's are lower than the threshold. We demonstrate that using estimates of read attempts enables more accurate detection of sequencing errors than using observed frequencies for a wide choice of thresholds. We further develop an error correction method to transform erroneous bases in each read to the correct ones and compare the results with SHREC [Schröder et al., 2009] and Reptile [Yang et al., 2010], two of the most recent error correction methods. The results demonstrate significant improvement in error correction capabilities for genomes with high repeat content.

3.1 Preliminaries

The notations used in this chapter are consistent with those defined in 2.1. Each k mer $x_l \in R^k$ has α_l occurrences in G and Y_l instances observed in read set R . Define $s_l = \frac{\alpha_l}{|G|^k - k + 1}$, the probability that a random k -length fragment in the genome is k mer x_l . Occurrence frequency α_l , or equivalently s_l , is unobserved, but of paramount interest. Indeed, if we knew $s_l = 0$, but observed $Y_l > 0$, then we would know each observed instance of x_l contains at least one misread base. Under the assumption that errors are rare, it makes sense to label k mers x_l with $Y_l < M$ as errors, where M is chosen such that $P(Y_l < M \mid s_l > 0)$ is reasonably small. Since s_l is unknown, the threshold M is set *ad hoc*, based on training or simulated data [Chaisson et al., 2004], or analytical calculations [Chin et al., 2009, Schröder et al., 2009] assuming the genome

to be a random sequence and errors to be distributed uniformly in the reads. In practice, these assumptions do not hold true. Moreover, the problem of misread k mers contributing to the observed frequencies Y_l (see Fig 3.1) is exacerbated in repetitious genomes where k mers with high genomic occurrence may result in generation of the same misread multiple times [Chaisson et al., 2004].

We will develop a model that estimates the expected number of *attempts* T_l to read each k mer x_l . The threshold is then applied to each estimated T_l instead of the corresponding observed Y_l . The model we propose is similar to that of RECOUNT [Wijaya et al., 2009] used to correct next generation short read counts. Both models derive from a method originally meant to detect sequencing errors in SAGE libraries [Beissbarth et al., 2004]. Our model differs from the previous models in that it works with k mers rather than full reads, since there is insufficient replication of full length reads in genomic data (as compared to transcriptome data). In addition, instead of assuming the misread bases to be drawn from $\{A, C, G, T\}$ with equal probability, we propose a parametric error model that can be trained from the reads produced by the control lane (e.g. using the Illumina Genome Analyzer) in the same experiment. This strategy has already proven to be useful in several pioneering works [Qu et al., 2009, Weese et al., 2009]. In addition, we will show that the model is somewhat robust to incorrect assumptions in the underlying error model. A further step is to modify erroneous bases to their true forms in each read. This task has rarely been attempted previously for repetitive regions. We propose a method that utilizes transition probabilities and the contextual information of individual reads to achieve this goal. Like others, we ignore insertion and deletion errors assuming they are rarely produced by next-generation sequencing technology, which is true for reads from the Illumina Genome Analyzer [Dohm et al., 2008].

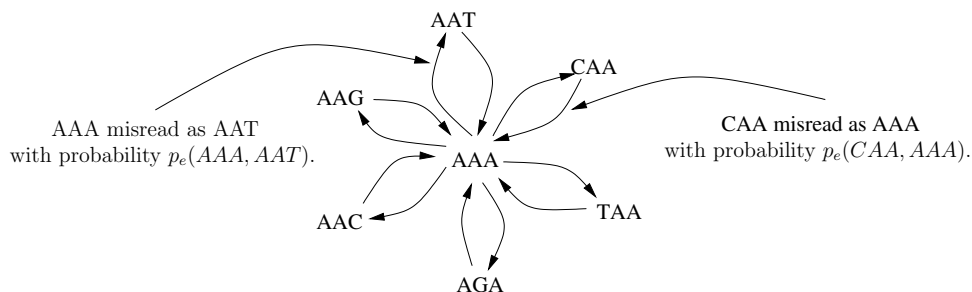


Figure 3.1 A k mer neighborhood. The neighborhood of trimer AAA is the collection of trimers in R^3 that have a nonvanishing chance of being misread as AAA, in this case trimers with at most one substitution.

3.2 Error Model

The simplest error model posits that sequencing errors occur independently at all sites with constant probability p_e . Let $p_e(x_m, x_l)$ be the probability that x_m is misread as x_l . This model produces symmetric misread probabilities:

$$p_e(x_m, x_l) = p_e(x_l, x_m) = (1 - p_e)^{k-d(x_l, x_m)} \left(\frac{p_e}{3}\right)^{d(x_l, x_m)} \quad (3.1)$$

where $d(\cdot, \cdot)$ denotes the Hamming distance between two k mers. It is known, however, that short read technology produces errors with distinct patterns [Dohm et al., 2008]. As a first approximation, we assume that errors strike sites in the k mer independently, but with varying probabilities. For example, we observe in dataset preparation section that errors cluster in the 3' portion of reads and, consequently, k mers. Let $q_i(\alpha, \beta)$ be the probability that nucleotide α at position i of a k mer is (mis)read as nucleotide β , with $\sum_{\beta} q_i(\alpha, \beta) = 1$. Then, the misread probability is

$$p_e(x_m, x_l) = \prod_{i=1}^k q_i(x_{mi}, x_{li}).$$

These misread probabilities are no longer symmetric, and can be arranged into a $4^k \times 4^k$ matrix P_e , where non-zero entries in the l th row identify all possible ways to (mis)read k mer x_l .

We now discuss some ways to reduce and simplify the calculations. We observe substitution errors are relatively rare, so misread k mers generally contain far fewer than k errors. Thus, when considering possible origins of a misread k mer, we can safely restrict our attention to k mers within some Hamming distance d_{\max} from the current k mer. Capping the maximum distance between k mers at d_{\max} induces a sparse P_e , whose entries are normalized by dividing each row by the corresponding row sum. Finally, we ignore k mers that are not observed in the data (i.e. $Y_m = 0$ or, equivalently, $x_m \notin R^k$), so the (incomplete) neighborhood of k mer x_l , denoted by $N_l^{d_{\max}}$, is given as $\{x_m \in R^k : d(x_l, x_m) \leq d_{\max}\}$. Failure to include unobserved k mers could bias estimation of α_l by ignoring k mers actually present in G and capable of contributing to Y_l . However, the bias cannot be large since α_m must be small, otherwise Y_m would not be zero.

After considering errors and applying the simplifications, the counts Y_l follow a Multinomial distribution

$$\mathbf{Y} = (Y_1, \dots, Y_{|R^k|}) \sim \text{Multinomial}(n(L - k + 1), \mathbf{p}),$$

but unobserved k mers are ignored and the probability vector $\mathbf{p} = (p_1, p_2, \dots, p_l, \dots, p_{|R^k|})$ depends on the k mer neighborhood. Specifically,

$$p_l = \sum_{x_m \in N_l^{d_{\max}}} s_m p_e(x_m, x_l),$$

where $\mathbf{s} = (s_1, \dots, s_{|R^k|})$ is restricted to the set of observed k mers R^k . It becomes clear that when x_l is surrounded by highly repetitious x_m with large s_m , then Y_l may exceed threshold M

because of high misread occurrence with probability $s_m p_e(x_m, x_l)$. Thus, when errors combine with repeats, it is more appropriate to apply a threshold to estimates of the parameters s_l than observed Y_l .

The observed log likelihood, $l(\mathbf{s} \mid \mathbf{Y})$, involves a mixture over the neighborhood (Fig. 3.1) of k mers that could be (mis)read as k mer x_l ,

$$l(\mathbf{s} \mid \mathbf{Y}) \propto \sum_{x_l \in R^k} Y_l \ln \left[\sum_{x_m \in N_l^{d_{\max}}} s_m p_e(x_m, x_l) \right].$$

This setup lends itself to maximum likelihood estimation via the EM algorithm [Dempster et al., 1977]. The update equations are adapted from Beissbarth et al. [2004] using a different error model and are given as follows:

The expectations of hidden data Y_{lm} obtained in the E step are

$$E[Y_{lm} \mid \mathbf{Y}, \mathbf{s}] = \frac{Y_m s_l p_e(x_l, x_m)}{\sum_{x_{l'} \in N_m^{d_{\max}}} s_{l'} p_e(x_{l'}, x_m)}$$

The M step yields maximum likelihood estimates

$$\hat{s}_l = \frac{\sum_{m: x_m \in N_l^{d_{\max}}} E[Y_{lm} \mid \mathbf{Y}, \mathbf{s}]}{n(L - k + 1)}$$

Notice the estimated expected number of attempts to read k mer x_l is $T_l = \hat{s}_l n(L - k + 1)$, directly proportional to \hat{s}_l and sitting on the same scale as Y_l . In fact, by observing the E step is unchanged and the log likelihood $l(\mathbf{s} \mid \mathbf{Y})$ is computed up to an additive constant when s_l is replaced with T_l , we use the EM algorithm to compute T_l directly. For inference, we apply the threshold to estimates $\mathbf{T} = (T_1, \dots, T_{|R^k|})$ rather than $\hat{\mathbf{s}}$, to more easily compare

our method with thresholding on \mathbf{Y} .

The algorithm is initialized by setting $T_l = Y_l$ and iterating until the log likelihood converges.

3.3 Error Detection and Correction

Error detection, in practice, requires a method to choose a threshold M that minimizes the number of wrong decisions when classifying k mers as erroneous or not. We discuss a model-free method for estimating the threshold M in section 3.7, but no results presented in this chapter use estimated thresholds.

To correct errors, consider each of the nucleotides in a read r . Each nucleotide appears in at least one and up to k k mers. Suppose the nucleotide at position $1 \leq i \leq L$ of the read appears at position $1 \leq t \leq k$ of k mer x_l . The probability that the true nucleotide at position t was b prior to possible misread is

$$p_{it}(b) = \frac{\sum_{x_m \in N_l^{d_{\max}}, x_{mt}=b} \alpha_m p_e(x_m, x_l)}{\sum_{x_m \in N_l^{d_{\max}}} \alpha_m p_e(x_m, x_l)},$$

where estimates T_m are substituted for the unknown α_m . Since multiple overlapping k mers provide non-independent information about the base at position i , we average across available t to obtain distribution $p_i(b)$. If $\arg\max_b p_i(b) \neq r[i]$, then we declare nucleotide $r[i]$ misread and correct it to $\arg\max_b p_i(b)$. To limit computations, we apply this method to reads likely to contain at least one erroneous k mer, as identified with a liberal threshold M .

Table 3.1 Experimental datasets.							
Dataset	Type	Ref genome	Genome length	Repeats	Repeat Types (length, multiplicity)	C	Number of reads
$D1$	1(a)	-	1M	20%	(1000, 200)	80x	2.2M
$D2$	1(a)	-	1M	50%	(500, 400), (1500, 200)	80x	2.2M
$D3$	1(a)	-	1M	80%	(500, 400), (1500, 200) (3000, 100)	80x	2.2M
$D4$	1(b)	<i>NM</i>	2.1M	-	-	80x	4.8M
$D5$	1(b)	<i>Maize</i>	418K	-	-	80x	0.92M
$D6$	2	<i>E. coli</i>	4.6M	-	-	160x	20.7M

‘-’ denotes the information that is not quantified; K: thousand; M: million.

3.4 Results

3.4.1 Dataset Preparation

In order to test our model, we compiled various simulated and real datasets (Table 3.1). The datasets are classified into the following types (Table 3.1, column 2). Type 1 are simulated Illumina reads from (a) synthetically constructed genomes embedded with various types of non-overlapping repeats, and (b) previously sequenced genomes known to be rich in repeats. Type 2 are actual Illumina reads from a previously sequenced genome with a low degree of repetition.

Reference genome preparation. The reference genomes of type 1(a) were initially generated using the nucleotide distribution of a piece of B73 maize genome (A: 28% C:23% G: 22% T: 27%). Then, repeat regions of different lengths and multiplicities (Table 3.1, column 6) derived from the same nucleotide distribution were embedded at random locations in these reference genomes. The reference genome *N. meningitidis* (NC_013016) of $D4$ is known to be a small, repeat rich, viral genome. The maize genome is known to contain up to 80% repeats and only the relatively unique regions have been fully assembled. Hence, we concatenated the first 20 contigs from Chromosome 1 of the B73 assembly, and removed all non-ACGT characters to

form the reference genome of *D5*.

Short read preparation. The simulated Illumina reads (type 1) were produced by first estimating an error distribution from a real Illumina short read dataset, then simulating uniformly distributed reads of the reference genomes with these error rates. We used the RMAP software [Smith et al., 2008] to map Illumina data (Sequence Read Archive ID: SRX000429) to the reference genome *E. coli str. K-12* allowing up to three mismatches. We were able to map 98.5% of reads; this percent is increased to 99.1% by allowing up to ten mismatches. However, allowing more mismatches increases the chance of a mismapped read since reads are only 36bp, and typically, mapping software can work at full sensitivity for up to two mismatches. Unmapped reads were discarded, and all remaining reads were assumed correctly mapped. By comparing the mapped reads to the reference genome, we estimated $L \times 4 \times 4$ misread probability matrices $\mathbf{M} = (M_1, M_2, \dots, M_L)$, where L is the read length and each entry (α, β) ($\alpha, \beta \in \{A, C, G, T\}$) in misread probability matrix M_i ($1 \leq i \leq L$) is the probability a nucleotide α on the reference genome is (mis)read as β at position i in the read. This is calculated as the total number of times α is read to be β at position i among all mapped reads, divided by the number of times the corresponding position of the reference genome is α . Finally, we simulated Illumina sequencing to generate N reads by applying \mathbf{M} to N uniformly distributed L -substrings in the reference genome.

Rationale. Simulated data are essential because highly repetitive genomic regions, for which our error model is designed, are often masked prior to assembly. Even when assembly can be done, accurate mapping of sequenced reads back to the assembly is difficult when genomes are repetitive [Zhi et al., 2007]. Under these conditions, only simulation can provide unambiguous error information. Type 1(a) datasets were prepared such that they emulate repeat

Table 3.2 Estimated error probabilities $q_i(\cdot, \cdot)$, position $i = 11$.
E. coli str. K-12 substr. *Acinetobacter sp. ADP1*

$\times 10^{-2}$	A	C	G	T	$\times 10^{-2}$	A	C	G	T
A	98.96	0.63	0.18	0.23	A	96.18	2.53	0.19	1.10
C	0.15	99.60	0.10	0.15	C	0.20	99.32	0.08	0.40
G	0.05	0.17	99.25	0.53	G	0.12	0.30	97.60	1.98
T	0.05	0.19	0.18	99.58	T	0.09	0.18	0.13	99.60

content ranging from a microbial genome with low repeats to a highly repetitive plant genome. However, to inject reality wherever possible, the reference genomes of Type 1(b) were selected from the previous assemblies. Lastly, the type 2 dataset demonstrated the applicability of our model to real, although non-repetitive, real read data.

