Algorithms for Next-Generation High-Throughput Sequencing Technologies

Wei-Chun Kao

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2011-99
http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-99.html

September 2, 2011

Algorithms for Next-Generation High-Throughput Sequencing Technologies

by

Wei-Chun Kao

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science
and the Designated Emphasis
in
Computational and Genomic Biology
and
Communication, Computation, and Statistics

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Yun S. Song, Chair
Professor Sandrine Dudoit
Professor Kannan Ramchandran

Fall 2011

Algorithms for Next-Generation High-Throughput Sequencing Technologies

Abstract

Algorithms for Next-Generation High-Throughput Sequencing Technologies

by

Wei-Chun Kao
Doctor of Philosophy in Computer Science
and the Designated Emphasis
in
Computational and Genomic Biology
and
Communication, Computation, and Statistics

University of California, Berkeley

Professor Yun S. Song, Chair

Recent advances of DNA sequencing technologies are allowing researchers to generate immense amounts of data in a fast and cost effective fashion, enabling genome-wide association study and population genetic research which could not be done a decade ago. There are quite numerous computational challenges arising from this advancement, however. Examples include efficient algorithms for processing raw data generated by sequencing instruments, algorithms for detecting and correcting sequencing errors, algorithms for detecting genome variations from sequence data, just to name a few. Because different sequencing technologies can have drastically different characteristics, these algorithms often need to be adapted in order to produce most accurate results.

In this thesis, I will address a few of the aforementioned problems. First, I will describe two model-based basecalling algorithms for the Illumina sequencing platforms: BayesCall and naiveBayesCall. The novelty of BayesCall algorithm is that it is fully unsupervised, requiring no training data with known labels, and therefore it is applicable to data without a reference sequence. It also dramatically improves sequencing accuracies. Built upon BayesCall algorithm, naiveBayesCall dramatically improves computational efficiency by approximating the original model without sacrificing accuracy. We will also show that improved basecall can have positive effects on the downstream sequence analysis, such as the detection of single nucleotide polymorphism and the assembly of novel genomes.

In the third chapter, an algorithm, called ECHO, for correcting short-read sequencing errors will be described. The correction algorithm efficiently computes all overlaps between sequencing reads and corrects errors by using statistical models. Since it does not rely on reference genomes, ECHO can also be applied to *de novo* sequencing. Most other error correction algorithms require users to specify key parameters, but the optimal values for these parameters are unknown to users and can be hard to specify. Without key parameters being optimized, the effectiveness of error correction algorithm could sometimes be dramatically reduced. Based on statistical models, ECHO optimizes these parameters accordingly. We will show that ECHO can significantly reduce sequence error rates and also facilitate downstream sequence analysis. It is also demonstrated that ECHO can be extended to detect heterozygousity from sequencing data.

These algorithms are developed in hopes to make downstream analysis of sequence data easier and ultimately facilitate genome researches.

Contents

List of Figures

List of Tables

Acknowledgments

During the course of my Ph.D. study, I received countless help, encouragements, and enlightenments from many people. My advisor, Yun S. Song has been a superb mentor since I joined his lab. He gave me maximum freedom to explore ideas yet provide abundant guidance as I needed them. He led me into the field of computational biology and showed me some of the most exciting developments in the field. To some people, the process of pursuing a Ph.D. degree can be traumatic. But under Yun's guidance, the process is filled with the excitement of research and the sense of accomplishment from how much progress I have made. I also want to thank members of my qualifying committee and dissertation committee, Professor Sandarine Dudoit, Professor Lior Pachter, and Professor Kannan Ramchandran. Their suggestions during the qualifying exam greatly helped me shape my research direction. And I also greatly appreciate their comments on my thesis.

I would also like to thank my collaborators, Andrew H. Chan, Aileen Chen, Jerry Hong, Kristian Stevens, and Nanheng Wu. In the BayesCall project, the discussion with Kristian had brought intriguing ideas, which ultimately led to a successful model of Illumina sequencing. Aileen, Nanheng, and Jerry helped on conducting initial study for the error correction projet. I owe a big thanks to Andrew, who was an integral member of the ECHO project. He brought his skills in optimizing programs and made our program ran several folds faster. On top of that, he also came up with ideas for algorithmic improvements that are crucial to our success.

I am grateful to members in Song's group, Paul Jenkins, Matthias Steinrücken, Christopher Hallsworth, Anand Bhaskar, Ma'ayan Bresler, Jesper Nielsen, Joshua Paul, Sara Sheehan, and Junming Yin. From numerous discussions in our group meetings and journal clubs, they made it much easier to access the dense materials we read, and also much easier to keep up with current research.

I also really appreciate the help I received from Kurt T. Miller, Blaine Nelson, Benjamin I. P. Rubinstein, and Alex Shyr. The first few years of my graduate school were quite tough. As an international student, it really required some efforts to settle down and get up to speed. With the help from them, I was able to bring my knowledge into another level and integrate them into my research. Their help was the key to my success. In addition to the academic aspect of their help, they are also great friends and give me supports when I need them. Biking with Kurt, Blaine and Alex is also a fond memory that I will miss greatly.

Professor Peter L. Bartlett also provided significant help to me. In the first few years of my study, he counseled me on the advances in statistical learning theories. These experience opened my eyes and had great influence on my research. I'd also like

to thank members in Bartlett's group with whom I had various interesting exchanges on learning theories. I am truly grateful to Professor Michael I. Jordan. He suggested several research directions to me and guided me on choosing them. I also thank members in Mike's SAIL group as they were always a good source of ideas in machine learning.

The designated emphasis in computational and genomic biology was also a very valuable experience to me. The retreats and talks organized by the program helped me understand the area quickly and also sparkled countless ideas. The designated emphasis in communication, computation, and statistics also greatly helped me grasp the materials in statistics. I would like to thank the professors and the organizers in these DE programs.

Finally, I would like to close by thanking my girlfriend Chia-Chen Chang, my family and my friends. They are the unsung heros who gave me moral support all the way through my graduate study. Their support was crucial and will always be vital to my success.

Chapter 1

Introduction

One goal of the genetic studies is to understand the relationship between genetic materials and the phenotypical variances. Pioneered by numerous scientists, such as Frederick Griffith[17], Oswald Theodore Avery, Colin McLeod, Maclyn McCarty[1], Alfred Hershey and Martha Chase[21], it has been discovered that DNA is the genetic material for most of the life forms. Since then, scientists have been eager to uncover the location and function of each gene in the DNA. The understanding of genes is not only of scientific interest but also of great importance in developing personalized disease preventions and treatments. To fully understand gene and their respective functions, the first step is to decode the nucleotide sequence from DNA, a step called DNA sequencing.

The sequencing of the whole human genome was not possible before the invention of Sanger sequencing[43]. Sanger sequencing uses chain termination method, which consists of a DNA polymerization step and a capillary electrophoresis step. It can generate DNA reads with length as long as one kilo-bases(kb) with error rates as low as one error in 10,000 bases[14]. Together with the fact that it can be automated, Sanger sequencing became the method of choice for DNA sequencing in the last decade. In fact, the first human genome was sequenced with the Sanger sequencing method in the Human Genome Project(HGP) [1]. However, because Sanger sequencing requires a capillary electrophoresis step, the per-base cost and the lengthy preparation time can still be prohibitive. In the case of HGP, it took more than 13 years and over 3 billion dollars to complete the sequencing. Unfortunately, to identify common gene variations and unveil the connection between genes and their associated traits, hundreds if not thousands of individual's genome must be sequenced. It is obvious that population scale genome study is next to impossible with Sanger sequencing.

Currently, thanks to recent advances in high-throughput sequencing technology[3,

[1] `http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml`

36, 37], the generation of immense amounts of raw, unassembled sequence data in fast and cost-effective way has become possible. Most of these new technologies exploit massive parallelization of the DNA polymerization step and eliminate the capillary electrophoresis step. This results in the generation of millions of DNA reads in a matter of a few days. As a consequence, obtaining whole-genome sequence information will become routine soon. This remarkable technological advancement is opening up vast new opportunities for biological sciences and related fields, for example, [22, 20]. The whole-genome description of DNA sequence variation will help us understand the biological mechanisms through which genetic polymorphisms affect phenotypes. However, to bring this promise of whole genome sequencing to fruition, several immediate challenges must be overcome. In particular, the raw error rates of the high-throughput sequencing technologies are several folds higher compared to Sanger sequencing, and the read lengths are several folds shorter.

One immediate remedy for the shorter read lengths and higher error rates is to use reference sequences. Reference sequences are known genomes that were possibly obtained by Sanger sequencing. With a reference sequence from a different individual of the same species or closely related species, high-throughput sequencing reads can be mapped to a reference and have variations identified. This route is pursued by [31, 30, 33, 4]. These methods are computationally very efficient and widely used. Nevertheless, it has been shown that sequencing errors can still affect detection of variations and bias can be introduced when using a reference genome [10, 19, 9, 18]. Moreover, the requirement of a reference genome is sometimes too costly. Therefore, reducing the error rates and increasing the read lengths of high-throughput sequencing reads are promising alternative approaches to using a reference genome. To address these problems, it is necessary to develop scalable, accurate computational tools for data acquisition, characterization, and analysis.