3.4.2 Error Detection and Correction Results

Our model accommodates sequencing errors via the misread probabilities $p_e(x_m, x_l)$ between any two k mers x_m and x_l . To calculate $p_e(x_m, x_l)$, we need to specify the position specific misread probabilities, $q_i(\alpha, \beta)$, $1 \leq i \leq k$, $\alpha, \beta \in \{A, C, G, T\}$, for each position of a k mer. Ideally, we would set $q_i(\cdot, \cdot)$ to match the errors in the current dataset inferred from reads in the control lane [Weese et al., 2009, Qu et al., 2009]. When such information is not available, we can rely on information derived from other read data generated on the same platform. In the worst case, we can use the simple error model of Eq. (3.1), which only requires specification of the average error rate p_e .

Based on these choices, we tested our datasets using four types of sequencing error (misread) distributions: tIED, wIED, tUED, and wUED (defined below). Our simulation procedure introduced errors according to the misread probability matrices \mathbf{M} estimated from dataset SRX000429, so the *true* error distribution, tIED, was obtained by estimating $q_i(\cdot, \cdot)$ from the same dataset SRX000429. The estimation procedure is similar to the one used for estimating

Table 3.3 A comparison of minimum error rates. A Comparison of the minimum number of wrong predictions achieved by applying optimum thresholds to observed occurrences \mathbf{Y} , and our model with each of the error distributions tested. Bold numbers indicate that our model outperforms.

Data	Minimum (FP + FN) Value				
	Y	tIED	wIED	tUED	wUED
$D1$	2212	18	984	1020	4648
$D2$	6392	23	1300	3150	6729
$D3$	6809	19	1300	2696	3124
$D4$	216	10	236	80	719
$D5$	552	14	373	297	1346
$D6$	14236	13275	13441	13671	18793

\mathbf{M} (defined in the previous section), except each read is decomposed into $L - k + 1$ k mers and the count of each type of misread nucleotide at each k mer position is determined. (Note, the same nucleotide contributes counts in up to k distinct k mers.) Since, the estimated $q_i(\cdot, \cdot)$ represent fewer parameters than \mathbf{M} , $q_i(\cdot, \cdot)$ only approximates the true misread probability matrices \mathbf{M} , which themselves only approximate true read errors. The *wrong* Illumina error distribution, wIED, is the situation encountered when Illumina data are only available from a different experiment (and often different lab). To emulate this case, we derived a second set of error probabilities $q_i(\cdot, \cdot)$ from Illumina reads of *Acinetobacter sp. ADP1* (Short Read Archive acc. SRX001814, 17.7M reads of 36bp length). The error rates differ at k mer position $i = 11$ (Table 3.2) and others (not shown) in the *E. coli* and *A. sp. ADP1* short read datasets, demonstrating that wIED is indeed the *wrong* error distribution. Finally, in the absence of detailed error information, we can use the uniform error distribution with constant error probability p_e . When the average error rate $p_e = 0.006$ is estimated from dataset SRX000429, the error distribution is the *true* uniform error distribution (tUED). When the error rate is *overestimated* at $p_e = 0.02$, above the published rate of 0.01–0.015 [Shendure and Ji, 2008], it is the *wrong* uniform error distribution, wUED.

The same measures as in [Chin et al. \[2009\]](#) are used for evaluation, where a false positive (FP) denotes an error free k mer has been considered as erroneous and a false negative (FN) denotes an unidentified erroneous k mer.

Table 3.3 reports the minimum number of wrong predictions (WPs), FP+FN, achieved by applying optimum thresholds on observed \mathbf{Y} , used by existing methods, or by applying thresholds on the estimated number of attempts to read \mathbf{T} , used in our method. The results of our method are shown for the four types of error distributions in columns tIED, wIED, tUED, and wUED. Bolded entries indicate where lower minima were achieved with our method compared to the standard method. Given the *true* error distribution, our method committed over 95% *fewer* WPs for all datasets except $D6$, where our method still managed 7% fewer WPs. Interestingly, using the wrong Illumina error distribution (column wIED) achieved at least 33% fewer WPs in all repetitive genomes except $D4$, where our wIED method performed about on par with applying the threshold on \mathbf{Y} . The minimum WPs achieved by the true uniform error model tUED are two- to three-fold smaller than the corresponding values in column \mathbf{Y} . However, using elevated error rate $p_e = 0.02$ led to higher minimum WPs, except in dataset $D3$, the most highly repetitive simulated genome.

Even though we presented a method to choose the threshold value (see section 3.7), it is not possible for any method to guarantee the optimal threshold. Ideally, the error detection methods should be relatively insensitive to choice of threshold. To compare methods across many thresholds, we plot $\log(\text{FP} + \text{FN})$, with respect to a wide range of thresholds (Fig. 3.2). The plot in every case resembles a U-shape since many error k mers are missed (FN) when the threshold is too low and many correct k mers are declared errors (FP) when the threshold is too high. Our method achieved fewer WPs for datasets $D1, D2, D3$ and *Maize*

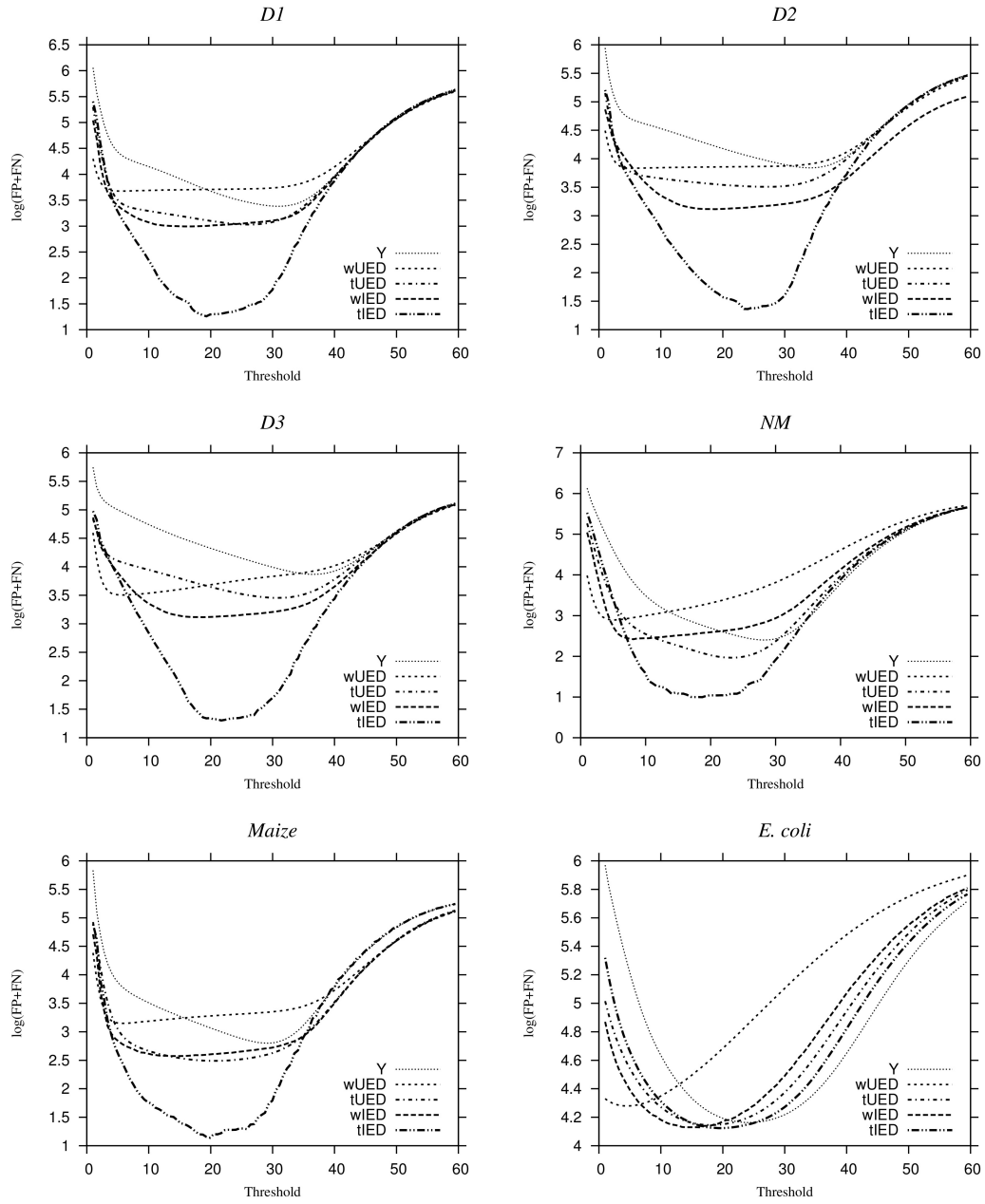


Figure 3.2 Plots of $\log(\text{FP} + \text{FN})$ vs. threshold for all datasets. In each plot, we compare the results by applying thresholds to \mathbf{Y} and to \mathbf{T} estimated by our model using the *tUED*, *wIED*, *tUED* and *wUED* error distributions.

with error distributions tIED, wIED, and tUED at *all* thresholds. The improvement in error detection increased with the degree of repetition, seen in simulations *D1* to *D3* and also in comparing *N. meningitidis* and *Maize*. Our method tended to flatten the bottom of the U-shape such that a wider range of thresholds often beat even the minimum error obtained under **Y** thresholding. In all datasets, our method often shifted the U-shape leftward, such that for very small thresholds, our WPs were far less than the **Y**-based methods, regardless of the error distribution used. As the threshold increases, WPs for all methods eventually converge to the same constant, where all *k*mers are considered erroneous. For moderately large thresholds, our method sometimes resulted in higher WPs, especially for the wrong error distribution wUED, and sometimes wIED, and genomes with a low degree of repeats.

REDEEM misclassified the fewest *k*mers when using the “true” error model, but even in our simulations, there was a mismatch between the simulated errors and the estimated “true” error model. The position-specific error probabilities used to compute *k*mer misread probabilities are not the true error probabilities that vary by position in the full length read. The difference is exacerbated as reads get longer relative to the *k*mer length. Since it would be possible to compute misread probabilities $p_e(x_m, x_l)$ using read-derived position probabilities, this mismatch between *k*mer and read errors can be eliminated with more sophisticated error models that account for the position of each *k*mer in the read. Since data to estimate the read error properties can be collected in parallel on a known, control genome, we contend that estimating the true error model is not an undue burden in practical applications [Weese et al., 2009, Qu et al., 2009]. Quality scores may also inform on errors [Wijaya et al., 2009] and could be incorporated in the REDEEM error model.

As discussed previously, only simulated data with different degrees of repeats can be utilized

Table 3.4 Error correction results.

Data	Method (<i>d</i>)	Sensitivity	Specificity	Gain	CPU Time(min)	Memory(GB)
<i>D1</i>	SHREC	81.2%	99.9%	80.3%	23.9	5.9
	Reptile	78.9%	99.9%	78.9%	0.6	0.19
	REDEEM	71.3%	99.9%	51.5%	114.1	2.5
<i>D2</i>	SHREC	54.0%	99.9%	52.7%	22.7	5.8
	Reptile	57.8%	99.9%	57.8%	0.5	0.16
	REDEEM	78.6%	99.9%	64.6%	72.7	1.6
<i>D3</i>	SHREC	29.3%	99.9%	26.7%	21.7	5.8
	Reptile	46.8%	99.9%	46.8%	0.5	0.13
	REDEEM	86.4%	99.9%	79.4%	31.2	0.63

to measure error correction results for repeat-rich genomic regions due to the fact that mapping short reads from such regions uniquely to the reference genome, and the assembly of genomes with high repeat content, remain open problems. We compare our correction results with SHREC [Schröder et al., 2009] and Reptile [Yang et al., 2010] using datasets *D1*, *D2* and *D3* with increasing degrees of repeat content. The results are shown in Table 3.4. To be self-contained, we reproduce the evaluation measures from Yang et al. [2010]: A *True Positive* (TP) is any erroneous base that is changed to the true base, a *False Positive* (FP) is any true base changed wrongly, a *True Negative* (TN) is any true base left unchanged, and a *False Negative* (FN) is any erroneous base left unchanged. $Sensitivity = TP / (TP + FN)$ and $Specificity = TN / (TN + FP)$. $Gain = (TP - FP) / (TP + FN)$ denotes the total percentage of erroneous bases removed from the dataset post-correction.

REDEEM is designed to specifically target error correction for repeat-rich genomes. While both SHREC and Reptile do not explicitly model the effect of repeats, the variable length suffixes captured by SHREC and different read decompositions explored by Reptile can provide richer and more precise information about errors. Currently, REDEEM does not utilize all such information. Therefore, in genomes with low repeat content, both SHREC and Reptile outperform. However, as the repetition within the genome increases, REDEEM significantly

outperforms both methods by accounting for the repetition in the k mer neighborhood. Further experiments show that error correction results are affected mainly by the percentage of the length of the genome spanned by repeats, rather than repeat types and lengths. This can serve as a yardstick in deciding when to use REDEEM over conventional error correction. It is also possible to combine the features of a conventional error correction method such as Reptile with the explicit modeling of repeats as done in REDEEM to produce an error-correction method that is superior both when sampling low repeat and highly-repetitive genomes.

All experiments were carried out on 3.16GHz Intel Xeon Processors; run time and memory usage of all three programs are shown in the last two columns in Table 4. As expected, the run time of REDEEM is longer due to the complexity of modeling repeats explicitly. The largest simulation, D6, took 120 minutes and 9 GB. No error detection/correction method except naïve thresholding on observed counts yet scales to practical next-generation applications, but REDEEM is at least comparable to existing, non-repeat-aware methods.

3.5 Discussion

We have presented a method for error detection that accommodates genomic repeats by considering the multiple potential sources of k mer counts, including both faithfully read matching strings and incorrectly read nonmatching strings in the genome. In the end, the method produces estimates of the expected number of attempts T_l to read k mer x_l , with or without errors. The k mers with small T_l identify strings x_l with low coverage in the dataset. Under the assumption of uniform coverage, very low T_l are likely to represent nonexistent k mers. The same logic is used to justify discarding k mers with small Y_l , but it is rarely recognized that Y_l is not directly proportional to genome occurrence α_l except in the absence of errors. Relying

on the overlapping erroneous k mers, we correct errors in the reads.