In this text, we will focus on data generated by Illumina sequencing platforms (Illumina). It is broadly adopted because of its low per-base cost and extremely high throughput. (More details on the Illumina sequencing platforms can be found in Chapter 2.) To summarize its capabilities, tens of millions of DNA fragments can be sequenced in a run, and the error rates are usually well below 1% at the beginning of the reads but gradually increase to more than 2% toward the end of the reads. Generally, the read lengths range from 36bp to 100bp, and are limited by the aforementioned increasing error rates. Based on the characteristics of Illumina, there are mainly two approaches to address the short read lengths and the high error rates. One is to develop improved basecall algorithms, and the other is to correct sequencing errors produced by Illumina. By reducing the error rates, one can also increase the read length in each run. In the following, we will briefly describe these

two approaches.

1.1 Developing improved basecall algorithms

The first approach is to develop improved image analysis and base-calling algorithms. In Illumina, each type of base is labeled with fluorescent dyes with different colors, and the fluorescent signals are recorded for each base in each of the DNA fragment. In the ideal case, the identity of the bases can be extracted by transforming signals from color space into base space and choosing the base that has the highest response. However, several types of non-stationary noises largely complicate the basecall process. (More detail on these noises can be found in Chapter 2.) These non-stationary noises include prephasing and phasing effects, death effects, residual effects, and other stochastic noises. Most of these effects are due to incomplete chemical reaction, and the strength of these effects can vary even from read to read in the same run. Because the noise is intrinsic to the chemical process, a basecaller must be able to decouple the noise from actual signals in order to accurately call bases.

There are several studies[50, 12, 41, 28] that try to develop improved basecall algorithms by addressing some of the aforementioned noises. Indeed, by employing more sophisticated statistical methods, it has been demonstrated that it is possible to deliver significant improvements over the tools developed by the manufacturers of the sequencing platforms. Nonetheless, most of these algorithms require a labeled training data set and assume that the noise in the training data set can be directly translated to testing data set. In Chapter 2, we will derive model based basecallers, named BayesCall[26] and naiveBayesCall[24, 25], that do not require a training data set and achieve better accuracy.

The BayesCall algorithm has several novelties. First, it is the first algorithm that addresses all the aforementioned noises. In fact, BayesCall is the first model to reveal the importance of the residual effect. In addition, BayesCall is based on unsupervised machine learning. It uses a fully parameterized probabilistic model to capture all the known non-stationary noises. Therefore, unlike most of other basecall algorithms, it does not require labeled training data. That is, it does not require a known genome to be sequenced as a reference dataset. Additionally, model parameters can be fit to each run without using a reference run. By detailed modeling of the underlying chemical process, the sequencing accuracy is drastically improved, and the sequencing cost is therefore reduced. Empirical studies show that BayesCall remarkably improves the basecall accuracy and also reduces false positives in gene variation detection.

One drawback of the BayesCall algorithm is that it requires a simulated annealing step[29] to extract the most probable bases and thus is computationally expensive.

To tackle this problem, we augment the original inference algorithm by utilizing numerical approximations. The resulting algorithm, called naiveBayesCall, runs orders of magnitude faster than the original BayesCall algorithm. With naiveBayesCall, basecall can be done in hours with accuracy comparable to the orignal BayesCall algorithm. We then show that the improvement in basecallers can translate to improved performance in *de novo* assembly [38, 46, 7, 35, 54, 48, 31, 30, 5].

1.2 Correcting sequencing errors

An alternative approach to reduce error rates is to correct for potential errors after base-calling has been performed by leveraging on the fact that each position in the genome on average is sequenced multiple times. Several methods have been developed for this approach; e.g., see [15, 7, 44, 45, 8, 42, 40, 51, 52]. There are usually two steps that is common in these methods: a neighbor finding step and a correction step. The neighbor finding step attempts to group reads that cover the same region in the original genome. These reads are regarded as "neighbors" to each other. After neighboring reads are identified, one can aggregate information from neighboring reads and correct for potential sequencing errors.

When there is a reference genome, identifying reads that cover the same region can be done via "read mapping" [31, 30], which involves aligning reads against the reference genome. Reads that are mapped to the same region of the reference genome are regarded as neighbors. Subsequently, correction can be done by assessing disagreements of reads that cover the same region. When the reference genome is unavailable, it is harder to find reads that cover the same region. One common approximation to the neighbor finding step is to only consider k-mers instead of the reads. k-mers are nucleic acid sequences with length k, i.e. sub-reads that are of length k. In this approximation, k-mers that are similar to each other is regarded as potential neighbors, and the correction is done in the k-mer level instead of read level.

This approximation is popular for several reasons. First, it leads to very efficient algorithms. Both the memory and computational requirements are minimum since only k-mer information needs to be kept. In addition, many popular *de novo* assemblers are based on de Bruijn graphs which use k-mers as building blocks. Therefore, it is very natural to have a preprocessing step that corrects for potential errors in k-mers. However, reads can sometimes provide key information in recognizing potential errors and resolving them. In Chapter 3, we will develop a read-overlap based error algorithm called ECHO[23].

ECHO does not rely on the k-mer approximation and attempts to use as much read information as possible. Furthermore, ECHO has a few novelties. First, it does

not require users to specify some of the key parameters. The optimal values for these parameters can sometimes be hard for a user to specify, and suboptimal parameters usually yield unfavorable results. Having that in mind, ECHO was designed to search optimal parameters automatically and estimate error characteristics specific to each dataset. The other novelty is that ECHO corrects sequencing errors based on probabilistic models. It can take non-uniform error rates of each postion in each reads into account. It is also easy to generalize it to handle diploid genomes. We then show that ECHO drastically improves the sequencing error rates in both simulated and real datasets. It is worth noting that, in a study of a *Saccharomyces cerevisiae* whole-genome sequencing dataset, we discover that ECHO outperforms other methods on this more complex genome by being more conservative on regions that have abnormal coverages.

Chapter 2

Basecall Algorithms for Illumina Genome Analyzer

2.1 Introduction

There now exist several competing ultra high-throughput sequencing platforms, including the ones from Illumina (Solexa), Roche (454), and Applied Biosystems (SOLiD). While the discussion in this chapter applies specifically to Illumina's Solexa technology (hereafter Illumina), the high-level framework we propose—namely, using graphical models to devise model-based basecalling algorithms—should be applicable to other high-throughput sequencing platforms.

The best known basecallers for the Illumina platforms include *Bustard* (developed by Illumina themselves), *Alta-Cyclic* [12], *BayesCall*[26], *naiveBayesCall*[24, 25] (developed by ourselves), and lately, *Swift*[50] and *Ibis*[28]. We will elaborate on Bustard in Chapter 2.3. Alta-Cyclic is a method based on the support vector machine (SVM) and requires "supervised learning using a rich DNA library with a known reference genome" [12]. Specifically, it requires using a control lane containing a sample with a known reference genome. It then relies on Bustard and alignment to create a rich training set for supervised learning. Note that using a control lane in every sequencing run to create a rich training library can be burdensome, since it incurs cost and takes up space on the flow cell (defined later) that could otherwise be used to sequence a sample of interest to the biologist. Further, Alta-Cyclic's software architecture is designed to run only in a clustered computing environment, thus limiting its utility.

In this chapter, we introduce a novel model-based approach to basecalling, founded on the tools of statistical learning. Our main goal is to model the sequencing process to the best of our knowledge, by taking stochasticity into account and by explicitly

modeling how errors may arise. In contrast to Alta-Cyclic's SVM approach, our method, called *BayesCall*, does not require supervised training and hence does not require using a control lane. Furthermore, BayesCall is flexible enough to incorporate various features of the sequencing process. To illustrate this point, we explicitly model residual effects and incorporate time-dependent parameters into our method. We show that our method significantly improves the accuracy of basecalls, particularly in the later cycles of a sequencing run. For 76-cycle data on a standard viral sample PhiX174, BayesCall achieves about 51% improvement over Bustard in the average per-base error rate, and about 21% improvement over Alta-Cyclic. A detailed study of BayesCall's performance is presented in this chapter.

In our approach, we obtain basecalls by maximizing a posterior distribution of sequences given observed data (i.e., fluorescence intensities) and assuming a uniform prior on sequences. One of the advantages of this approach is that we can readily compute the probability of observing each base, and this probability can be transformed into a useful base-specific quality score. We show that BayesCall not only reduces the error rate substantially, but also produces quality scores with a high discrimination ability [13] that consistently outperforms both Bustard's and Alta-Cyclic's.

Unfortunately, however, this improvement in accuracy came at the price of substantial increase in running time. BayesCall is based on a generative model and performs base-calls by maximizing the posterior distribution of sequences given observed data (i.e., fluorescence intensities). This step involves using the Metropolis-Hastings algorithm with simulated annealing, which is computationally expensive; it would take several days to base-call a single lane using a desktop computer. This slow running time seriously restricts the practicality of BayesCall.

To overcome the difficulty, we develop another algorithm called naiveBayesCall. It is based on the same generative model as in BayesCall and employs the same parameter estimation method. However, in contrast to BayesCall, our alternative algorithm avoids doing Markov chain Monte Carlo sampling in the base-calling part of the algorithm. Instead, naiveBayesCall utilizes approximation and optimization methods to achieve scalability. To test the performance of our method, we use a standard resequencing data of PhiX174 virus, obtained from a 76-cycle run on Illumina's GA II platform. Then, we demonstrate how improved base-calling accuracy may facilitate *de novo* assembly.

We remark that the main novelty of our work is the explicit mathematical description of the sequencing process. Its flexibility should allow one to incorporate additional features which might later be found to be important. Moreover, being a fully parametric model, our approach provides information on the relative importance of various factors that contribute to the observed intensities; such information may

become useful for designing an improved sequencing technology. Lastly, we believe that the framework introduced here provides a basis for designing a more efficient, accurate algorithm in the future.

2.2 Illumina Genome Analyzer

In what follows, we provide a brief overview of the Illumina sequencing platform.

2.2.1 Sequencing-By-Synthesis

In sequencing-by-synthesis (SBS), a large number of random DNA fragments are clonally amplified and used as templates for sequencing in a highly parallel fashion. See [3, 36] for an accessible introduction to currently available SBS platforms and whole-genome resequencing. The Illumina sequencing platform in particular works as follows.

1. A DNA sample from an individual is obtained. The sample contains many copies of the same genomic DNA at full length.

2. Randomly fragment the genomes in the sample and then create a DNA library according to a chosen fragment size.

3. A small amount of single-stranded fragments from the DNA library is placed on a glass surface (called the flow cell). Each single-stranded DNA fragment gets covalently attached to the surface and gets amplified in the neighborhood of the initial attachment, resulting in a *cluster* of about 1000 identical copies of the fragment. The flow cell has eight lanes, and the process just described can generate several million clusters of DNA fragments on each lane. (Remark: Errors in the amplification step may be a significant source of error in basecalls, but quantifying this effect would require a detailed model of the amplification process. In our work, we assume for simplicity that the amplification step is error-free.)

4. Each single-stranded fragment in a given cluster serves as a template, and SBS is carried out by sequentially building up complementary bases as described below. The sequencing platform uses four distinct fluorescently-labeled *terminating bases*, essentially A, C, G, T nucleotides with respective fluorophores and reversible terminators attached. The role of the terminator is to prevent continuation of the complementary strand synthesis.

(a) Add a mixture of DNA polymerase and all four terminating base types to the flow cell. Ideally, the goal of this process is to add exactly a single complementary terminating base to each template, but as we describe later, this process is not perfect and some templates may jump ahead (prephasing) or lag behind (phasing) in building up complementary strands. This is a major source of complication for basecalling.

(b) The clusters are then excited by lasers and CCD images of fluorescence emission are taken four times in optimal wavelengths for the four fluorophores. Many non-overlapping CCD images need to be taken to cover the entire flow cell. Each lane of the flow cell is divided into 330 (respectively, 100) non-overlapping *tiles* in Genome Analyzer I (respectively, II).

(c) Fluorophores and reversible terminators are then chemically removed, and the mixture of polymerase and terminating bases is then added to the flow cell to start the next cycle. Repeating this process for L cycles produces short-reads of length L.

2.2.2 Extracting sequence information

Obtaining actual DNA sequences from the Illumina platform involves two problems: image analysis and basecalling. These steps are described below.

Image Analysis

One of the primary functions of image analysis is to correct for the imperfect repositioning of the CCD camera between cycles and for chromatic aberration of its lens. Currently this correction is done by aligning the images from subsequent cycles to a reference image from the initial cycle.

Another important function of image analysis is to identify clusters from their surrounding background, where each cluster contains identical copies of DNA templates (see step 3 in the above description of SBS). At present, this is done using thresholding and segmentation, which are standard image analysis techniques.

The signal for each cluster is characterized as a time series data of fluorescence intensities and noise. The intensities for each cluster are obtained by summing the pixel values within the cluster and subtracting out its neighborhood background signal.

Basecalling

For each cluster isolated by image analysis, the basecalling step converts the fluorescence signals into actual sequence data with quality scores. A primary challenge in basecalling for this sequencing technology is that the data between cycles are not independent. As explained in step 4(a) of the previous section, some templates in each cluster may undergo imperfect synthesis and jump ahead or fall behind each cycle. These effects become more pronounced in later cycles, thus putting a serious limitation on the read length. There are other stochastic effects in the sequencing process that we need to model to obtain accurate sequence data.

2.3 The Bustard Algorithm

The objective of basecalling is to transform observed intensities into sequences. Bustard achieves this goal by performing the three steps described below.

2.3.1 From intensities to concentrations

We use $\boldsymbol{I}_{t,k} = (I^A_{t,k}, I^C_{t,k}, I^G_{t,k}, I^T_{t,k})' \in \mathbb{R}^{4\times 1}$ to denote the fluorescence intensities of A, C, G, T channels at cycle t in cluster k, after subtracting out the background signals. Define $\boldsymbol{\mathcal{Z}}_{t,k} = (\mathcal{Z}^A_{t,k}, \mathcal{Z}^C_{t,k}, \mathcal{Z}^G_{t,k}, \mathcal{Z}^T_{t,k})' \in \mathbb{R}^{4\times 1}$, where $\mathcal{Z}^A_{t,k}$ denotes the concentration (or number) of templates in cluster k with A-terminators at cycle t; the variables $\mathcal{Z}^C_{t,k}, \mathcal{Z}^G_{t,k}, \mathcal{Z}^T_{t,k}$ are similarly defined. The spectra of the four fluorophores usually overlap [53], and this effect can be modeled as

$$\boldsymbol{I}_{t,k} = \boldsymbol{X}\boldsymbol{\mathcal{Z}}_{t,k}, \tag{2.1}$$

where $\boldsymbol{X} = (X_{ij}) \in \mathbb{R}^{4\times 4}$ is a 16 parameter matrix called the *crosstalk* matrix, with X_{ij} capturing the response in channel i due to fluorescence of a unit concentration of base j. The Bustard basecaller assumes that $\boldsymbol{X}$ is the same for all clusters within a given tile. While some physical properties (such as overlapping dye spectra) modeled by this matrix should remain constant over time, others are not. For instance the relative intensities of dyes to concentration are affected by variations in focus, temperature, and fluorescent illumination. Because the crosstalk matrix $\boldsymbol{X}$ is the unobserved basis for the intensity space, the problem of estimating the crosstalk matrix in multivariate statistics can be generalized as a factor analysis. [32] suggested a specialized iterative method for estimating $\boldsymbol{X}$ and that method is employed in Bustard. Upon estimating $\boldsymbol{X}$, its inverse is used in (2.1) to obtain an estimate of $\boldsymbol{\mathcal{Z}}_{t,k}$.

2.3.2 Renormalization of concentrations

After observed intensities $\boldsymbol{I}_{t,k}$ are transformed into concentrations $\boldsymbol{\mathcal{Z}}_{t,k}$, Bustard tries to address the signal decay problem as follows. First, for each cycle t, compute the average concentration $\bar{\mathcal{Z}}_t$ across a tile using

$$\bar{\mathcal{Z}}_t := \frac{1}{N}\sum_{k=1}^{N}(\mathcal{Z}_{t,k}^A + \mathcal{Z}_{t,k}^C + \mathcal{Z}_{t,k}^G + \mathcal{Z}_{t,k}^T),$$

where N denotes the total number of identified clusters in a given tile. Then, for every cycle t and cluster k, renormalize the concentration $\boldsymbol{\mathcal{Z}}_{t,k}$ by multiplying it with $\bar{\mathcal{Z}}_1/\bar{\mathcal{Z}}_t$. Note that this renormalization leads to the same tile-wide average concentration for all cycles. To avoid introducing more notation, in what follows we use the same symbol $\boldsymbol{\mathcal{Z}}_{t,k}$ to denote renormalized concentrations. For a sequencing run with L cycles, we use $\boldsymbol{\mathcal{Z}}_k = (\boldsymbol{\mathcal{Z}}_{1,k}, \ldots, \boldsymbol{\mathcal{Z}}_{L,k}) \in \mathbb{R}^{4\times L}$ to denote the time series data of renormalized concentrations for cluster k. The subsequent basecalling analysis is done on $\boldsymbol{\mathcal{Z}}_k$.

2.3.3 From renormalized concentrations to sequences

The DNA synthesis process is modeled by the following Markov model:

- *Phasing:* With probability p, no new base is synthesized.
- *Prephasing:* With probability q, two bases are synthesized.
- *Normal Incorporation:* With probability $1 - p - q$, exactly one new base is synthesized.

At cycle 0, all templates start at position 0; i.e., no nucleotide has been synthesized. In each subsequent cycle, the position of the terminator in a template changes from i to j according to the following $(L+1)$-by-$(L+1)$ transition matrix $\boldsymbol{P} = (P_{ij})$, where $0 \le i, j \le L$:

$$P_{ij} = \begin{cases} p, & \text{if } j = i, \\ 1-p-q, & \text{if } j = i+1, \\ q, & \text{if } j = i+2, \\ 0, & \text{otherwise.} \end{cases}$$

The estimation of p and q is done for each lane of the flow cell, and the same p and q are used for all cycles. Note that the (i, j) entry of the matrix $\boldsymbol{P}^t$ corresponds to the probability that a terminator at position i moves to position j after t cycles.