Choosing a threshold is a challenge for all error detection methods. In practice, the threshold can be trained on comparable data where errors are directly observable, perhaps by comparing with a known reference genome. Alternatively, analytical calculations can justify a threshold that minimizes some measure of incorrect predictions [Schröder et al., 2009]. A very simple calculation is available with some knowledge of genome length $|G|$, because then it is possible to estimate the FP rate, $P(Y_l < M \mid \alpha_l > 0)$ assuming uniform coverage. Our datasets have expected k -length string coverage $\frac{N(L-k+1)}{|G|-L+1}$ around 60X, which implies expected FP rates ranging from 1×10^{-14} at threshold $M = 10$ and 1×10^{-3} at threshold $M = 35$. Our own experiments, however, find that analytically calculated thresholds may actually introduce more errors than originally present in the dataset. Given these challenges, it is a positive feature of the proposed method that in almost all cases, a broader range of thresholds achieved nearly the same low wrong predictions (see Fig. 3.2). Under these conditions, success of the method becomes less dependent on the choice of threshold. In addition, the leftward shift of our wrong prediction curves tends to favor our method if thresholds are chosen conservatively.

Assemblers that incorporate error k mer detection tend to be conservative, reducing gaps in the assembled genome by holding FP low with small thresholds M [Butler et al., 2008, Jackson et al., 2010, Zerbino and Birney, 2008]. Although our method claimed lower overall wrong predictions in this range, we note that our FP error rates are higher than those achieved with \mathbf{Y} thresholding, barely so in the absence of repetition and correct error distribution, but notably so otherwise. Lower FN at the cost of higher FP reflects our ability to reduce errors by detecting k mers that do not exist in the genome, but have neighbors that are highly repetitive in the genome. Thus, identifying errors with our method reduces the complexity of assembly

by removing FN that cause tangles [Pevzner et al., 2001], but increases FP that cause contig breaks. The resulting trade-off may not be appropriate for all applications. FP k mers tend to be k mers that occur once in the genome in a region with low coverage. Our method could be improved by incorporating variable coverage effects or scanning for k mers possibly discarded due to low coverage.

Our estimated error distributions, tIEP, wIEP, tUEP, and wUEP, do not match the error distribution in the simulated reads and especially do not match the errors in the true reads of the *E. coli* dataset SRX000429. All error models are approximations, and the error distribution we can estimate is a combination of read errors, errors introduced by the mapping software, and errors in the assembly. Either as a consequence of this mismatch of model and reality, the longer genome, or both, Fig. 3.2 shows that prediction errors were substantially higher in the *E. coli* dataset for all methods. The error distributions of datasets from *E. coli* and *A. sp. ADP1* differed notably (Table 3.2), which may imply error rates vary substantially across organisms and sample preparations. For all these reasons, error estimation will continue to be a challenge in the future. Ideally, errors should be estimated along with T_l , but a less parameter-rich model is required, perhaps along the lines of the mixture models used to estimate genome length [Li and Waterman, 2003].

Choosing k mer length k and maximum Hamming distance d_{\max} are additional challenges with few easy answers. We chose k such that the average non-repetitive k mer has one genome occurrence. This choice relies on having some idea of genome repetitiveness. In general, choosing a large value of k results in low read occurrences Y_l and little information for inference of T_l . Yet choosing k too small populates k mer neighborhoods with counts from k mers occurring in largely incompatible contexts. In both cases, the noise of too little data or the noise of

irrelevant data interferes with the useful signal needed to detect possible error reads. Choice of k is connected to choice of d_{\max} . All results reported here are for $d_{\max} = 1$; results for $d_{\max} = 2$ changed little. However, as k increases, $d_{\max} = 1$ may not adequately capture the true neighbor context of k mers.

Our estimation method uses the EM algorithm to compute the expected number of k mer misreads Y_{ml} , which makes it possible to not only detect error k mers, as we have done, but also identify errors in valid k mers. Specifically, the computed Y_{ml} could be used to identify *valid* k mers that are over- or under-counted. For example, if Y_l is high because error count Y_{ml} is large due to some repetitive x_m , then the prevalence of k mer x_l in the genome is overestimated by Y_l . Indeed, T_l can be used to estimate genome length and repetition [Li and Waterman, 2003].

There have been some attempts to formally characterize repeats in genomes [Haubold and Wiehe, 2006], but generally, the term “repeat” is used loosely in the literature, with meaning varying by context. In this chapter, we consider k mer x_l a repeat when its genomic occurrence α_l is higher than what is expected in a random genomic sequence. Because genomes are not random, all genomes display some degree of repetition. Perhaps such cryptic repetition explains why we can achieve lower false prediction rates at optimal thresholds even on non-repetitive genomes, like *E. coli*. In fact, *E. coli* is somewhat repetitive according to the I_r measure of Haubold and Wiehe [2006].

3.6 Our Contributions

We develop a statistical model and a computational method for error detection and correction in the presence of genomic repeats. We propose a method to infer genomic frequencies

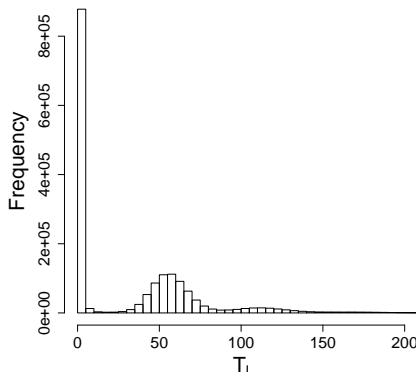


Figure 3.3 Histogram of estimated T_l for *E. coli* dataset.

of kmers from their observed frequencies by analyzing the misread relationships among observed kmers. We also propose a method to estimate the threshold useful for validating kmers whose estimated genomic frequency exceeds the threshold. We demonstrate that superior error detection is achieved using these methods. Furthermore, we break away from the common assumption of uniformly distributed errors within a read, and provide a framework to model position-dependent error occurrence frequencies common to many short read platforms. Lastly, we achieve better error correction in genomes with high repeat content.

The proposed method is made available through the software package REDEEM (Read Error Detection and Correction via Expectation Maximization) under GNU GPL3 license and Boost Software V1.0 license at “<http://aluru-sun.ece.iastate.edu/doku.php?id=redeem>”.

3.7 Detailed Threshold Inference

The true expected number of attempts to read k mer x_l , $\frac{\alpha_l n(L-k+1)}{|G|-k+1}$, are constant multiples of the discrete genome occurrences $\alpha_l \in \{0, 1, \dots\}$, where the coverage-related constant $\frac{n(L-k+1)}{|G|-k+1}$ is unknown. The estimated T_l vary from these true values because of sampling and estimation

error. A histogram of estimated T_l (see Fig. 3.3 for the *E. coli* dataset) thus reveals peaks corresponding to $\alpha_l = 0$, $\alpha_l = 1$, and $\alpha_l = 2$. The constant multiple is about 57, which can also be verified from Table 1. The k mers with T_l near 0 have no occurrences in the genome. One approach to model multi-modal distributions, such as that of Fig. 3.3, is to use a mixture model. Then, erroneous k mers would be those derived from the mixture component corresponding to the first mode. We propose mixture distribution

$$T_l \stackrel{\text{iid}}{\sim} \pi_0 f(\alpha, \beta) + \sum_{g=1}^{\mathcal{G}} \pi_g \phi(\mu_g, \sigma_g^2) + \frac{\pi_{\mathcal{G}+1}}{\max_l T_l}, \quad (3.2)$$

where the mixing probabilities $\pi_0, \dots, \pi_{\mathcal{G}+1}$ sum to 1. The first component $f(\alpha, \beta)$ of the mixture is a Gamma distribution, corresponding to the erroneous k mers. The second through $(\mathcal{G} + 1)$ th components are a series of normal distributions fitting the subsequent peaks for $\alpha_l = 1, 2, \dots, \mathcal{G}$. The last component is a uniform distribution over the observed range of T_l . We use the uniform distribution to account for the few k mers with large $\alpha_l > \mathcal{G}$. One particular parameterization of the normal components is justified as follows. Under the assumption of reads distributed uniformly throughout the genome, the true T_l are Poisson random variables, where the mean depends on the identity of k mer x_l and the error model. Rather than model the errors again, we hypothesize these means are Gamma random deviates. Then, T_l follows a Negative Binomial distribution [McCullagh and Nelder, 1989], with means $\mu_g = g\mu p/(1-p)$ and variances $\sigma_g^2 = g\mu p/(1-p)^2$ when $\alpha_l = g \in \{1, \dots, \mathcal{G}\}$. Finally, by the Central Limit Theorem, the Negative Binomial is well-approximated by the Normal distribution with matching means and variances when the coverage-related parameter $\frac{p\mu}{1-p}$ is large.

We can obtain the maximum likelihood estimate of the parameter vector $\theta = (\pi_0, \dots, \pi_{\mathcal{G}+1}, \alpha, \beta, \mu, p)$ using another EM algorithm. Let $Z_{lg}, l \in \{1, \dots, |R^k|\}, g \in \{0, \dots, \mathcal{G} + 1\}$ indicate

if k mer x_l is in group g . Then, the complete log likelihood for the proposed mixture model (Eq. 3.2) is

$$\sum_{l=1}^{|R^k|} \left\{ Z_{l0} \ln \left[\pi_0 \frac{\beta^\alpha T_l^{\alpha-1}}{\Gamma(\alpha)} \right] + \sum_{g=1}^{\mathcal{G}} Z_{lg} \ln \left[\frac{\pi_g}{\sqrt{2\pi}\sigma_g} \exp \left(-\frac{(T_l(1-p)-g\mu p)^2}{2g\mu p} \right) \right] + Z_{l,\mathcal{G}+1} \ln \left[\frac{1-\pi_0-\dots-\pi_{\mathcal{G}}}{\max_l T_l} \right] \right\}$$

Let $z_{lg} = E[Z_{lg} | T_l, \hat{\theta}]$ be the probability x_l belongs in group g given the current estimate $\hat{\theta}$ of all model parameters. For computational efficiency, at each iteration, we first compute

$$\begin{aligned} E[N_g] &= \sum_{l=1}^{|R^k|} z_{lg} & E[T | Z_g = 1] &= \sum_{l=1}^{|R^k|} T_l z_{lg} \\ E[T^2 | Z_g = 1] &= \sum_{l=1}^{|R^k|} T_l^2 z_{lg} & E[\ln T | Z_0 = 1] &= \sum_{l=1}^{|R^k|} z_{l0} \ln T_l, \end{aligned}$$

the first three for all $g = 0, \dots, \mathcal{G} + 1$. Here, N_g is the number of k mers in group g , T is the number of attempts to read a random k mer, and Z_g indicates if this k mer is in group g . Then, the update equations for the parameters are

$$\begin{aligned} \hat{\pi}_g &= \frac{E[N_g]}{|R^k|} && \text{for all } g \\ \ln \alpha - \Psi(\alpha) &= \ln E[T | Z_0 = 1] - \ln E[N_0] - \frac{E[\ln T | Z_0 = 1]}{E[N_0]} && \text{solve for } \hat{\alpha} \\ \hat{\beta} &= \frac{E[N_0] \hat{\alpha}}{E[T | Z_0 = 1]} \\ \hat{\mu} &= \frac{\sum_{g=1}^{\mathcal{G}} E[N_g] - \sqrt{(\sum_{g=1}^{\mathcal{G}} E[N_g])^2 + 4(1-\hat{p})^2 (\sum_{g=1}^{\mathcal{G}} g E[N_g]) (\sum_{g=1}^{\mathcal{G}} E[T^2 | Z_g = 1] / g)}}{-2\hat{p} \sum_{g=1}^{\mathcal{G}} g E[N_g]}, \end{aligned}$$

where \hat{p} is found as the root of

$$\begin{aligned} (1 - \hat{p})(1 + \hat{p}) \sum_{g=1}^{\mathcal{G}} E[T^2 | Z_g = 1] / g - 2\hat{\mu}\hat{p}^2 \sum_{g=1}^{\mathcal{G}} E[T | Z_g = 1] \\ - \hat{\mu}^2 \hat{p}^2 \sum_{g=1}^{\mathcal{G}} g E[N_g] - \frac{\hat{\mu}\hat{p}(1 + \hat{p})}{1 - \hat{p}} \sum_{g=1}^{\mathcal{G}} E[N_g], \end{aligned}$$

given $\hat{\mu}$ above. This and the second equation are implicit functions for $\hat{\alpha}$ and \hat{p} that can be solved using any one-dimensional root finder. To choose the best number of groups $\hat{\mathcal{G}}$, we

compute and minimize the BIC [Schwarz, 1978] over a range of plausible \mathcal{G} . Members of the gamma component that represent *kmers not* present in the genome are identified as those with $\text{argmax}_g P(Z_{lg} = 1 \mid T_l, \hat{\theta}, \hat{\mathcal{G}}) = 0$.

CHAPTER 4. MapReduce Framework for Read Clustering in Metagenomics

Metagenomics is the study of a population of organisms by fragmenting and sequencing their collective DNA [Gill et al., 2006, Poinar et al., 2006, Venter et al., 2004]. It is often applied to communities of microbial organisms in their native environments where species-wise separation is difficult, expensive, or downright impossible. Such studies are essential for identifying and discovering novel genes, studying ecosystems, and inferring the impact of microbial composition on host species. As with other areas of genomics, high-throughput next generation DNA sequencing [Ansorge, 2009, Perkel, 2009] is replacing Sanger sequencing due to its enormous advantages in cost, throughput, and scale of data generation. In the case of metagenomics, this comes with one distinct handicap – most bacterial genes are small enough to be fully contained in a Sanger read (up to a 1000 bp of DNA), making the task of gene identification easier. This is lost due to the short read lengths supported by next-generation sequencing.

Among the next-generation sequencing techniques, the 454 technology [Margulies et al., 2005] is particularly suitable for metagenomic analysis [Poinar et al., 2006, Gordon et al., 2005, Simon et al., 2009, Turnbaugh et al., 2009]. With the average length around 400 bp, 454 reads are about a third to half the length of Sanger reads, but still long enough for reliable gene identification. An important concern in surveying metagenomic samples is the clouding

out of low abundance species by highly abundant species. The significantly higher throughput of 454 technology (10-20 million reads per run) facilitates deep coverage sequencing, and helps uncover numerous new species with low abundance in an environmental sample, hence, beating Sanger sequencing in gene discovery. Most other next-generation sequencing techniques have even higher throughputs (for example Illumina Genome Analyzer can produce 300-400 million reads [[Lakdawalla and Steenhouse, 2008](#)]), but produce much shorter reads (25-150 bp), making sequence homology inference less reliable [[Liu et al., 2008](#)].