Now, let $\boldsymbol{\mathcal{Z}}_k^\circ = (\boldsymbol{\mathcal{Z}}_{1,k}^\circ, \ldots, \boldsymbol{\mathcal{Z}}_{L,k}^\circ) \in \mathbb{R}^{4\times L}$ denote the concentrations in the ideal case in which there is no phasing or prephasing. Then, the observed concentrations

$\boldsymbol{\mathcal{Z}}_k = (\boldsymbol{\mathcal{Z}}_{1,k}, \ldots, \boldsymbol{\mathcal{Z}}_{L,k})$ in the presence of phasing and prephasing are assumed to be related to $\boldsymbol{\mathcal{Z}}_k^\circ$ by $\boldsymbol{\mathcal{Z}}_k = \boldsymbol{\mathcal{Z}}_k^\circ \boldsymbol{Q}$, where $\boldsymbol{Q} = (Q_{jt})$ is an L-by-L matrix with $Q_{jt} = [\boldsymbol{P}^t]_{0,j}$, the probability that a template terminates at position j after t cycles. More explicitly, $\boldsymbol{\mathcal{Z}}_{t,k} = \sum_{j=1}^{L} \boldsymbol{\mathcal{Z}}_{j,k}^\circ [\boldsymbol{P}^t]_{0,j}$. Hence, to invert the phasing and prephasing effects and to infer $\boldsymbol{\mathcal{Z}}_k^\circ$, Bustard computes $\boldsymbol{\mathcal{Z}}_k \boldsymbol{Q}^{-1}$ for each cluster k. Finally, basecalling for that cluster k is done as follows: For $j = 1, \ldots, L$, the jth base of the templates is chosen to be the row index of the largest entry in column j of the 4-by-L matrix $\boldsymbol{\mathcal{Z}}_k \boldsymbol{Q}^{-1}$. (Recall that the row indices $1, 2, 3, 4$ correspond to A, C, G, T, respectively.)

2.4 The BayesCall Algorithm

Let $s_{1,k}, s_{2,k}, \ldots, s_{L,k}$ denote the length-L prefix of the true complementary DNA sequence in cluster k. Ideally, at cycle t we want each template in the cluster to be synthesizing the base $s_{t,k}$. However, because of the stochastic effects described earlier, at cycle t some templates might be synthesizing the base-pair at position $t' < t$, while others might be synthesizing at position $t' > t$. The extent of this "getting out of phase" phenomenon increases with t.

2.4.1 The underlying graphical model

Here, we introduce a flexible basecalling algorithm based on statistical learning. This framework allows us to handle stochasticity in a more sophisticated way than does Bustard. For example, as we elaborate later, it is possible to incorporate cycle-dependent parameters into our model. For ease of notation, we present our algorithm for the case in which parameters are cycle independent.

The Basic Model

Let $\boldsymbol{e}_i$ denote a 4-component column unit vector with a 1 in the ith entry and 0s elsewhere. As before, we use the basis with A, C, G, T corresponding to indices $1, 2, 3, 4$, respectively. Hence, $s_{t,k} = A$ corresponds to the column vector $\boldsymbol{S}_{t,k} = \boldsymbol{e}_A$, etc. Whenever we modify the character $s_{t,k}$, we modify the corresponding vector $\boldsymbol{S}_{t,k}$ accordingly, and vice versa. We use $\boldsymbol{S}_k = (\boldsymbol{S}_{1,k}, \ldots, \boldsymbol{S}_{L,k})$ to denote the 4-by-L binary *sequence matrix* corresponding to the complementary DNA sequence $s_{1,k}, s_{2,k}, \ldots, s_{L,k}$ in cluster k. The main goal of basecalling is to infer $\boldsymbol{S}_k$, and hence the sequence $s_{1,k}, s_{2,k}, \ldots, s_{L,k}$.

Let $\boldsymbol{Q}_t$ denote the column t of the L-by-L matrix $\boldsymbol{Q}$ defined above. Recall that the jth component of $\boldsymbol{Q}_t$ is $[\boldsymbol{P}^t]_{0,j}$, where the matrix $\boldsymbol{P}^t$ gives the probability distribution

of the position of the terminator at cycle t in a given template. Therefore, the product $\boldsymbol{S}_k\boldsymbol{Q}_t \in [0,1]^{4\times 1}$ gives the probability distribution for a given template in cluster k to end with an A-, C-, G-, or T-terminator.

We explicitly model the decay in intensities as follows. We use $\Lambda_{t,k}$ to denote the random variable corresponding to the per-cluster density of templates that are "active" (i.e., able to synthesize further) at cycle t in cluster k. Our model for stochastic changes in $\Lambda_{t,k}$ is

$$\Lambda_{t,k} = (1-d)\Lambda_{t-1,k} + (1-d)\Lambda_{t-1,k}\epsilon, \tag{2.2}$$

where ϵ is a 1-dimensional Gaussian noise with zero mean and variance σ^2. Recall that, in Bustard, $\mathcal{Z}^A_{t,k}, \mathcal{Z}^C_{t,k}, \mathcal{Z}^G_{t,k}$, and $\mathcal{Z}^T_{t,k}$ denote the concentration of templates in cluster k at cycle t with A-, C-, G-, and T-terminators, respectively. In our framework, $\Lambda_{t,k}\boldsymbol{S}_k\boldsymbol{Q}_t$ is analogous to Bustard's $\boldsymbol{\mathcal{Z}}_{t,k} = (\mathcal{Z}^A_{t,k}, \mathcal{Z}^C_{t,k}, \mathcal{Z}^G_{t,k}, \mathcal{Z}^T_{t,k})'$, and we use $\boldsymbol{Z}_{t,k} = (Z^A_{t,k}, Z^C_{t,k}, Z^G_{t,k}, Z^T_{t,k})'$ to denote $\Lambda_{t,k}\boldsymbol{S}_k\boldsymbol{Q}_t$.

We also explicitly model the stochasticity associated with observing intensities as multivariate Gaussian noise. Our basic model is given by the following formulation:

$$\boldsymbol{I}_{t,k} = \boldsymbol{X}\boldsymbol{Z}_{t,k} + \sum_{b\in\{A,C,G,T\}} Z^b_{t,k}\boldsymbol{\eta}^b, \tag{2.3}$$

where $\boldsymbol{X}$ is the previously described 4-by-4 crosstalk matrix, and $\boldsymbol{\eta}^A, \boldsymbol{\eta}^C, \boldsymbol{\eta}^G, \boldsymbol{\eta}^T \in \mathbb{R}^{4\times 1}$ are independent and identically distributed 4-dimensional Gaussian noises each with zero mean and covariance matrix $\boldsymbol{\Sigma}$. In what follows, we simplify this basic model to speed up the computation and enrich it to make it more accurate.

Simplification: In practice, the L-dimensional column vector $\boldsymbol{Q}_t$ will have only a few dominant components, with the rest being very small. More precisely, the dominant components will be concentrated about the tth entry. Therefore, for each given t, we can simplify the computation by considering only ℓ positions before t and r positions after t. We refer to this simplification as looking at "a window of size $\ell+r+1$ about t" and use $\boldsymbol{Q}^w_t$ to denote the L-dimensional column vector obtained from $\boldsymbol{Q}_t$ by setting the entries outside the window to zero. Then, instead of $\boldsymbol{Z}_{t,k}$, we use

$$\boldsymbol{Z}^w_{t,k} := \Lambda_{t,k}\boldsymbol{S}_k\boldsymbol{Q}^w_t, \tag{2.4}$$

which being the concentration of templates with A-, C-, G-, T-terminators at cycle t, including the contributions of phased and prephased templates in the window w about position t. In our implementation, ℓ and r are free parameters that the user may specify. We used $\ell = 5$ and $r = 5$ in our experiment.

Enrichment

In addition to phasing and prephasing effects, we have observed other residual effects that propagate from one cycle to the next. We speculate that this is perhaps due to incomplete washing of chemicals or an increase in nonspecific illumination. Importantly, we found that modeling such extra residual effects improves the basecall accuracy. In our model, we introduce a parameter α and assume that the observed intensity $\boldsymbol{I}_{t,k}$ at cycle t contains the residual contribution $\alpha(1-d)\boldsymbol{I}_{t-1,k}$ from the previous cycle.

Summary

To recapitulate, our model is

$$\begin{aligned} \boldsymbol{S}_{t,k} &\sim \text{Unif}(\{\boldsymbol{e}_A, \boldsymbol{e}_C, \boldsymbol{e}_G, \boldsymbol{e}_T\}), && (2.5)\\ \Lambda_{t,k}|\Lambda_{t-1,k} &\sim \mathcal{N}((1-d)\Lambda_{t-1,k}, (1-d)^2\Lambda_{t-1,k}^2\sigma^2), && (2.6)\\ \boldsymbol{I}_{t,k}|\boldsymbol{I}_{t-1,k}, \boldsymbol{S}_k, \Lambda_{t,k} &\sim \mathcal{N}(\boldsymbol{\mu}_{t,k}, \|\boldsymbol{Z}_{t,k}^w\|_2^2\boldsymbol{\Sigma}), && (2.7) \end{aligned}$$

where $\|\cdot\|_2$ denotes the 2-norm and

$$\boldsymbol{\mu}_{t,k} = \begin{cases} \boldsymbol{X}\boldsymbol{Z}_{t,k}^w, & \text{if } t = 1, \\ \boldsymbol{X}\boldsymbol{Z}_{t,k}^w + \alpha(1-d)\boldsymbol{I}_{t-1,k}, & \text{if } t > 1. \end{cases} \tag{2.8}$$

We put a uniform (improper) prior on $\Lambda_{1,k}$. A graphical representation of the underlying model is illustrated in Figure 2.1.

2.4.2 Basecalling

Let $\Theta = \{p, q, d, \alpha, \sigma^2, \boldsymbol{X}, \boldsymbol{\Sigma}\}$ denote the set of parameters in our model. We first estimate these parameters for a given tile using the method described presently. Then, our basecall for cluster k is given by $\boldsymbol{S}_k$ in the maximum a posteriori (MAP) estimate of the pair $(\boldsymbol{S}_k, \boldsymbol{\Lambda}_k)$.

In our implementation, we adopt simulated annealing to achieve a faster convergence rate and more accurate MAP estimate. For a given cluster k, the unobserved variables we sample are $\boldsymbol{S}_{t,k}$ and $\Lambda_{t,k}$ for $t \in \{1, \ldots, L\}$. This procedure is detailed below.

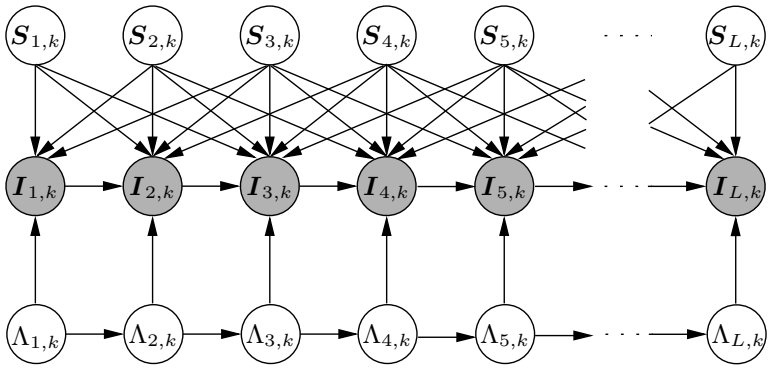


Figure 2.1: **The graphical model for BayesCall. The observed random variables are the intensities $\boldsymbol{I}_{t,k}$.** Base-calling is done by finding the maximum a posteriori estimates of $\boldsymbol{S}_{t,k}$. In this illustration, the window within which we consider phasing and prephasing effects has size 5. In our implementation, we use a window of size 11.

Initialization

For each cycle $t \in \{1, \ldots, L\}$ and cluster k, the initialization $\boldsymbol{S}^{\circ}_{t,k}$ of $\boldsymbol{S}_{t,k}$ is obtained by inferring its associated base using

$$s_{t,k} = \underset{b \in \{A,C,G,T\}}{\operatorname{argmax}} \; (\boldsymbol{X}^{-1}\boldsymbol{I}_{t,k})_b, \tag{2.9}$$

where $(\boldsymbol{X}^{-1}\boldsymbol{I}_{t,k})_b$ denotes the bth component of the matrix product $\boldsymbol{X}^{-1}\boldsymbol{I}_{t,k}$. The initialization $\Lambda^{\circ}_{t,k}$ of $\Lambda_{t,k}$ is given by

$$\Lambda^{\circ}_{t,k} = \underset{\lambda}{\operatorname{argmin}} (\lambda \boldsymbol{X}\boldsymbol{S}^{\circ}_{t,k} - \boldsymbol{I}_{t,k})^T \boldsymbol{\Sigma}^{-1} (\lambda \boldsymbol{X}\boldsymbol{S}^{\circ}_{t,k} - \boldsymbol{I}_{t,k}). \tag{2.10}$$

This is the maximum likelihood estimate of $\Lambda_{t,k}$ if we assume $\boldsymbol{I}_{t,k} \sim \mathcal{N}(\Lambda_{t,k}\boldsymbol{X}\boldsymbol{S}^{\circ}_{t,k}, \boldsymbol{\Sigma})$, which is an approximation of our full model.

Subsequent Iterations

For iteration i of the Metropolis-Hastings algorithm, let $\boldsymbol{\lambda}_k = (\lambda_{1,k}, \ldots, \lambda_{L,k})$ denote the value of $\boldsymbol{\Lambda}_k = (\Lambda_{1,k}, \ldots, \Lambda_{L,k})$ from iteration $i-1$. We first sample a position $x \sim \text{Unif}\{1, 2, \ldots, L\}$ to modify and update $\boldsymbol{S}_{x,k}$ according to

$$\mathbb{P}(\boldsymbol{S}_{x,k} = \boldsymbol{e}_b) \propto (\boldsymbol{X}^{-1}\boldsymbol{I}_{x,k})_b. \tag{2.11}$$

Then, we update $\Lambda_{x,k}$ according to a proposal distribution that is an approximation to the conditional distribution $\mathbb{P}(\Lambda_{x,k} \mid \boldsymbol{I}_k, \boldsymbol{S}_k, \mathcal{D}_x(\boldsymbol{\lambda}_k))$, where $\mathcal{D}_x(\boldsymbol{\lambda}_k)$ denotes $\boldsymbol{\lambda}_k$ with the xth component $\lambda_{x,k}$ deleted. More exactly, we update $\Lambda_{x,k}$ according to the proposal distribution

$$\Lambda_{x,k}|\boldsymbol{S}_k, \lambda_{x-1,k}, \lambda_{x,k}, \lambda_{x+1,k}, \boldsymbol{I}_{x,k}, \boldsymbol{I}_{x-1,k} \sim \mathcal{N}(m_{x,k}, v_{x,k}^2), \tag{2.12}$$

where the mean $m_{x,k}$ and variance $v_{x,k}^2$ of the normal distribution are given by

$$\begin{aligned} v_{x,k}^2 &= \left[\frac{1}{(1-d)^2\lambda_{x-1,k}^2\sigma^2} + \frac{(1-d)^2}{\lambda_{x+1,k}^2\sigma^2} + \frac{(\boldsymbol{X}\boldsymbol{S}_k\boldsymbol{Q}_x^w)'\boldsymbol{\Sigma}^{-1}(\boldsymbol{X}\boldsymbol{S}_k\boldsymbol{Q}_x^w)}{\|\boldsymbol{z}_{x,k}\|_2^2}\right]^{-1}, \\ m_{x,k} &= v_{x,k}^2\left[\frac{1}{(1-d)\lambda_{x-1,k}\sigma^2} + \frac{1-d}{\lambda_{x+1,k}\sigma^2} + \frac{(\boldsymbol{X}\boldsymbol{S}_k\boldsymbol{Q}_x^w)'\boldsymbol{\Sigma}^{-1}(\boldsymbol{I}_{x,k} - \boldsymbol{r}_{x-1,k})}{\|\boldsymbol{z}_{x,k}^w\|_2^2}\right], \end{aligned}$$

with

$$\boldsymbol{z}_{x,k}^w = \lambda_{x,k}\boldsymbol{S}_k\boldsymbol{Q}_x^w,$$

and

$$\boldsymbol{r}_{x-1,k} := \begin{cases} 0, & \text{if } x = 1, \\ \alpha(1-d)\boldsymbol{I}_{x-1,k}, & \text{if } x > 1. \end{cases}$$

A detailed derivation of the above distribution is provided in Chapter A.1. In computing $v_{x,k}^2$ and $m_{x,k}$, all terms involving $x-1$ are set to zero if $x \leq 1$ and all terms involving $x+1$ are also set to zero if $x \geq L$.

Simulated Annealing: Our target distribution is

$$\mathbb{P}(\boldsymbol{S}_k, \boldsymbol{\Lambda}_k|\boldsymbol{I}_k, \Theta),$$

where $\boldsymbol{\Lambda}_k = (\Lambda_{1,k}, \ldots, \Lambda_{L,k})$. To obtain the MAP estimate of $(\boldsymbol{S}_k, \boldsymbol{\Lambda}_k)$ efficiently, we adopt simulated annealing. In an execution with n total iterations, the temperature in the ith iteration is taken as $(n-i+1)/n$. The main advantage of simulated annealing over the straight Metropolis-Hastings algorithm is its enhanced ability to converge to the maximum in fewer iterations, leading to a more accurate MAP estimate.