This work is motivated by the need to profile microbial organisms from human or mouse gut for obesity-association studies [[Gill et al., 2006](#), [Turnbaugh et al., 2009](#), [Verberkmoes et al., 2009](#)]. The composition of these communities is affected by diet, disease states and environmental factors. The goal is to quantify species abundance as per hierarchical taxonomic units, where each taxonomic unit is a group of organisms that belong to the same defined biological type such as phylum, genus, or species. The abundance of a taxonomic unit in the metagenomic sample is estimated by computing the ratio of the number of DNA reads belonging to the organisms of this unit over the total number of sequenced reads. In the current application, the reads are derived from the 16S ribosomal RNA (rRNA) pool of the sample. These serve as a good proxy for profiling abundance since they are conserved among organisms within a species while diverging across species.

Two approaches have been pursued in addressing this problem: The first relies on a database of known 16S rRNA sequences (*e.g.*, the Ribosome Database [[Cole et al., 2009](#)]). The quantification process is carried out by aligning every sequenced DNA read to the database sequences, and assigning it to the taxonomic unit based on taxonomic classification of the corresponding database sequences [[Liu et al., 2008](#), [DeSantis et al., 2006a,b](#), [Diaz et al., 2009](#), [Huson](#)

et al., 2007, McHardy et al., 2007, Meyer et al., 2008, Wang et al., 2007]. Such an approach is obviously limited to currently known organisms, which are far from comprehensive [Meyer et al., 2008, Blow, 2008]. In fact, advances in sequencing technology are expected to be used to further this knowledge and uncover many hitherto unknown species. Hence, the second approach advocated is to perform direct clustering or binning of the reads based on pairwise homologies [Li et al., 2008]. It is expected that carefully calibrated alignment thresholds will cluster sequences at different levels of the hierarchy of taxonomical classification.

Current methods for metagenomic clustering are deficient in both the scale of data they can handle and the quality of clustering. This is particularly an issue for terascale metagenomics projects [TM] – 400 Mbp to 600 Mbp data could be collected via the recent 454 Roche GS Titanium system within 10 hours [Margulies et al., 2005]. Among the existing methods, CD-HIT [Li et al., 2008] can handle relatively large datasets. However, it is intended to cluster protein sequences that are highly similar. Other methods, such as sketching techniques [Cameron et al., 2007] and parallel clustering or alignment algorithms [Kalyanaraman et al., 2003, 2006, Zola et al., 2007], were proposed to scale to large datasets. But none are applicable to metagenomic data clustering. The former can only cluster highly similar sequences. The single linkage clustering technique used in [Kalyanaraman et al., 2003, 2006] would cause different taxonomic units to be non-differentiable. In addition, in metagenomic data analysis, the clustering problem itself has not been defined precisely so far.

In this paper, we propose a formal model for the metagenomic clustering problem and present parallel algorithms for computing it. We adapted the sketching technique [Broder et al., 1997] to partially offset the $O(n^2)$ time complexity for identifying pairwise sequence homologies, where n is the number of sequences in the input. Multiple hierarchical taxo-

onomic clusterings are achieved through setting acceptable threshold values on the homologies. We formalize clustering as an iterative quasi-clique enumeration problem. We cast the presented algorithms in the map-reduce framework [Dean and Ghemawat, 2008] and develop a cloud-enabled software for metagenomic clustering named CLOSET (CLOud Open Sequence clusTering). The map-reduce paradigm has been drawing the attention of the computational biology community [McKenna et al., 2010, Schatz et al., 2010, Wall et al., 2010], particularly in the last year. However, majority of these applications are limited to simply distributing the data which is then handled by existing sequential software. We demonstrate that relatively complicated algorithms can utilize map-reduce framework. Although we demonstrate our clustering framework with an application to 16S rRNAs, it is a general purpose framework that would also be suitable for the upcoming ultra-long read technologies, for instance, single molecule real time sequencing [Eid et al., 2009].

The rest of this paper is organized as follows. In Section 4.1 we provide a brief introduction and formalize the clustering problem. In Section 4.2 we first outline our proposed approach, and then in Section 4.3 we give details of the two underlying algorithms. Section 4.4 provides information about how the algorithms are translated into a map-reduce implementation. Finally, Section 4.5 gives experimental results, and Section 4.6 concludes the paper.

4.1 Problem Formulation and Computational Modeling

Biologists group and categorize living organisms in the form of a taxonomic hierarchy. From a computer science perspective, this is a hierarchical tree where each node corresponds to a taxonomic rank, such as phylum, class, family, and genus. At each rank, the groups of organisms that belong to the same biological type is termed a *taxonomic unit* (TU). To quantify

a TU in an environmental sample amounts to calculating the ratio of the number of individuals belonging to this TU over the total number of organisms in the same sample. The problem is trivial if each individual in a metagenomic sample can be identified properly. However, this information is only indirectly available through reads taken from individual genomes of different organisms in the sample. In our case, the input is a set of reads from 16S rRNAs which we take as providing strong but imperfect signatures for identifying the underlying organisms.

Given a set of Roche 454 reads with average length around 400 bp, which are partial sequences derived from full length 16S rRNAs (~ 1600 bp) of microbial organisms in an environmental sample, the goal is to quantify the composition of taxonomic units at different taxonomic ranks. The number of reads belonging to the same taxonomic unit defines its prevalence in the sample. With majority of these reads coming from undocumented organisms, one cannot associate reads with organisms directly. Rather, we indirectly infer through alignment of reads coming from the same organism or from multiple organisms within the same taxonomic unit. This task is achieved via clustering. In general, reads are more similar at a lower taxonomic rank (*e.g.*, genus) than at a higher taxonomic rank (*e.g.*, family).

To tackle the above problem, we assume the availability of a pairwise similarity function such that two reads of the same taxonomic unit can be differentiated from those belonging to different taxonomic units at the same taxonomic rank. The function is not expected to be (indeed it cannot be) perfect but provide trustworthy differentiation in a good majority of the cases. Examples of such functions include sequence alignment, secondary structure alignment or their approximations. After the similarity relationships have been established, reads can be grouped together using different clustering algorithms [Xu and Wunsch, 2005].

Computationally, we need to address the following two tasks. Given a set of reads $R =$

$\{r_1, r_2, \dots, r_n\}$ as input:

1. Identify all pairs (r_i, r_j) such that $F(r_i, r_j) \geq t$, where F is the chosen similarity function and t is a threshold.
2. Cluster reads based on their established similarity association.

Current methods for metagenomic read clustering do not document a goal for clustering but rather leave it to be inferred through the algorithmic approach. As observed in [Estivill-Castro, 2002], diverse needs of clustering in relation to the application domains is the main reason behind the plethora of clustering algorithms developed in the literature. In case of metagenomics, single linkage clustering or hierarchical clustering techniques are widely used (*e.g.*, [DeSantis et al., 2006a, Li and Godzik, 2006]). The major flaw of these strategies lies in their inability to properly deal with inaccuracies and ambiguities in the similarity measure, prevalent in metagenomic data analysis. Two major problems arise due to the way current methods deal with ambiguities: when a read is highly similar to reads from multiple taxonomic units, the read is included in one of them leading to a wrong count, or even worse, the taxonomic units are merged into a single one (in case of single linkage clustering). Once a mistake is committed, it percolates upwards in the hierarchy of taxonomic ranks. In other words, current methods look for partitioning of the read set – meaningful only if read clustering can be made accurately. Moreover, the correct similarity score threshold that differentiates a taxonomic rank successfully from all others at the same rank is unknown.

To address these issues, we model the clustering problem as follows: we regard true clustering at a certain taxonomic rank to be a partitioning of the input reads, the ideal we seek. However, since the function F is unlikely to faithfully reflect evolution such that reads of the same taxonomic unit can be unambiguously differentiated from reads of other units, we allow

a read to concurrently occur in multiple clusters when applicable. The ambiguities in read assignments are likely to be alleviated as we lower the similarity threshold. For instance, let $F(r_i, r_j) = 90\%$, $F(r_i, r_k) = 90\%$, $F(r_j, r_k) = 84\%$, and $F(r_p, r_q) = 86\%$. If we choose $t = 85\%$ as a cutoff corresponding to the species rank and look for complete linkage clustering, three clusters can be formed: $\{r_i, r_j\}$, $\{r_i, r_k\}$ and $\{r_p, r_q\}$, where r_i is equally justified to be placed in two clusters. While at the family rank where t is lowered to 80% , a partition is achieved: $\{r_i, r_j, r_k\}$ and $\{r_p, r_q\}$. Ultimately, when the similarity threshold reduces to below a certain value, the input becomes a single cluster – a faithful reflection that all reads belong to the same domain. Due to false positives resulting from choosing the similarity function F , it is too stringent a requirement to expect complete linkage between every pair of reads in the same taxonomic unit. Furthermore, when the read similarity function is used, two reads from the same individual rRNA will not score highly on the similarity score if they sample different parts of the rRNA. To account for these, we propose to enforce a certain degree of partial linkages within a cluster that grows as a function of the cluster size.

We define a cluster to be consisting of a set of reads such that there exists a sufficient number of pairwise similarities among them. Formally, a cluster is a maximal set $T \subseteq R$ of reads such that $|\{(r, s) \in T \times T : r \neq s, F(r, s) \geq t\}| \geq \gamma \binom{|T|}{2}$. Note that this definition takes into account inaccuracies in determining taxonomic unit membership based on read similarities. The parameter γ can be dialed up or down to reflect the trust in this assessment, and can even be tuned as a function of the threshold t . The clustering problem is then defined as that of finding all maximal clusters T at a given threshold level t . Note that the clusters need not be a partition. In addition, the clusters could be computed for a decreasing sequence of threshold values. A biologist can then view the resulting clusters and identify the thresholds at which the

clusters appear to be classifying taxonomic units at a particular rank level. For instance, one could identify thresholds at which the resulting clustering best approximates a partition. This leads to more accurate classification because the data supports the inference that clustering can be done meaningfully at these threshold levels.

4.2 Proposed Algorithmic Approach

We now present our algorithmic approaches for computing pairwise read similarities and subsequent clustering. The typical practice in solving the first task is to compute all pairwise scores, which has $O(Cn^2)$ complexity, where C is the time taken by the function $F(r_i, r_j)$. This is straightforward to compute both sequentially and in parallel but infeasible for large n . Different clustering algorithms try to address this problem using different strategies. For example, CD-HIT [Li and Godzik, 2006] greedily searches for clusters among protein sequences, however, its worst-case time complexity remains $O(n^2)$. Note that an overwhelming majority of F function evaluations result in unconnected pairs of reads, even at higher taxonomic ranks (the highest taxonomic rank where all reads would be in one cluster is not considered). In this paper, we avoid all pairwise similarity computations and propose a sketching based technique to directly infer pairs whose estimated similarity scores exceed a given threshold. Our technique relies on adapting sketching techniques widely utilized for web document clustering [Broder et al., 1997, Charikar, 2002, Manku et al., 2007], where all vs. all comparisons are infeasible.

Our algorithm takes a decreasing sequence of similarity cutoffs $T = (t_1, t_2, \dots, t_m)$ as input. For each cutoff t_k , we perform clustering based on pairs of reads (r_i, r_j) such that $F(r_i, r_j) \geq t_k$. We formalize the above strategies using graph theory. Let $G^k = (V, E^k, W)$ be a set of undirected graphs ($1 \leq k \leq m$), where vertex $v_i \in V$ denotes read $r_i \in R$ for $1 \leq i \leq n$.

W captures the edge weights: $w_{ij} = F(r_i, r_j)$ if $F(r_i, r_j) \geq t_k$, and is 0 otherwise. In practice, we neither compute nor store zero weight edges. For a specific threshold t_k , $e_{ij}^k = (v_i, v_j) \in E^k$ if and only if $w_{ij} \geq t_k$, ($k \geq 1$). Note that $E^{k-1} \subseteq E^k$ ($1 < k \leq m$). We compute clustering on graphs G^1, G^2, \dots, G^m , in that order. We add edges incrementally in going from one graph to the next. In what follows, the superscript is dropped for convenience and each graph is simply referred to as G with the threshold becoming clear from the context. A cluster is a maximal quasi-clique in G : let $U \subseteq V$; the U -induced subgraph $G' = (U, E_U)$ is a γ -quasi-clique if $|E_U| \geq \gamma \binom{|U|}{2}$, for $0 < \gamma \leq 1$.

4.3 Algorithms for Metagenomic Sequence Clustering

In this section, we present our algorithms for 1) edge construction and validation, and 2) clustering via incremental maximal quasi-clique enumeration. The map-reduce realization of these algorithms will be described in detail in the next section. In what follows, we use the notation $\langle key, value \rangle$ to denote a key and value pair.

4.3.1 Phase I: Edge Construction and Validation

Our algorithm for edge construction is adapted from the sketching technique originally proposed for web based document clustering [Broder et al., 1997]. We briefly recall here the relevant key ideas from document clustering. Initially, each document D_i is converted to its corresponding tuple set S_i , where each tuple in S_i is a sequence of k consecutive words in D_i . Then, a universal hash function is utilized to map every tuple in S_i uniformly to the space of integers (usually 64-bit). After hashing, D_i has been converted to a set of integers, denoted by H_i . The Jaccard similarity coefficient between D_i and D_j , defined as $\frac{|H_i \cap H_j|}{|H_i \cup H_j|}$, has been shown [Broder et al., 2000] to be equivalent to the probability that the minimum values of H_i

and H_j are the same. Theoretically, to derive the Jaccard similarities among a set of highly similar documents, it is sufficient to use the above strategy to compare the extracted minimum values from each of them. These extracts are termed as sketches, and using the sketches to pair documents that share the same sketch avoids pairwise comparisons. Nonetheless, in practice, the clustering results are greatly influenced by the chosen hash function. The accuracy of using this technique degrades when used to cluster documents that are less similar [Henzinger, 2006]. At the same time, more sketches can be chosen (*e.g.*, via a modular function) to better represent each document under comparison.