2.4.3 Parameter estimation

As above, let Θ denote the set of parameters in our model. The parameters are assumed to be shared across all clusters within a given tile, so Θ will be estimated only once per tile. Our estimation procedure uses the expectation-maximization (EM) algorithm, viewing $\boldsymbol{S}_k$ and $\boldsymbol{\Lambda}_k$ as missing data. Let K denote the number of clusters

used in parameter estimation. (In the empirical study discussed in Chapter 2.5, we used $K = 250$.) The log joint probability over the K clusters can be written as

$$\log \prod_{k=1}^{K} \mathbb{P}(\boldsymbol{I}_k, \boldsymbol{\Lambda}_k, \boldsymbol{S}_k \mid \Theta) \tag{2.13}$$

$$\begin{aligned} &= \log \left\{ \prod_{k=1}^{K} \Bigg[\mathbb{P}(\boldsymbol{S}_k \mid \Theta) \mathbb{P}(\Lambda_{1,k}) \mathbb{P}(\boldsymbol{I}_{1,k} \mid \Lambda_{1,k}, \boldsymbol{S}_k, \Theta) \right. \\ &\quad \left. \times \prod_{t=2}^{L} \mathbb{P}(\boldsymbol{I}_{t,k} \mid \boldsymbol{I}_{t-1,k}, \Lambda_{t-1,k}, \Lambda_{t,k}, \boldsymbol{S}_k, \Theta) \mathbb{P}(\Lambda_{t,k} \mid \Lambda_{t-1,k}, \Theta) \Bigg] \right\} \\ &= -\frac{1}{2} \sum_{k=1}^{K} \sum_{t=1}^{L} \left[\log \det(\|\boldsymbol{Z}_{t,k}^{w}\|_2^2 \boldsymbol{\Sigma}) + \frac{(\boldsymbol{I}_{t,k} - \boldsymbol{\mu}_{t,k})' \boldsymbol{\Sigma}^{-1} (\boldsymbol{I}_{t,k} - \boldsymbol{\mu}_{t,k})}{\|\boldsymbol{Z}_{t,k}^{w}\|_2^2} \right] \\ &\quad -\frac{1}{2} \sum_{k=1}^{K} \sum_{t=2}^{L} \left[2 \log((1-d)\Lambda_{t-1,k}\sigma) + \frac{(\Lambda_{t,k} - (1-d)\Lambda_{t-1,k})^2}{((1-d)\Lambda_{t-1,k}\sigma)^2} \right] \\ &\quad + \text{constant}, \end{aligned} \tag{2.14}$$

where $\boldsymbol{\mu}_{t,k}$ is defined in (2.8), $\det(\|\boldsymbol{Z}_{t,k}^{w}\|_2^2 \boldsymbol{\Sigma})$ denotes the determinant of $\|\boldsymbol{Z}_{t,k}^{w}\|_2^2 \boldsymbol{\Sigma}$, and the constant term in last line depends only on the total number L of cycles and the number K of clusters. Let Θ_i denote the set of parameters in the ith EM iteration. The estimate of Θ_i is given by

$$\Theta_i = \underset{\Theta}{\operatorname{argmax}}\, \mathbb{E}_{\Theta_{i-1}} \left[\log \prod_{k=1}^{K} \mathbb{P}(\boldsymbol{I}_k, \boldsymbol{\Lambda}_k, \boldsymbol{S}_k \mid \Theta) \right], \tag{2.15}$$

where the expectation is taken with respect to $\mathbb{P}(\boldsymbol{\Lambda}_k, \boldsymbol{S}_k \mid \boldsymbol{I}_k, \Theta_{i-1})$ and is computed using Monte-Carlo integration with the Metropolis-Hastings algorithm.

In the maximization step, we optimize the expected log-likelihood with respect to one parameter at a time, and iterate through all parameters. One can obtain analytic solutions for updating $\alpha, \sigma^2, \boldsymbol{X}, \boldsymbol{\Sigma}$ in the maximization step, but there is no analytic solution for updating d, p and q. We therefore employ an interior point method to maximize the expected log-likelihood function with respect to p and q directly and approximate d with a simpler formulation. Details can be found in Chapter A.2.

2.4.4 Cycle-dependent parameters

Both Alta-Cyclic [12] and Bustard assume that parameters are cycle independent. One novelty of our model is the ability to incorporate cycle-dependent parameters.

We will see in Chapter 2.5 that the use of cycle-dependent parameters not only increases the accuracy significantly but also helps us to understand the underlying noise structure better. This may provide information on what can be improved in the Illumina technology.

Although the algorithm described above is for cycle-independent parameters, it is straightforward to incorporate cycle dependency into our probabilistic framework. For concreteness, consider the decay parameter d in (2.2). Although in general the total observed intensity tends to decay with cycles, it neither decays at a constant rate nor is a monotonically decreasing function of time; various stochastic effects (e.g., temperature fluctuation) can lead to fluctuations in the total intensity as time passes. Hence, to model the fluctuation in intensities more accurately, we modify (2.2) as

$$\Lambda_{t,k} = (1 - d_t)\Lambda_{t-1,k} + (1 - d_t)\Lambda_{t-1,k}\epsilon_t, \tag{2.16}$$

where the rate d_t now depends on the cycle t, and the 1-dimensional Gaussian noise ϵ_t with zero mean now has cycle-dependent variance σ_t^2. We remark that d_t may take on a negative value for some cycle t. Hence, unlike in (2.2), it no longer admits interpretation as a pure decay parameter.

More generally, we use the subscript t to denote cycle dependency. Our current implementation can handle the following cycle-dependent parameters: $d_t, \alpha_t, \sigma_t^2, \boldsymbol{X}_t, \boldsymbol{\Sigma}_t$. To avoid over-fitting and to reduce the number of clusters required to estimate parameters, we divide the read length L into non-overlapping windows of size 5 and assume that the parameters remain constant within each window. Cycle-dependent parameters can be estimated using the EM algorithm. To reduce the fluctuation of parameters between windows, we devised an estimation method that uses information from adjacent windows. See Chapter A.2 for details.

We observed that d_t is highly correlated with the average intensity and that the contribution of residual effects tends to grow with t. Such observations may help us to characterize the intrinsic properties of the Illumina platform.

2.5 Empirical Study of the Bayescall Algorithm

In this section, we compare the performance of our new algorithm BayesCall with that of Bustard, Alta-Cyclic [12].

2.5.1 Data and test setup

In our empirical study, we used a standard resequencing data of PhiX174 virus, provided to us by the DPGP Sequencing Lab at UC Davis. The data were obtained from a 76-cycle run on the Genome Analyzer II platform, with the viral sample in a single lane of the flow cell. The lane consisted of 100 tiles, containing a total of 14,820,478 clusters. Illumina's base-calling pipeline, called Integrated Primary Analysis and Reporting, was applied to the image data to generate intensity files.

The entire intensity data were used to train Alta-Cyclic and BayesCall. Further, since Alta-Cyclic requires a labeled training set, the reads base-called by Bustard and the PhiX174 reference genome were also provided to Alta-Cyclic. To estimate parameters in BayesCall and naiveBayesCall, the intensity data for only 250 randomly chosen clusters were used.

To create a classification data set for testing the accuracy of the four base-calling algorithms, the sequences base-called by Bustard were aligned against the PhiX174 reference genome, and those reads containing more than 22 mismatches (i.e., with more than 30% of difference) were discarded. This filtering step reduced the total number of clusters to 6,855,280, and the true sequence associated with each cluster was assumed to be the 76-bp string in the reference genome onto which the alignment algorithm mapped the sequence base-called by Bustard. The same set of clusters was used to test the accuracy of all four methods.

Note that since the classification data set was created by dropping those clusters for which Bustard produced many errors, the above experiment setup slightly favored Bustard. Also, it should be pointed out that since Alta-Cyclic was trained on the entire lane, it actually had access to the entire testing data set during the training phase.

2.5.2 Short-read data and experiment setup

To test the performance of our method, we used sequencing data on a standard viral sample PhiX174, provided to us by Illumina and the DPGP Sequencing Lab at UC Davis. The data from Illumina were obtained from a 36-cycle run on the Genome Analyzer I (GA-I) platform, while the data from UC Davis were obtained from a 76-cycle run on the Genome Analyzer II (GA-II) platform. PhiX174 has a known assembled genome, which we assumed to be correct. The viral sample has two outstanding features. First, in contrast to a typical diploid genomic sample, there is only one unique underlying parent DNA molecule. Secondly, the parent DNA sample is relatively short and complex, and thus has a well characterized sequence alignment.

To create a classification data set, sequences determined by the Bustard basecaller

were aligned against the assembled reference sequence. Illumina's ELAND alignment algorithm allows only mismatches and is constrained to at most 2 of them in the first 32 bp, effectively removing some reads with very high error rates from the test. We further discarded reads which could not be aligned to the reference genome with at least 70% identity (out of the entire read length). In the remaining set of reads, we then assumed that all mismatches in alignment are basecalling errors. The same set of reads was used to test the accuracy of Alta-Cyclic and BayesCall, thus favoring Bustard a bit. For each data set, we used 4 tiles for basecalling, resulting in 19,063 reads and 686,268 bases for the 36-cycle data, and 268,997 reads and 20,443,772 bases for the 76-cycle data.

Recall that Alta-Cyclic requires supervised learning using a rich training set. We ran Alta-Cyclic only on the 76-cycle data, since we did not have a sufficiently large training set available to us for the 36-cycle data. The 76-cycle data from GA-II consisted of 100 tiles in total. In constructing the training set for Alta-Cyclic, we omitted the 4 test tiles mentioned above and used the remaining 96 tiles. Alta-Cyclic used Bustard and alignment to create a training set containing about 100 000 reads.

2.5.3 Convergence of simulated annealing

Figure 2.2 shows the convergence of simulated annealing with $1000, 5000, 10\,000$, and $20\,000$ total iterations. Recall that the temperature parameter in the ith iteration of simulated annealing is taken as $(n - i + 1)/n$, where n denotes the total number of iterations. We remark that the MAP estimate for $n = 1000$ has a lower likelihood. However, although using a larger value of n maximizes the likelihood better, the inferred MAP estimate of $\boldsymbol{S}_k$ does not change so much, if at all. We conclude that the total number n of annealing iterations can be chosen to be less than $10\,000$.