For metagenomic read clustering, we are facing the challenge to cluster not only reads with high similarities (*e.g.*, 95%) but also reads with much lower similarities (*e.g.*, 60%) to be able to classify microbial organisms at different taxonomic ranks. In addition, reads are in DNA alphabet of size just 4, compared to the much larger alphabet set used in documents and a large number of reads may share a common substring. Our solution for adapting sketching techniques to metagenomic reads is given in Algorithm 3. Even though presented as a serial algorithm for convenience, it is designed such that the individual steps can be easily executed in parallel. Details of parallel execution using the map-reduce framework are deferred to the next section.

Each read in $R = \{r_1, r_2, \dots, r_n\}$ is initially converted to a set of integers comprising of the hash value of every constituent k -mer (substring of length k) (line 1). Instead of choosing the minimum value to represent a read r_i , we select a subset S_i of hash values that are l modulo M , where M is a preset constant (line 3). The similarity between reads r_i and r_j is computed as $|S_i \cap S_j|$ (derived in lines 4–14) divided by $\min(|S_i|, |S_j|)$. Our design of this similarity function is motivated by the need to capture containment relationships, and account for differences in

read lengths. Note that if read r_i is a substring of read r_j , then $S_i \subseteq S_j$, resulting in a perfect similarity score of 100% as desired.

- 1: For each read $r_i \in R$, hash every constituent k -mer to a 64-bit integer, resulting in a hash set $H_i = \{h_1, h_2, \dots\}$.
- 2: **for** $l = 0$ to $M - 1$ **do**
- 3: Generate the l^{th} sketch for each r_i :
 $S_i = \{h_j \mid h_j \in H_i \wedge (h_j \bmod M) = l\}$.
- 4: For each S_i , and each $s_j \in S_i$ generate $\langle s_j, rID_i \rangle$, where rID_i is the unique identifier assigned to r_i . Let SR denote the list of all pairs so generated.
- 5: Let n_j denote the number of elements in SR with key s_j . Let C_{max} be a user specified threshold.
- 6: **for** every unique key s_j of SR **do**
- 7: **if** $1 < n_j \leq C_{max}$ **then**
- 8: For all pairs of $\langle s_j, rID_a \rangle, \langle s_j, rID_b \rangle \in SR$, such that $rID_a \neq rID_b$, generate $\langle (rID_a, rID_b), 1 \rangle$.
- 9: **else**
- 10: Retain all $rIDs$ sharing the same key s_j as an entry in the list SR_{rem} .
- 11: **end if**
- 12: **end for**
- 13: Merge all entries generated in line 8 that share the same key by summing up the corresponding values.
- 14: Update every pair $\langle (rID_a, rID_b), count \rangle \in SR$ by incrementing count by one each time rID_a, rID_b concurrently belong to an entry in SR_{rem} .
- 15: For each $\langle (rID_i, rID_j), count \rangle \in SR$, derive similarity score $J_{ij} = \frac{count}{\min(|S_i|, |S_j|)}$. If $J_{ij} \geq C_{min}$, where C_{min} is the user specified similarity value, add the pair $\langle rID_i, rID_j \rangle$ to the candidate list L_l .
- 16: **end for**
- 17: Let $\mathbf{L} = \bigcup_{l=0}^{M-1} L_l$.
- 18: For every $\langle rID_i, rID_j \rangle \in \mathbf{L}$, if $F(r_i, r_j) \geq t$, add $\langle (rID_i, rID_j), F(r_i, r_j) \rangle$ to the final list of edges E .

Algorithm 3: Edge Construction and Validation

To avoid the $O(n^2)$ complexity in computing the similarity function for each pair, we let common hash values dictate which pairs to evaluate. Each common hash value shared between a pair of reads causes the generation of that pair with a frequency count of one. The frequency counts are later aggregated to reflect the number of common hash values. However, since DNA alphabet is small, some common k -mers may appear even between reads that do not

share significant similarity. In particular, short repeats or elements that frequently appear in multiple rRNAs can cause a significant throwback to the $O(n^2)$ complexity by creating many pairs with low frequency counts that will later be eliminated. To avoid this, Algorithm 3 postpones using high frequency k -mers to avoid generating exceedingly large number of read pairs (line 10). This practice is also supported by biological justification: substrings common to rRNAs from many organisms are not useful in differentiating among them. However, once we decide to explore similarity of two reads based on sharing of low frequency substrings, the high frequency ones have to be added back into the mixture to determine the corresponding similarity score (line 14).

The quality of results can be severely affected by sketching bias – the hash function does not guarantee to map k -mers uniformly into the integer space as assumed by the sketching technique [Broder et al., 2000]. To mitigate the effect of this, we apply M rounds of sketching and read pair (or equivalently, graph edge) generation. In round l ($0 \leq l < M$), the sketch is composed of hash values that are l modulo M . An edge survives as long as it is generated by at least one of these sketches. In a probabilistic sense, this exponentially decreases the chance that a read pair is completely missed. On the flip side, many edges are expected to be identified multiple times. The storage issues concerning this can be mitigated by combining the set of pairs for each successive sketch on the fly with the pairs seen so far, rather than combine them all at once as mathematically indicated in line 17.

Finally, the entire exercise of generating read pairs based on sketching can be seen as a filter to produce pairs worthy of further evaluation. Any user defined similarity function F can then be applied to assess the generated read pairs (equivalently, edges) directly. Some examples of such functions which biologists like to use are pairwise sequence alignment, and

secondary structure alignment. However, the sketch-based function we designed is accurate enough to be directly used in practice. If so, line 18 of the algorithm is unnecessary, and t can be used in place of C_{min} in line 15. Thus, we are able to provide a standalone solution for metagenomics clustering, while at the same time providing flexibility to the user to specify any arbitrary similarity function of their choice and immediately benefit from a highly efficient parallel implementation.

4.3.2 Phase II: Incremental Quasi-clique Enumeration

Exact maximal (quasi-) clique-enumeration has been extensively studied in the literature and there have been parallel algorithms designed for this problem using both message passing and shared memory paradigms [Du et al., 2006, Schmidt et al., 2009, Wu and Pei, 2009], as well as the MapReduce framework [Wu et al., 2009]. Due to the irregular graph structures arising in practice and the existence of an exponential number of maximal (quasi-) cliques in such graphs, existing methods are typically applied to random graphs or graphs with relatively smaller size than the ones introduced from the current application. Therefore, we designed an approximate maximal quasi-clique enumeration strategy, described as follows.

To cluster reads at different taxonomic ranks, we will use a series of similarity cutoffs $T = (t_1, t_2, \dots, t_m)$, sorted in the decreasing order. At each similarity level, edges are incrementally introduced into the graph and combined with the previously computed clusters. Using a series of similarity cutoffs would give biologists the flexibility to view different resulting clusters and identify the thresholds at which the clusters would be more meaningful towards a particular application.

Our method is presented in Algorithm 4. Each cluster c (a maximal γ -quasi-clique) is denoted by a pair: $\langle key, value \rangle$, where the key field, denoted by $c.key$, represents the set

```

1: Let  $T = (t_1, t_2, \dots, t_m)$  for  $t_1 > \dots > t_m$ .
2:  $i \leftarrow 1$ 
3: Let  $C$  denote the clustering results,  $C \leftarrow \emptyset$ .
4: while  $i \leq m$  do
5:   for every  $\langle (rID_i, rID_j), F(r_i, r_j) \rangle \in E$  do
6:     if  $F(r_i, r_j) \geq t_i$  then
7:        $C = C \cup \{ \{rID_i, rID_j\}, \{(rID_i, rID_j)\} \}$ 
8:     end if
9:   end for
10:  repeat
11:    Identify  $c_i, c_j \in C$  such that  $c_i.key \cap c_j.key \neq \emptyset$ ;
     $n_1 \leftarrow c_i.key \cup c_j.key$ ;  $n_2 \leftarrow c_i.value \cup c_j.value$ .
12:    if  $|n_2| / \binom{|n_1|}{2} \geq \gamma$  then
13:       $C = C \setminus \{c_i, c_j\}$ ;  $C = C \cup \{n_1, n_2\}$ .
14:    end if
15:  until no change in  $C$  is observed.
16:   $i \leftarrow i + 1$ 
17: end while

```

Algorithm 4: Maximal Quasi-clique Enumeration

of vertices in c , and the value field, denoted by $c.value$, represents the set of edges in this cluster. Initially, every cluster is a two-clique (line 6). Then, two clusters that share common vertices are joined together if a larger γ -quasi-clique can be formed (lines 10–15). Note that our algorithm offers no theoretical guarantee that it enumerates all maximal γ -quasi-cliques. In fact, the number of such cliques can be exponential in the worst case. Our algorithm is a heuristic to generate maximal quase-cliques appropriate to the problem at hand. We do not expect arbitrary input but given that the reads come from organisms which should fall into groups at various taxonomic ranks, the clique generation process is expected to discover these groupings. Note that the above notation for a cluster is for ease of presentation. In practice, the vertices need not be store explicitly but can be inferred from the edges.

4.4 Map-Reduce Implementation

We implemented our clustering algorithm using the map-reduce framework. This choice was dictated by several practical considerations: Typical metagenomic datasets consist of gigabytes of data and impose serious I/O and storage requirements. At the same time available map-reduce implementations, for instance Apache Hadoop, deliver highly efficient distributed file systems (*e.g.*, GFS, HDFS) that remove burden of managing I/O explicitly, and hide its low-level details from the programmer. Because map-reduce applications are typically deployed in large cluster environments, with a high probability of single node failure, map-reduce implementations provide strong fault-tolerance mechanisms. Finally, map-reduce applications can be easily ported to popular cloud environments, for instance Amazon EC2 (<http://aws.amazon.com/ec2/>), making them more accessible to users who are not HPC experts. The later argument seems to be especially noteworthy, taking into account increasing interest of biologists in cloud-enabled solutions.

The aforementioned advantages of map-reduce come at a price of constrained flexibility, since algorithms must be expressed as a series of map and reduce stages, through which the input data is streamed. While this pattern is sufficient for many embarrassingly parallel applications it becomes challenging, and sometimes infeasible, for more algorithmically involved problems. To express our clustering procedure using map-reduce, we designed a series of data transformations, where each transformation is a single map-reduce task, and collectively these tasks reflect steps described in Algorithms 3 and 4. Below we provide a brief description of each task specifying its input as well as map and reduce functions. For convenience we use notation from Algorithms 3 and 4. Additionally we denote by α a key assigned to a reducer, and we use $B = [\beta_1, \beta_2, \dots]$ to denote a list of values that share key α .

4.4.1 Edge Construction

We start by generating a list of candidate edges. In M iterations we process input reads generating their sketches, and producing potential edges that correspond to pairs of reads whose similarity is at least C_{min} . This is achieved by repeatedly calling *Task 1* and *Task 2*, which identify reads with a common k -mer, and do similarity counting, respectively. To pass information about k -mers that occurred more than C_{max} times, we use a temporary file. In *Task 1*, mapper generates pairs of hash value and read id, while reducer simply performs counting of reads that share given hash value. In *Task 2* on the other hand, mapper generates pairs of read identifiers, while reducer counts how many k -mers are shared by a given read pair and checks if it can be considered a candidate pair. Here, *Task 2* uses information stored previously in the temporary file:

Task 1: sketch selection

Input: $\langle rID, r \rangle$

Map: Generate sketch set S .

For each $s_i \in S$ emit $\langle s_i, rID \rangle$.

Reduce: If $|B| \leq C_{max}$ emit $\langle |B|, B \rangle$.

Otherwise, store B in a temporary file.

Task 2: edge generation

Input: $\langle n, [rID_1, rID_2, \dots, rID_n] \rangle$

Map: For each pair (rID_i, rID_j) , $i \neq j$,
emit $\langle (\min(rID_i, rID_j), \max(rID_i, rID_j)), 1 \rangle$.

Reduce: For $\alpha = (rID_i, rID_j)$

- Generate pair $\langle \alpha, m = |B| \rangle$.
- If rID_i and rID_j are both present in the temporary file generated in *Task 1*, increase m by one.
- Calculate J_{ij} (Algorithm 3, Line 15).
If $J_{ij} \geq C_{min}$ emit $\langle rID_i, rID_j \rangle$.

We expect that many edges will be generated several times, although the number of duplicates will depend on the hashing function used in *Task 1* and similarity of sequences. Therefore, in *Task 3* we remove redundant edges, and at the same time we replace each undirected edge with two corresponding directed edges. This step will facilitate data flow required to validate candidate edges (by calculating their exact similarity) in later stages. In this task, mapper emits a pair of vertices describing a given edge and reducer, for every source vertex, selects all unique target vertices:

Task 3: redundant edges removal

Input: $\langle rID_i, rID_j \rangle$

Map: Emit each input entry as it is.

Reduce: For every unique value $\beta_i \in B$

emit $\langle \alpha, \beta_i \rangle$ and $\langle \beta_i, \alpha \rangle$.

The next two tasks are probably the most interesting in the edge generation stage. In order to bring together information about edges that must be validated and the corresponding reads, mapper uses the original input data and information about edges delivered by *Task 3*. Reducer creates a tuple containing read sequence and a list of read ids with which given read will be compared. Then in *Task 5* this list can be used to generate another tuple, with pair of read identifiers as a key, and sequence read as a value. Because in *Task 3* every undirected edge is replaced with two directed counterparts, for every undirected edge, mapper in *Task 5* will generate exactly two tuples, each with one of the read sequences. Since read identifiers used to create a key are sorted, both tuples will be assigned to the same reducer, which will perform edge validation via computing the exact similarity score:

Task 4: data aggregation

Input: $\langle rID, r \rangle$ or $\langle rID_i, rID_j \rangle$

Map: Emit each input entry as it is.

Reduce: Identify $\beta^* \in B$ which is a read sequence.

Emit $\langle \alpha, [\beta^*, B \setminus \{\beta^*\}] \rangle$.

Task 5: edge validation

Input: $\langle rID, L = [r, rID_{j1}, rID_{j2}, \dots] \rangle$

Map: For each pair $(rID, rID_{ji}), rID_{ji} \in L$,

emit $\langle (\min(rID, rID_{ji}), \max(rID, rID_{ji})), r \rangle$.

Reduce: If $F(\beta_1, \beta_2) \geq C_{min}$ emit $\langle \alpha, F(\beta_1, \beta_2) \rangle$.

Although *Tasks 4* and *5* may appear to be compute intensive, they provide an efficient way to avoid storing all reads in the main memory, which is infeasible taking into account size of the metagenomic datasets. Moreover, because map-reduce tasks are streaming data following a sequence of consecutive hard drive accesses, we are avoiding expensive random access to the secondary storage, which would be necessary if we decided to fetch sequences from the hard drive each time validation is performed.

4.4.2 Incremental Quasi-clique Enumeration

The output from the first stage of our implementation is a list of edges together with their similarity. To implement the second stage of our algorithm, we first perform edge filtering depending on the thresholds $t_k \in T$. This is achieved by *Task 6*. Next we call iteratively *Tasks 7* and *8*. *Task 7* takes as input two types of data: the edge tuples that survived filtering in *Task 6*, and clusters generated in the previous iterations. Note that during the first iteration the set of input clusters is empty. Because *Task 7* may generate identical clusters, or clusters sharing the same vertex set but different edge set, these clusters are merged by keeping the vertex set intact, while taking the union of edges in *Task 8*. All three tasks are

called repeatedly for the changing value t_k , to obtain the clustering at different similarity levels:

Task 6: edge filtering

Input: $\langle (rID_i, rID_j), F(r_i, r_j) \rangle$

Map: If $t_k \leq F(r_i, r_j)$, emit $\langle rID_i, rID_j \rangle$.

Reduce: Emit $\langle \alpha, B \rangle$.

Task 7: quasi-clique merging

Input: $\langle rID, L_v = [rID_{j1}, rID_{j2}, \dots,] \rangle$ or $\langle h, L_e = [(rID_j, rID_k), (rID_l, rID_m), \dots] \rangle$

Map: For every $rID_{ji} \in L_v$, emit $\langle rID, (rID, rID_{ji}) \rangle$ and $\langle rID_{ji}, (rID, rID_{ji}) \rangle$. For every $rID_{ji} \in L_e$, emit $\langle rID_{ji}, L_e \rangle$.

Reduce: If and only if a pair (β_j, β_k) can be merged to form a γ -quasi-clique, emit $\langle h, \beta_j \cup \beta_k \rangle$, where h is a hash value of the concatenation of vertex labels in the clique.

Task 8: merge clusters sharing the same vertices

Input: $\langle h, L = [(rID_j, rID_k), (rID_l, rID_m), \dots] \rangle$

Map: Emit each input entry as it is.

Table 4.1 Characteristics of the data used for experiments.

	Small	Medium	Large
No. reads	312,296	1,741,911	5,656,392
Size [MB]	128.2	717.9	2,334.0
Read length [bp] (min./avg./max.)	192/371/813	167/373/894	167/375/894

Reduce: Identify all elements $\{\beta_1, \beta_2, \dots, \beta_l\}$ sharing the same vertex set, and emit $\langle h, \beta_1 \cup \beta_2 \cup \beta_l \rangle$, where h was the hash value defined in *Task 7*.

4.5 Experimental Results

4.5.1 Test Data and Experimental Environment

To assess efficiency of our clustering approach and its map-reduce implementation we performed a set of experiments for clustering 454 reads sampled from 709 mouse guts. The reads are subsequences from the full length 16S rRNA, and as such they represent a microbiome, *i.e.*, a microbial composition, found in the gut of each host mouse. The total data consists of 5,656,392 reads, and ability to cluster it at different taxonomic levels is important for understanding how the microbiome is shaped by the host genome, or for discovering unknown bacteria populations that may affect digestive tract of the host. For the purpose of experiments we randomly sampled (without replacement) the original dataset to obtain two smaller data collections. Properties of the resulting datasets are summarized in Table 4.1.

To implement map-reduce tasks described in the previous section, and to deploy the map-reduce cluster, we used the latest Apache Hadoop framework. Hadoop provides three different interfaces (Java, Pipes and Streaming), which vary in trade-off between efficiency and flexibility. Because our algorithm requires high data throughput (difficult to achieve in Hadoop Stream-

ing), and depends on computationally intensive subroutines (hard to implement efficiently in Java), we decided to rely on C++ and Hadoop Pipes combined with the `libhdfs` library to provide direct access to Hadoop distributed file system. To implement hashing function used in sketching stage we depend on the C++ Boost library (<http://www.boost.org/>), which provides efficient hashing of different data types into 64-bit space. To orchestrate map-reduce tasks we implemented additional scripts that keep track of iteration progress, and manage intermediate data whenever required. Finally, we created a set of small tools to convert input and output data between the standard biological data file formats and our internal representation. The complete software package constitutes a cloud-enabled framework, which we named CLOSET (CLOUD Open SequenceE clusTering).

We deployed CLOSET on a 32 node Hadoop cluster with a total of 256 GB main memory, and 6 TB secondary storage under the control of HDFS. The cluster provides 256 AMD 2.2 GHz cores and uses Gigabit Ethernet for interconnect. We used a typical Hadoop configuration with one node serving as a master tracking jobs and maintaining the HDFS metadata, and remaining nodes acting as workers executing computations and storing data blocks. We set HDFS replication factor to 2, and used 64 MB block size.

4.5.2 Experiments

We subjected our three datasets (Table 4.1) to analysis using CLOSET. We set the parameter M such that the expected fraction of hash values used to obtain sketch size is 0.02. Note that different choices of M affect how accurately we assess similarity between reads, and they allow us to control trade-off between accuracy of the edge generation stage and the run time. By choosing $M = 1$ we will obtain exact Jaccard similarity coefficient, which will lead to all pairs comparison in the latter stages of our algorithm. On the other hand, it is possible to

computationally choose M such that cardinality of resulting sketches for every read will be at most 1, which will eliminate pairwise comparison completely. We found that different choices of M have minor effect on quality of clustering; however, they influence overall execution time. Consequently, taking into account length of input reads we set M such that expected number of sketches per read is 5 to 16. We further constrain the number of iterations in the sketching stage to $l = 3$, which we find sufficient to accurately capture potential edges.

We set $k = 15$ to generate a reliable k -spectrum for our data, *i.e.*, the one which consists of k -mers that have low probability of occurring randomly in a given read. Note that to obtain such spectrum it is sufficient to set k such that $4^k \ll d$, where 4 is the size of the DNA alphabet and $d = 1,600$ is the approximate length of 16S rRNA sequences from which our data has been derived. Hence, our choice of k is stringent.

To select candidate edges we set C_{min} to 60%. This conservative choice can be considered a default value one would use when trying to capture clusters at the relatively high taxonomic rank in a *de-novo* experiment. In order to form the initial quasi-cliques, we set $\gamma \geq 2/3$, which can be tuned up or down as needed in later iterations.

Using the above parameters for each input dataset we executed the first stage of our algorithm and then a single iteration of the second stage for varying initial similarity threshold t_1 . We measured time taken by CLOSET, and quantity of data it generated including intermediate stages. The obtained results are summarized in Tables 4.2 and 4.3.

We start analysis of obtained results by first looking at the quantity of the data generated by our clustering algorithm (Table 4.2). Although these quantities strongly depend on the input, they provide several interesting insights. As expected the sketching algorithm provides tremendous saving in terms of execution: to obtain the set of confirmed edges, *i.e.*, edges with

Table 4.2 Quantity of the data generated in different stages when analyzing the three input datasets.

	Small	Medium	Large
Predicted edges	182,465,582	311,341,492	443,709,163
Unique edges	120,635,707	240,662,224	381,661,077
Confirmed edges	53,840,766	139,044,928	206,753,539
	$t_1 = 95\%$		
Clusters processed	1,907,730	11,381,018	30,272,606
Resulting clusters	243,735	103,497	1,206,096
	$t_1 = 92\%$		
Clusters processed	4,865,107	31,953,877	61,524,815
Resulting clusters	249,830	973,298	2,235,523
	$t_1 = 90\%$		
Clusters processed	21,028,890	136,133,245	238,177,413
Resulting clusters	351,823	1,640,042	3,343,883

similarity score above C_{min} , only a tiny fraction (between 2.38×10^{-5} and 0.002) of all vs. all comparisons is required. Note that the sketching technique may not generate some valid edges. However, taking into account that the ratio between the number of unique edges that were subjected to validation and the number of confirmed edges varied between 1.7 and 2.2 for all three datasets, and the number of additional edges introduced by increasing l was negligible, we believe that the number of false negative predictions is not significant to drastically affect clustering outcome in the later stages.

Another observation concerns change in the number of processed and generated clusters depending on the size of the input dataset and the similarity threshold t_1 . From Table 4.2, decreasing similarity threshold, which means including more edges in the clustering stage, leads to increased number of processed and output clusters. Here, by processed clusters we mean all clusters that have been generated and merged to obtain the final output. This is consistent with what one expects to see when enumerating maximal quasi-cliques in a random graph; however, it does not have to hold for all types of graphs. Nevertheless, since the input set of edges reflects

Table 4.3 Run time in seconds for different stages of the CLOSET algorithm executed on our 32 node Hadoop cluster.

	Small	Medium	Large
Sketching	771	2,116	4,220
Validation	549	1,138	1,639
	$t_1 = 95\%$		
Filtering	45	45	63
Clustering	633	696	1,072
	$t_1 = 92\%$		
Filtering	47	47	80
Clustering	933	1,442	1,498
	$t_1 = 90\%$		
Filtering	48	49	82
Clustering	1,772	2,690	3,733

a biological data (which often follows a scale-free node degree distribution [Barabasi and Albert, 1999]) it is justified to expect that by introducing more edges (lowering the similarity threshold) we will obtain more dense quasi-cliques, with small overlaps. Indeed, the ratio between the number of input edges and the number of quasi cliques generated in the small dataset changes from 0.01 for the threshold of 95% to 0.06 for the threshold of 90%. We observe the same trend in the other two datasets.

We now take a look at the run-times obtained when executing CLOSET on our 32 node cluster (Table 4.3). Analysis of map-reduce task efficiency is in general a challenging task. Performance of any map-reduce job is strongly affected by how well different (often very obscure) parameters of the underlying map-reduce library are configured, and it becomes routine to refer to the process of map-reduce job configuration as a “black art” [Sharma]. To be as fair as possible, we set all typically configured Hadoop parameters (*e.g.*, the number of reducers spawned, memory allocation to Java daemons, etc.) as we would do when running CLOSET without any prior knowledge about properties of the input data (except the data size). Overall we tried to balance memory and CPU pressure, avoiding excessive CPU over-subscription. To

make results comparable, we set all parameters with respect to the largest dataset, and we used them for analyzing other two datasets as well.

Table 4.3 shows that increasing size of the input data we only slightly increase overall execution time. This suggests that our implementation maintains very good scalability. Longer execution times observed for the small dataset can be explained by the fact that the data is not able even to partially saturate capacity of the cluster. For instance, only 2 mappers are started to handle *Task 1*. Moreover, because of much smaller computational requirements, I/O overhead, especially in the data-copying stage of map-reduce, becomes the dominating factor. As expected the execution time increases when more edges are introduced into the clustering phase.

Assessment of the Resulting Clusters (Methodology)

To validate clusters produced by any *de novo* clustering algorithm, extensive wet-lab experiments are required. Yet, currently no standard procedure exists. Therefore, validation based on datasets curated by biological experts, where the taxonomic rank of each read is known, is a method of choice. Given such data, one can infer clusters, termed *canonical clusters*, corresponding to each taxonomic rank, and compare them with the clusters derived from the clustering algorithm. In this section, we describe approach to analyze the consistency between the canonical clusters and the clusters produced by CLOSET, and to select, for a particular taxonomic rank, the similarity threshold to best separate sequences belonging to different taxonomic units.

We first introduce Adjusted Rand Index (ARI) [Hubert and Arabie, 1985], a statistical measure of consistency between two hard clusterings of a given set of elements. Then, we explain how this measure can be used in the current application.

Table 4.4 Contingency Table for calculating ARI

	V_1	V_2	\cdots	V_c	Sums
U_1	c_{11}	c_{12}	\cdots	c_{1c}	a_1
U_2	c_{21}	c_{22}	\cdots	c_{2c}	a_2
\cdots	\cdots	\cdots	\cdots	\cdots	\cdots
U_r	c_{r1}	c_{r2}	\cdots	c_{rc}	a_c
Sums	b_1	b_2	\cdots	b_c	

Given a set of elements \mathbf{E} ($|\mathbf{E}| = n$), and its two partitions: $\mathbf{U} = \{U_1, U_2, \dots, U_r\}$ and $\mathbf{V} = \{V_1, V_2, \dots, V_c\}$, where $U_1 \cup U_2 \cup \dots \cup U_r = V_1 \cup V_2 \cup \dots \cup V_c = \mathbf{E}$, $U_i \cap U_j = \emptyset$ for $1 \leq i < j \leq r$, and $V_i \cap V_j = \emptyset$ for $1 \leq i < j \leq c$. We define a contingency table as in Table 4.4. We can then calculate ARI as:

$$ARI = \frac{\sum_{ij} \binom{c_{ij}}{2} - \sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2} / \binom{n}{2}}{\frac{1}{2}(\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}) - \sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2} / \binom{n}{2}}$$

ARI has been used in different classification and clustering studies [Kelley and Salzberg, 2010, Santos and Embrechts, 2009, Yeung and Ruzzo, 2001], serving as a good criterion to be maximized in order to select a good set of features to be used in classification or clustering process. For example, given a set of features that could be used for clustering a set of elements, a feature set that leads to the largest ARI value can be considered to be the most meaningful. In the current application, we adapt the above idea and use the ARI criterion as a way to select CLOSET parameter values. The ARI value between the canonical clusters at a certain taxonomic rank and the resulting clusters is calculated for each selected set of parameters. The parameter value set that leads to the largest ARI value is considered to have the best discrimination power at the corresponding taxonomic rank. There are mainly three parameters to be tuned for CLOSET: the k value used in the sketching stage, the similarity threshold t that determines the minimum similarity value between a pair of sequences at a certain taxonomic

rank, and the γ value that defines the density of a quasi-clique. Empirically, we can identify a reasonable range for each of the individual parameters. Then, we can use any grid search method to identify optimal values for all three parameters.

According to the ARI formula, only hard clusterings can be compared. Nonetheless, in our approach, overlapping clusters are allowed in order to tolerate the imperfect similarity measure, sequences with unequal length, and other problems due to sequencing. Therefore, a method to convert the resulting overlapping clusters to a partition is necessary. Currently, no such method is readily available and this problem is left open.