2.5.4 Parameters in BayesCall

In contrast to Alta-Cyclic [12], our method does not require supervised learning. In BayesCall, it is possible to estimate the parameters of the underlying model using a small number of randomly chosen clusters. To estimate the cycle-dependent parameters $d_t, \alpha_t, \sigma_t^2, \boldsymbol{X}_t, \boldsymbol{\Sigma}_t$ for a given tile, we randomly chose a total of 250 clusters from the tile and used the algorithm described in Chapter 2.4.3 to perform parameter estimation. (We also tried using 150 and 500 clusters in the training set. The accuracy of basecalls changed very little, while the running time of parameter estimation scaled roughly linearly with the number of clusters. See Chapter A.2 for details.) Here, we highlight the cycle dependency of the parameters in our model.

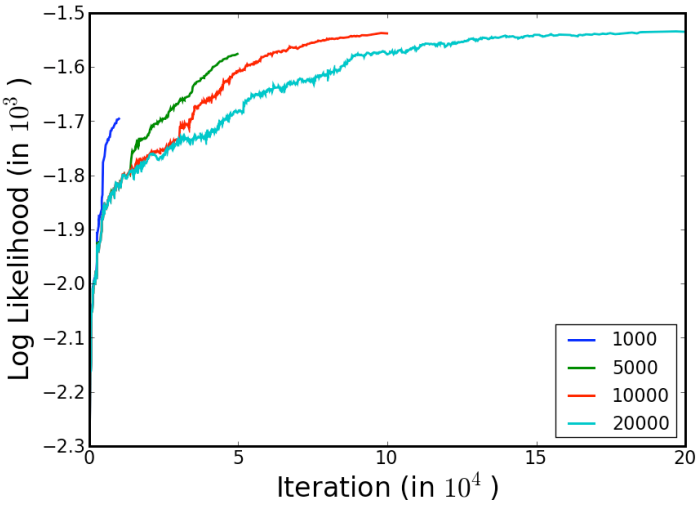
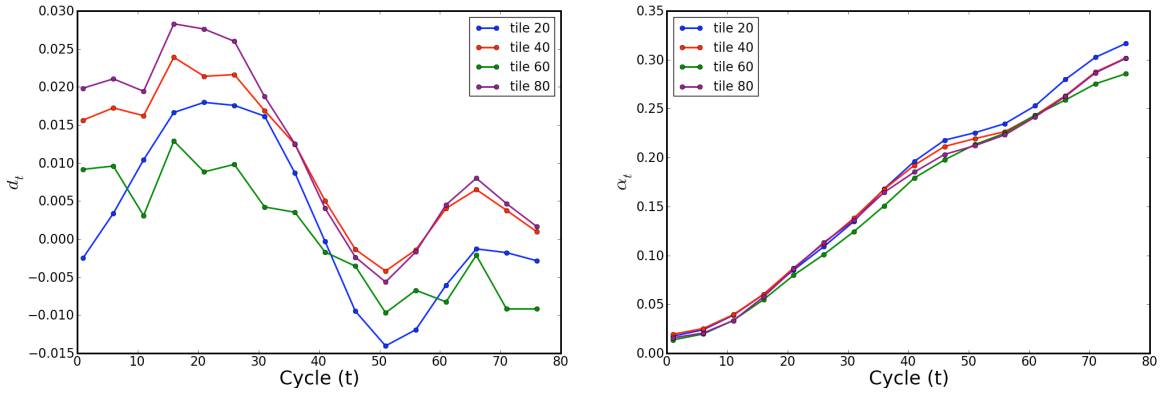


Figure 2.2: **Convergence of simulated annealing with** $1000, 5000, 10000,$ **and** 20000 **total iterations.** The temperature parameter in the ith iteration of simulated annealing is taken as $(n-i+1)/n$, with n being the total number of iterations. Although using a larger value of n maximizes the likelihood slightly better, the inferred MAP estimate of $\boldsymbol{S}_k$ does not change so much.

Recall that Bustard tries to address the signal decay problem by renormalizing concentrations, whereas in our method, we explicitly model how the per-cluster density of active templates evolve according to 2.16. Figure 2.3(a) illustrates why the rate d_t should be modeled as a cycle-dependent parameter. It shows that d_t varies significantly over cycles. The parameter α_t captures extra residual effects in addition to phasing and prephasing that propagate from one cycle to the next. Our estimates of α_t for the 76-cycle data are shown in Figure 2.3(b). This plot illustrates that the extra residual effects captured by α_t grow with cycle t. We found that it is important to model extra residual effects in later cycles to improve the basecall accuracy.

2.5.5 A detailed example

For a particular cluster k in the 76-cycle PhiX174 data, Figure 2.4(a) shows the observed intensity $\boldsymbol{I}_{t,k}$ for $t = 1, \ldots, 76$. We can see that in later cycles, the observed intensity of base T increases abnormally; we refer to this as "anomalous T" effect. This was also noted as an anomaly in [12]. Because of this anomaly, Bustard produced many basecalling errors—14 errors, to be exact, with 13 of them incorrectly called as T. Alta-Cyclic made 3 basecalling errors on this cluster, with all three being incorrectly called as T. See Figure 2.5 for details; errors are indicated by "*" symbols.



(a) (b)

Figure 2.3: **Estimated cycle-dependent parameters for 4 different tiles of the 76-cycle PhiX174 data set.** These plots illustrate that parameters change over time and that the residual effect associated with α_t tends grow with time. (a) d_t. (b) α_t.

In contrast, our method BayesCall was able to call all 76 bases correctly. We can explain this as follows. The inferred mean $\boldsymbol{\mu}_{t,k}$ of the distribution for $\boldsymbol{I}_{t,k}$ can be decomposed into $\Lambda_{t,k}\boldsymbol{X}_t\boldsymbol{S}_k\boldsymbol{Q}_t^w$ and $\alpha_t(1-d_t)\boldsymbol{I}_{t-1,k}$, the latter capturing extra residual effects. This decomposition is illustrated in Figure 2.4(b) and Figure 2.4(c). From these figures, we see that the aforementioned anomaly can be attributed to growing residual effects (illustrated in Figure 2.4(c)) in later cycles. Because of its ability to decouple such residual effects from other stochastic effects, BayesCall was able to call all bases correctly for this cluster. Clearly the presence of artifacts of this kind supports the value of modeling residual effects.