4.6 Conclusions

Clustering of large-scale environmental samples is considered a difficult and challenging problem in metagenomics. Our work is motivated by the inability of existing methods and software to scale to the large datasets obtained by our biology collaborators as a part of routine research. In addition, there is lack of formal computational modeling and elegant algorithmic solutions. This paper represents our effort to satisfactorily address all the above problems and issues. We presented a rigorous framework for metagenomics clustering based on sketching and iterative quasi-clique enumeration. Our framework has the following advantages: It can accommodate arbitrary user-defined similarity functions. By performing clustering at many threshold levels, it can guide the biologist in tuning clustering to better fit different taxonomic ranks. Our software is implemented on widely available Hadoop platform with map-reduce framework for easy deployability. It can easily handle some of the largest data collections that are currently being generated by biologists for metagenomics studies. While the presented algorithms are motivated by this specific problem in metagenomics, we presented

two novel techniques that have much broader applicability. One is the development of sketching techniques for the realm of genomics. The other is a parallel algorithm for quasi clique enumeration.

CHAPTER 5. Summary and Future Directions

In this work, we have described two methods to improve the state-of-the-art in error correction. The first method, available as *Reptile*, relies on four major ideas: 1) context information of adjacent k mers to reduce ambiguities, 2) a space-efficient method for retrieving the d -neighborhood of any given k mer, 3) a flexible read decomposition strategy to correct sparse and clustered errors, and 4) a more meaningful metric for validating error correction results. Together, these ideas lead to a much faster, a more memory efficient, and a more accurate solution when applied to data from Illumina GAIIx system, a major next generation sequencing platform. However, there are two major remaining issues, *i.e.*, to identify a better way to handle genomic repeats and to devise a more fault-tolerant method to choose the threshold, comparing to which a substring is determined to be error free or not. To resolve these, we propose a second method *Redeem* to first infer the genomic occurrence of each k mer, and then to use a mixture parameter model to model the distribution of genomic occurrences of all observed k mers. The threshold is chosen based on this model and is compared against the genomic occurrence of each k mer. These strategies lead to a better prediction of valid versus invalid k mers in repeat-rich genomes and an automated selection of the threshold. Overall, both methods can be applied and extend to any next-generation platforms which dominantly generate substitution errors.

However, several challenges remain, which we briefly described below.

- To distinguish errors from polymorphisms, *e.g.*, SNPs. Reptile can accommodate SNP prediction by modifying the tile correction stage (algorithm 3), where ambiguities may indicate polymorphisms. So far, this is controlled by a user specified parameter, whereas, statistical modeling may provide a more reliable result. In addition, we are currently dealing with haplotype genomes. Given metagenomic data or sequences from population samples, a better modeling may be needed for differentiating natural variations from sequencing errors.
- To handle the growing read length of upcoming high-throughput sequencers. Currently in Reptile we define tiles as concatenations of two k mers. it might prove useful to extend the tile definition to more than two k mers in order to address error correction in much longer reads. However, this may add additional computational cost.
- A better memory management is required when we try to model genomic repeats in order to scale to large genomes. So far in Redeem, the complete Hamming graph needs to be stored in main memory because of the global optimization strategy used in the EM algorithm. Instead, a more localized EM algorithm and a distributed Hamming graph can be used to alleviate the large memory usage.
- To identify and correct the insertion and deletion errors, edit distance should be used to replace Hamming distance. Currently, the use of Hamming distance metric can be justified since major next-generation platforms such as Illumina dominantly make substitution errors. Upcoming new platforms such as those produced by Ion Torrent or PacBio may have similar error rate in insertions/deletions as in substitutions. Addressing all three types of errors may become equally important.

- To reduce the run time and memory usage, distributed parallelization strategy needs to be developed in addition to the commonly used multi-threaded implementation suitable for multicores.

Other than the aforementioned issues, there are several new directions to investigate: better utilization of quality scores, combining the use of contextual information used in Reptile and the automated calculation of threshold used in Redeem to devise a more generalized error correction strategy that is applicable to any dataset regardless of the prevalence of genomic repeats. In addition, in certain applications such as *de novo* genome assembly, it would also be interesting to see the association between the assembly results and the ratio of $\frac{TP}{FP}$, which indicates the tradeoff between errors corrected and errors introduced.

In this thesis, we also present an important application in next-generation sequencing – read clustering method for microbiota analysis. This is a fundamental task that will be carried out as a routine procedure in other applications such as association studies. We have considered this problem in its full biological complexity and integrated knowledges from different domains: sketching technique used in clustering web documents, approximate quasi-clique clustering strategy used in numerous graph theory applications, as well as MapReduce method implemented in a distributed computational framework. This problem is still in its infant stage, and some improvements can be made with respect to our current approach:

- To associate computational results with biological relevance.
- Use our framework as a backbone, plug in/out or replace different components in order to better adapt to a specific application, such as utilizing different clustering methods, hashing techniques, and alignment methods.

- To further improve scalability to ultra-large datasets in order to accomodate the foreseeable needs for large scale metagenomic data analysis.

Bibliography

Computational methods and terabase metagenomics. <http://www.icis.anl.gov/programs/summer-1>.

I.P. Adams, R.H. Glover, W.A. Monger, and R. Mumford *et al.* Next-generation sequencing and metagenomic analysis: a universal diagnostic tool in plant virology. *Mol Plant Pathol*, 10(4):537–545, Jul 2009. doi: 10.1111/j.1364-3703.2009.00545.x. URL <http://dx.doi.org/10.1111/j.1364-3703.2009.00545.x>.

W. J. Ansorge. Next-generation DNA sequencing techniques. *N Biotechnol*, 25(4):195–203, 2009.

KF. Au, H. Jiang, L. Lin, and Y. Xing *et al.* Detection of splice junctions from paired-end rna-seq data by splicemap. *Nucleic Acids Res*, 38(14):4570–4578, Aug 2010. doi: 10.1093/nar/gkq211. URL <http://dx.doi.org/10.1093/nar/gkq211>.

A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

S. Batzoglou. Arachne: A whole-genome shotgun assembler. *Genome Res.*, 12(1):177–189, 2002.

T Beissbarth, L Hyde, G K Smyth, and C Job *et al.* Statistical modeling of sequencing errors in SAGE libraries. *Bioinformatics*, 20 Suppl 1:i31–i39, 2004.

- D. Bentley, S. Balasubramanian, H. Swerdlow, and G. Smith *et al.* Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456(7218):53–59, Nov 2008. doi: 10.1038/nature07517. URL <http://dx.doi.org/10.1038/nature07517>.
- N. Blow. Metagenomics: Exploring unseen communities. *Nature*, 453(7195):687–690, 2008.
- A. Broder *et al.* Min-wise independent permutations. *J. Comput. System Sci.*, 60(3):630 – 659, 2000.
- A.Z. Broder, S.C. Glassman, M.S. Manasse, and G. Zweig. Syntactic clustering of the web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166, 1997. ISSN 0169-7552.
- J. Butler, I. MacCallum, M. Kleber, and I.A. Shlyakhter *et al.* Allpaths: de novo assembly of whole-genome shotgun microreads. *Genome Research*, 18:810–820, 2008.
- M. Cameron, Y. Bernstein, and H. E Williams. Clustered sequence representation for fast homology search. *J Comput Biol*, 14(5):594–614, Jun 2007.
- M. Chaisson, P. Pevzner, and H. Tang. Fragment assembly with short reads. *Bioinformatics*, 20(13):2067–2074, 2004.
- MJ. Chaisson and PA. Pevzner. Short read fragment assembly of bacterial genomes. *Genome Research*, 18:324–330, 2008.
- MJ. Chaisson, D. Brinza, and PA. Pevzner. De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Res*, 19(2):336–346, Feb 2009. doi: 10.1101/gr.079053.108. URL <http://dx.doi.org/10.1101/gr.079053.108>.
- C.K. Chan, A.L. Hsu, S.K. Halgamuge, and S. Tang. Binning sequences using very sparse labels

- within a metagenome. *BMC Bioinformatics*, 9:215, 2008. doi: 10.1186/1471-2105-9-215. URL <http://dx.doi.org/10.1186/1471-2105-9-215>.
- M.S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC '02: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 380–388, New York, NY, USA, 2002. ACM. ISBN 1-58113-495-9. doi: <http://doi.acm.org/10.1145/509907.509965>.
- F. Y L Chin, H. C M Leung, W. Li, and S. Yiu. Finding optimal threshold for correction error reads in dna assembling. *BMC Bioinformatics*, 10 Suppl 1:S15, 2009.
- NL. Clement, Q. Snell, MJ. Clement, and PC. Hollenhorst *et al.* The gnumap algorithm: unbiased probabilistic mapping of oligonucleotides from next-generation sequencing. *Bioinformatics*, 26(1):38–45, Jan 2010. doi: 10.1093/bioinformatics/btp614. URL <http://dx.doi.org/10.1093/bioinformatics/btp614>.
- J. R. Cole, Q. Wang, E. Cardenas, and J. Fish *et al.* The ribosomal database project: improved alignments and new tools for rRNA analysis. *Nucleic Acids Res*, 37(Database issue):D141–D145, 2009.
- J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
- A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J R STAT SOC, Series B*, 39(1):1–38, 1977.
- T. Z. DeSantis, P. Hugenholtz, K. Keller, and E. L. Brodie *et al.* NAST: a multiple sequence alignment server for comparative analysis of 16S rRNA genes. *Nucleic Acids Res*, 34(Web Server issue):W394–W399, 2006a.

- T. Z. DeSantis, P. Hugenholtz, N. Larsen, and M. Rojas *et al.* Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Appl Environ Microbiol*, 72(7):5069–5072, 2006b.
- N.N. Diaz, L. Krause, A. Goesmann, and K. Niehaus *et al.* TACOA: taxonomic classification of environmental genomic fragments using a kernelized nearest neighbor approach. *BMC Bioinformatics*, 10:56, 2009. doi: 10.1186/1471-2105-10-56. URL <http://dx.doi.org/10.1186/1471-2105-10-56>.
- JC. Dohm, C. Lottaz, T. Borodina, and H. Himmelbauer. Substantial biases in ultra-short read data sets from high-throughput DNA sequencing. *Nucleic Acids Res*, 36(16):e105, 2008.
- R. Drmanac, A. Sparks, M. Callow, and A. Halpern *et al.* Human genome sequencing using unchained base reads on self-assembling dna nanoarrays. *Science*, 327(5961):78–81, Jan 2010. doi: 10.1126/science.1181498. URL <http://dx.doi.org/10.1126/science.1181498>.
- N. Du, B. Wu, L. Xu, and B. Wang *et al.* A parallel algorithm for enumerating all maximal cliques in complex network. In *Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on*, pages 320 –324, 2006.
- R. Edwards, B. Rodriguez-Brito, L. Wegley, and M. Haynes *et al.* Using pyrosequencing to shed light on deep mine microbial ecology. *BMC Genomics*, 7:57, 2006.
- J. Eid, A. Fehr, J. Gray, and K. Luong *et al.* Real-time DNA sequencing from single polymerase molecules. *Science*, 323(5910):133–138, 2009.
- V. Estivill-Castro. Why so many clustering algorithms: a position paper. *ACM SIGKDD Explor. Newsl.*, 4(1):65–75, 2002.

- M.C. Frith, R. Wan, and P. Horton. Incorporating sequence quality data into alignment improves dna read mapping. *Nucleic Acids Res*, 38(7):e100, Apr 2010. doi: 10.1093/nar/gkq010. URL <http://dx.doi.org/10.1093/nar/gkq010>.
- P. Gajer, M. Schatz, and SL. Salzberg. Automated correction of genome sequence errors. *Nucleic Acids Res*, 32(2):562–569, 2004.
- S. Gill et al. Metagenomic analysis of the human distal gut microbiome. *Science*, 312(5778):1355–1359, 2006.
- J. Gordon et al. Extending our view of self: the human gut microbiome initiative (HGMI). <http://www.genome.gov/10002154>, 2005.
- M. Griffith, O. Griffith, J. Mwenifumbo, and R. Goya *et al.* Alternative expression analysis by rna sequencing. *Nat Methods*, 7(10):843–847, Oct 2010. doi: 10.1038/nmeth.1503. URL <http://dx.doi.org/10.1038/nmeth.1503>.
- B. Haubold and T. Wiehe. How repetitive are genomes? *BMC Bioinformatics*, 7:541, 2006.
- P. Havlak, R. Chen, K. J. Durbin, and A. Egan *et al.* The atlas genome assembly system. *Genome Res*, 14(4):721–732, 2004.
- M. Henzinger. Finding near-duplicate web pages: A large-scale evaluation of algorithms. In *Proc. of ACM SIGIR*, pages 284–291, 2006.
- X. Huang et al. Pcap: A whole-genome assembly program. *Genome Res.*, 13(9):2164–2170, 2003.
- Y. Huang, B. Niu, Y. Gao, L. Fu, and W. Li. CD-HIT Suite: a web server for clustering and comparing biological sequences. *Bioinformatics*, 26(5):680–682, 2010.

- L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.
- D.H. Huson, K. Reinert, and E. Myers. The greedy path-merging algorithm for contig scaffolding. *Journal of the ACM*, 49(5):603–615, 2002.
- D.H. Huson, A.F. Auch, J. Qi, and S.C. Schuster. MEGAN analysis of metagenomic data. *Genome Res*, 17(3):377–386, Mar 2007.
- D.H. Huson, D.C. Richter, S. Mitra, and A.F. Auch *et al.* Methods for comparative metagenomics. *BMC Bioinformatics*, 10 Suppl 1:S12, 2009. doi: 10.1186/1471-2105-10-S1-S12. URL <http://dx.doi.org/10.1186/1471-2105-10-S1-S12>.
- R. M. Idury and M. S. Waterman. A new algorithm for dna sequence assembly. *J Comput Biol*, 2(2):291–306, 1995.
- L. Ilie, F. Fazayeli, and S. Ilie. Hitec: accurate error correction in high-throughput sequencing data. *Bioinformatics*, 27(3):295–302, Feb 2011. doi: 10.1093/bioinformatics/btq653. URL <http://dx.doi.org/10.1093/bioinformatics/btq653>.
- F. V. Jacinto, E. Ballestar, and M. Esteller. Methyl-dna immunoprecipitation (medip): hunting down the dna methylome. *Biotechniques*, 44(1):35, 37, 39 passim, 2008.
- B.G. Jackson, M. Regennitter, X. Yang, and P.S. Schanble *et al.* Parallel de novo assembly of large genomes from high-throughput short reads. In *24th IEEE International Parallel & Distributed Processing Symposium, Accepted*, 2010.
- Michal Janitz, editor. *Next-generation Genome Sequencing*. Wiley-VCH, 1 edition, 2008.
- D. Johnson, A. Mortazavi, R. Myers, and B. Wold. Genome-wide mapping of in vivo protein-dna interactions. *Science*, 316(5830):1497–1502, 2007.

- A. Kalyanaraman, S. Aluru, V. Brendel, and S. Kothari. Space and time efficient parallel algorithms and software for EST clustering. *IEEE Trans. Para*, 14(12):1209–1221, 2003.
- A. Kalyanaraman, S.J. Emrich, P.S. Schnable, and S. Aluru. Assembling genomes on large-scale parallel computers. In *Proc. of IPDPS 2006*, pages 1–10, 2006.
- D.R. Kelley and S.L. Salzberg. Clustering metagenomic sequences with interpolated markov models. *BMC Bioinformatics*, 11:544, 2010. doi: 10.1186/1471-2105-11-544. URL <http://dx.doi.org/10.1186/1471-2105-11-544>.
- DR. Kelley, MC. Schatz, and SL. Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome Biol*, 11(11):R116, 2010. doi: 10.1186/gb-2010-11-11-r116. URL <http://dx.doi.org/10.1186/gb-2010-11-11-r116>.
- J. Korf, K. P. Bjornson, B. P. Chaudhuri, and R. L. Cicero *et al.* Real-time dna sequencing from single polymerase molecules. *Methods Enzymol*, 472:431–455, 2010. doi: 10.1016/S0076-6879(10)72001-2. URL [http://dx.doi.org/10.1016/S0076-6879\(10\)72001-2](http://dx.doi.org/10.1016/S0076-6879(10)72001-2).
- A. Lakdawalla and H. Steenhouse. *Illumina Genome Analyzer II System*, chapter 2, pages 15–28. Wiley-VCH, 1 edition, 2008.
- B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.
- H. Li, J. Ruan, and R. Durbin. Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome Res*, 18(11):1851–1858, Nov 2008. doi: 10.1101/gr.078212.108. URL <http://dx.doi.org/10.1101/gr.078212.108>.

- R. Li, C. Yu, Y. Li, and T. Lam *et al.* Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.
- W. Li and A. Godzik. CD-HIT: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006.
- X. Li and M. S. Waterman. Estimating the repeat structure and length of DNA sequences using l-tuples. *Genome Res*, 13(8):1916–1922, 2003.
- H. Lin, Z. Zhang, M. Zhang, and B. Ma *et al.* ZOOM! Zillions of oligos mapped. *Bioinformatics*, 24(21):2431–2437, 2008.
- Z.i Liu, T. DeSantis, G. Andersen, and R. Knight. Accurate taxonomy assignments from 16S rRNA sequences produced by highly parallel pyrosequencers. *Nucleic Acids Res*, 36(18): e120, 2008.
- G. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *Proc. of WWW*, pages 141–150, 2007.
- B. Manthey and R. Reischuk. The intractability of computing the hamming distance. *THEOR COMPUT SCI*, 1-3(337):331–346, 2005.
- S. Marguerat, B. T. Wilhelm, and J. Bähler. Next-generation sequencing: applications beyond genomes. *Biochem Soc Trans*, 36(Pt 5):1091–1096, 2008.
- M. Margulies, M. Egholm, W. Altman, and S. Attiya *et al.* Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380, Sep 2005. doi: 10.1038/nature03959. URL <http://dx.doi.org/10.1038/nature03959>.

- P. McCullagh and J. A. Nelder. *Generalized Linear Models*. Chapman & Hall, New York, 2 edition, 1989.
- Alice Carolyn McHardy, Héctor García Martín, Aristotelis Tsirigos, and Philip Hugenholtz *et al.* Accurate phylogenetic classification of variable-length dna fragments. *Nat Methods*, 4(1): 63–72, Jan 2007. doi: 10.1038/nmeth976. URL <http://dx.doi.org/10.1038/nmeth976>.
- A. McKenna et al. The genome analysis toolkit: a mapreduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, 20(9):1297–1303, 2010.
- K. McKernan, H. Peckham, G. Costa, and S. McLaughlin *et al.* Sequence and structural variation in a human genome uncovered by short-read, massively parallel ligation sequencing using two-base encoding. *Genome Res.*, 19(9):1527–1541, Sep 2009. doi: 10.1101/gr.091868.109. URL <http://dx.doi.org/10.1101/gr.091868.109>.
- A. E. Men, P. Wilson, K. Siemering, and S. Forrest. *Sanger DNA Sequencing*, chapter 1, pages 3–11. Wiley-VCH, 1 edition, 2008.
- F. Meyer et al. The metagenomics RAST server – a public resource for the automatic phylogenetic and functional analysis of metagenomes. *BMC Bioinf.*, 9(1):386, 2008.
- E. W. Myers, G. G. Sutton, A. L. Delcher, and I. M. Dew *et al.* A whole-genome assembly of drosophila. *Science*, 287(5461):2196–2204, 2000.
- E.W. Myers. The fragment assembly string graph. *Bioinformatics*, 21:ii79–ii85, 2005.
- J. M. Perkel. Sanger who? sequencing the next generation. *Science*, 10:275–279, 2009.
- P.A. Pevzner and H. Tang. Fragment assembly with double-barreled data. *Bioinformatics*, 21: S225–S233, 2001.

- P.A. Pevzner, H. Tang, and M.S. Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences of the United States of America*, 98:9748–9753, 2001.
- H. Poinar, C. Schwarz, J. Qi, B. Shapiro, and R. Macphee *et al.* Metagenomics to paleogenomics: large-scale sequencing of mammoth DNA. *Science*, 311(5759):392–394, 2006.
- W. Qu, S. Hashimoto, and S. Morishita. Efficient frequency-based de novo short-read clustering for error trimming in next-generation sequencing. *Genome Res*, 19(7):1309–15, 2009.
- S. M Rumble, P. Lacroute, A. V Dalca, and M. Fiume *et. al.* SHRiMP: accurate mapping of short color-space reads. *PLoS Computational Biology*, 5(5):e1000386, 2009.
- L. Salmela. Correction of sequencing errors in a mixed set of reads. *Bioinformatics*, 26(10):1284–1290, May 2010. doi: 10.1093/bioinformatics/btq151. URL <http://dx.doi.org/10.1093/bioinformatics/btq151>.
- F. Sanger, S. Nicklen, and A. R. Coulson. Dna sequencing with chain-terminating inhibitors. *Proc Natl Acad Sci U S A*, 74(12):5463–5467, Dec 1977.
- J.M. Santos and M. Embrechts. On the use of the adjusted rand index as a metric for evaluating supervised classification. In *ICANN (2)*, pages 175–184, 2009.
- E.E. Schadt, S. Turner, and A. Kasarskis. A window into third-generation sequencing. *Hum Mol Genet*, 19(R2):R227–R240, Oct 2010. doi: 10.1093/hmg/ddq416. URL <http://dx.doi.org/10.1093/hmg/ddq416>.
- M.C. Schatz, A.L. Delcher, and S.L. Salzberg. Assembly of large genomes using second-

- generation sequencing. *Genome Res*, 20(9):1165–1173, Sep 2010. doi: 10.1101/gr.101360.109.
URL <http://dx.doi.org/10.1101/gr.101360.109>.
- M.C. Schmidt, N.F. Samatova, K. Thomas, and B. Park. A scalable, parallel algorithm for maximal clique enumeration. *J. of Parallel and Distrib. Comput.*, 69(4):417 – 428, 2009.
- J. Schröder, H. Schröder, S. J. Puglisi, and R. Sinha *et al.* SHREC: a short-read error correction method. *Bioinformatics*, 25(17):2157–2163, 2009.
- G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.
- S. Sharma. Advanced Hadoop tuning and optimization.
- J. Shendure and H. Ji. Next-generation DNA sequencing. *Nat Biotechnol*, 26(10):1135–1145, 2008.
- H. Shi, B. Schmidt, W. Liu, and W. Müller-Wittig. Accelerating error correction in high-throughput short-read dna sequencing data with cuda. In *IEEE International Parallel & Distributed Processing Symposium*, pages 1–8, 2009.
- C. Simon *et al.* Phylogenetic diversity and metabolic potential revealed in a glacier ice metagenome. *Appl. Environ. Microbiol.*, 75(23):7519–7526, 2009.
- J.T. Simpson and R. Durbin. Efficient construction of an assembly string graph using the fm-index. *Bioinformatics*, 26(12):i367–i373, Jun 2010. doi: 10.1093/bioinformatics/btq217.
URL <http://dx.doi.org/10.1093/bioinformatics/btq217>.
- JT. Simpson, K. Wong, SD. Jackman, and JE. Schein *et al.* Abyss: a parallel assembler for short read sequence data. *Genome Research*, 19:1117–1123, 2009.

- A. Smith, Z. Xuan, and M. Zhang. Using quality scores and longer reads improves accuracy of solexa read mapping. *BMC Bioinformatics*, 9(1):128–135, 2008. ISSN 1471-2105.
- M.I. Stratton. Genome resequencing and genetic variation. *Nat. Biotechnol.*, 26(1):65–66, 2008.
- M.T. Tammi, E. Arner, E. Kindlund, and B. Andersson. Correcting errors in shotgun sequences. *Nucleic Acids Res*, 31(15):4663–4672, 2003.
- C. Tassell, T. Smith, L. Matukumalli, and J. Taylor *et al.* Snp discovery and allele frequency estimation by deep sequencing of reduced representation libraries. *Nat Methods*, 5(3):247–252, 2008.
- P. Turnbaugh *et al.* A core gut microbiome in obese and lean twins. *Nature*, 457(7228):480–484, 2009.
- J. Venter, K. Remington, J. Heidelberg, and A. Halpern *et al.* Environmental genome shotgun sequencing of the sargasso sea. *Science*, 304(5667):66–74, Apr 2004.
- N. Verberkmoes *et al.* Shotgun metaproteomics of the human distal gut microbiota. *J. ISME*, 3(2):179–189, 2009.
- D. Wall *et al.* Cloud computing for comparative genomics. *BMC Bioinf.*, 11:259, 2010.
- Q. Wang *et al.* Naive Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. *Appl. Environ. Microbiol.*, 73(16):5261–5267, 2007.
- D. Weese, A. Emde, T. Rausch, and A. Döring *et al.* Razers—fast read mapping with sensitivity control. *Genome Research*, 19(9):1646–1654, 2009.

- E. Wijaya, MC. Frith, Y. Suzuki, and P. Horton. Recount: expectation maximization based error correction tool for next generation sequencing data. *Genome Inform*, 23(1):189–201, Oct 2009.
- B Wu and X Pei. A parallel algorithm for enumerating all the maximal k-plexes. In *Emerging technologies in knowledge discovery and data mining*, volume 4819 of *Lecture notes in computer science*, page 476–483, 2009.
- B. Wu, S. Yang, H. Zhao, and B. Wang. A distributed algorithm to enumerate all maximal cliques in mapreduce. In *Frontier of Computer Science and Technology, 2009. FCST '09. Fourth International Conference on*, pages 45–51, 2009.
- R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Trans. Neural Netw.*, 16(3): 645–678, 2005.
- X. Yang, KS. Dorman, and S. Aluru. Reptile: representative tiling for short read error correction. *Bioinformatics*, 26(20):2526–2533, Oct 2010. doi: 10.1093/bioinformatics/btq468. URL <http://dx.doi.org/10.1093/bioinformatics/btq468>.
- K. Y. Yeung and W. L. Ruzzo. Principal component analysis for clustering gene expression data. *Bioinformatics*, 17(9):763–774, Sep 2001.
- D.R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, 18:821–829, 2008.
- D. Zhi, U. Keich, P. Pevzner, and S. Heber *et al.* Correcting base-assignment errors in repeat regions of shotgun assembly. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, 4(1):54–64, 2007.

J. Zola, X. Yang, S. Rospondek, and S. Aluru. Parallel-TCoffee: A parallel multiple sequence aligner. In *In Proc. of ISCA PDCS 2007*, pages 248–253, 2007.

ACKNOWLEDGEMENTS

I would like to thank all the POS committee members for their participation in my preliminary exam and final defense, and for providing me valuable suggestions.

I would like to thank my advisor, Dr. Srinivas Aluru, for providing me the flexibility to select problems that I was interested in; I thank him for spending time to discuss my research, to stay up late and work with me for deadlines, and to teach me a proper way of writing; I thank him for the support of my work and the belief in the quality of my work.

I would like to thank Dr. Karin S Dorman for discussing and contributing research ideas, for helping me to develop statistical thinking, and for her support in difficult times during my PhD. I thank her for collaborating in developing error correction methods.

I would like to thank Dr. Jaroslaw Zola and Dr. Scott Emrich for sharing their views on how to work out to be a PhD. I thank Dr. Zola for collaborating on the problems in metagenomic sequence clustering and parallel sequence alignment. I thank Dr. Emrich for collaborating on assembling metagenomic gene sequences.

I would like to thank Dr. Patrick Schnable for providing a collaborative environment that has helped me to develop biological thinking. I thank him and Dr. Andy Benson for introducing the metagenomic read clustering problem.

I would like to thank Dr. Srikanta Tirthapura for introducing me to sketching techniques.

I would like to thank Dr. David Fernández-Baca for explaining and discussing parametric

alignment problem.

I would like to thank Dr. Guillaume Blin to invite me to IGM of Marne-la-Vallée University for collaborating on Synteny and gene cluster detection problem. I thank Drs. Sylvie Hamel, Florian Sikora in working on this problem together. I thank Dr. Steven Cannon for providing biological insights into this problem.

I would like to thank Abhinav Sarje, Ben Jackson, Olga Nikolova, Jaroslaw Zola, Matthew Mouscou for proof-reading and providing suggestions on various research manuscripts.

I would like to thank Abhinav Sarje, Matthew Moscou, Grishma Parikh, Misha Rajaram, Jaroslaw Zola, Ben Jackson, Olga Wodo, Guillaume Blin, Florian Sikora, Olga Nikolova, Matt Regennitter, Scott Emrich, Anantharaman Kalyanaraman, Pang Ko, Sudip Seal, Andre Wehe, Daniel Standage, and Indranil Roy for listening to my presentations, giving me feedback on my work, and being good friends.

I would like to thank Trish Stauble for helping me throughout PhD, and being a good friend.

I would like to thank BCB program for funding my first year of PhD. I would like to thank both BCB program and ECPE department for partially supporting my travel to various conferences.

I would like to give some special thanks to my remote friends Su, Yangfan for their encouragements.

Finally, I would like to thank my parents Mingzhong and Yuehong for always being there supporting me, my wife Lei for standing by my side during the hard times, and for sharing the joy and bitterness.