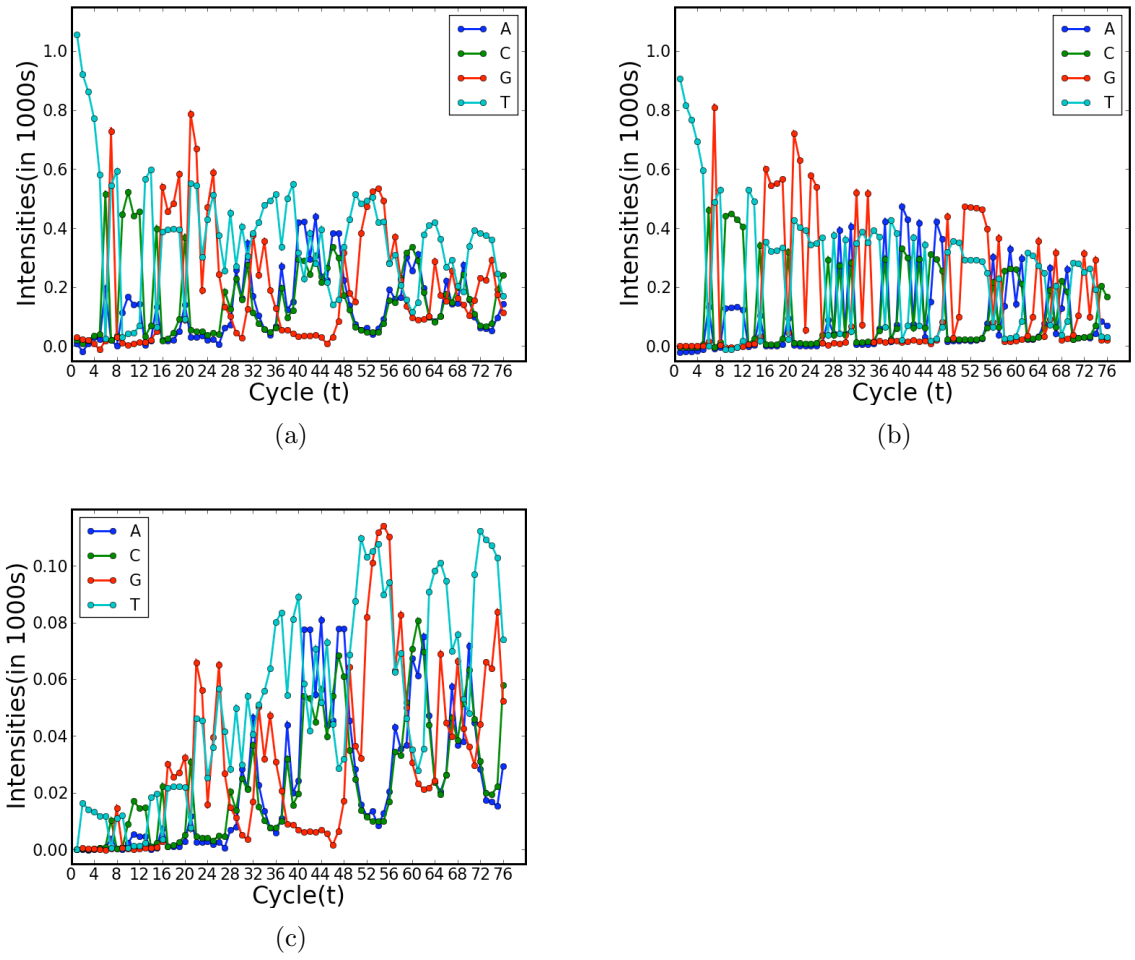


Figure 2.4: **Observed intensities and the decomposition of $\boldsymbol{\mu}_{t,k}$ for a particular cluster k in the 76-cycle PhiX174 data.** (a) Observed intensities $\boldsymbol{I}_{t,k}$. (b) The contribution of $\Lambda_{t,k}\boldsymbol{X}_t\boldsymbol{S}_k\boldsymbol{Q}_t^w$ to $\boldsymbol{\mu}_{t,k}$. (c) The contribution of the residual effect $\alpha_t(1-d_t)\boldsymbol{I}_{t-1,k}$.

```
True sequence     : TTTTTCGTCCCCTTCGGGGCGGTGGTCTATAGTGTTATTAATATCAAGTTGGGGGAGCACATTGTAGCATTGTGCC
Bustard result    : TTTTTCGTCCCCTTCGGGGCGGTGGTTTTTAGTTTTTTTAATATTAAGTTTGGGGGGCACATTTTTTTATTTTTTC
Errors                                    * *    *  *       *     *    *       * ***   * **

True sequence     : TTTTTCGTCCCCTTCGGGGCGGTGGTCTATAGTGTTATTAATATCAAGTTGGGGGAGCACATTGTAGCATTGTGCC
Alta-Cyclic result: TTTTTCGTCCCCTTCGGGGCGGTGGTCTATAGTTTTATTAATATTAAGTTGGGGGAGCACATTGTAGCATTTTGCC
Errors                                          *         *                         *
```

Figure 2.5: **Basecalling results for the particular cluster discussed in Figure 2.4.** Our method BayesCall called all 76 bases correctly for this particular cluster. In contrast, Bustard made 14 basecalling errors, with 13 of them incorrectly called as T. Alta-Cyclic made 3 errors, with all of them incorrectly called as T. Basecalling errors are indicated by "*"s. In general, both Bustard and Alta-Cyclic tend to suffer more from the "anomalous T" effect than does our method. (See text for details.)

Table 2.1: Comparison of error rates on PhiX174 data.

	36-cycle data		76-cycle data	
Method	By Base	By Read	By Base	By Read
(1) Bustard	0.01313	0.29806	0.01557	0.39484
(2) Alta-Cyclic	n/a	n/a	0.00969	0.31150
(3) BayesCall	0.00805	0.17757	0.00762	0.23188
Improvement of (3) over (1)	39%	40%	51%	41%
Improvement of (3) over (2)	n/a	n/a	21%	26%

Alta-Cyclic was run only on the 76-cycle data. The "by-base" error rate refers to the ratio of the number of miscalled bases to the total number of basecalls made, while the "by-read" error rate refers to the ratio of the number of reads each with at least one miscalled base to the total number of reads considered.

2.5.6 Summary of basecall accuracy

Shown in Table 2.1 are the overall error rates of the three basecalling methods, averaged over four tiles for each data set. In the table, "by-base" error rate refers to the ratio of the number of miscalled bases to the total number of basecalls made, while "by-read" error rate refers to the ratio of the number of reads each with *at least* one miscalled base to the total number of reads considered. For both data sets we considered, our new method BayesCall produced significantly more accurate basecalls than did Bustard. For the 76-cycle PhiX174 data from GA-II, BayesCall achieved an improvement of about 51% over Bustard in the by-base error rate, and about 41% improvement in the by-read error rate. For the same data set, BayesCall outperformed Alta-Cyclic as well, achieving an improvement of about 21% in the by-base error rate, and about 26% improvement in the by-read error rate.

For a finer comparison of the methods, we examined the per-cycle error rates, illustrated in the plots in the left column of Figure 2.6. Bustard and BayesCall had comparable error rates in the first 25 or so cycles. However, BayesCall had substantially lower error rates in later cycles compared to Bustard, and the difference tended to increase with cycles. This observation suggests that it is feasible to run the sequencing machine for longer cycles and obtain useful sequence information for longer reads by using an improved basecalling algorithm such as ours. For the 76-cycle data, BayesCall had a lower average error rate than that of Alta-Cyclic's for every cycle. Although Alta-Cyclic was considerably more accurate than Illumina's basecaller Bustard in later cycles, note that the opposite was true for earlier cycles.

Shown in the right column of Figure 2.6 are histograms for the number n_e of errors per read. As mentioned before, BayesCall produced substantially more perfect reads

Table 2.2: Confusion matrices for the 76-cycle PhiX174 data on GA-II.

	Base called by Bustard			
True	A	C	G	T
A	0.98896	0.00337	0.00296	0.00470
C	0.00877	0.97716	0.00336	0.01071
G	0.00485	0.00252	0.98617	0.00646
T	0.00289	0.00517	0.00665	0.98529

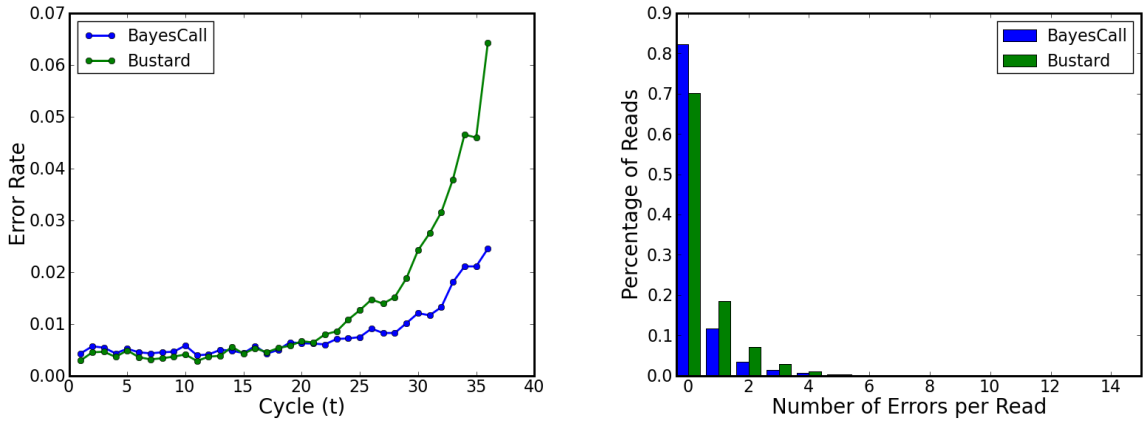
	Base called by Alta-Cyclic			
True	A	C	G	T
A	0.99404	0.00235	0.00141	0.00220
C	0.00586	0.98678	0.00144	0.00591
G	0.00336	0.00091	0.99288	0.00285
T	0.00104	0.00273	0.00338	0.99285

	Base called by BayesCall			
True	A	C	G	T
A	0.99548	0.00179	0.00129	0.00144
C	0.00500	0.98908	0.00132	0.00460
G	0.00349	0.00090	0.99384	0.00176
T	0.00158	0.00339	0.00333	0.99170

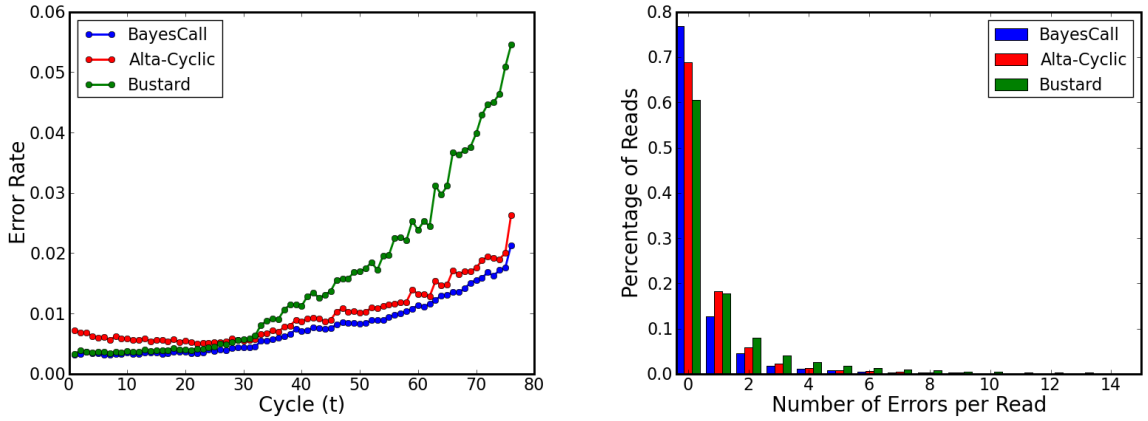
The (x, y) entry in each table corresponds to the percentage of times that base x is called as base y. These tables illustrate that BayesCall is in general less "confused"; that is, the diagonal entries for BayesCall are higher than the corresponding entries for the other methods, except for the (T,T) entry in Alta-Cyclic. Further, BayesCall suffers less from the "anomalous T" effect than does Bustard or Alta-Cyclic; i.e., the (A,T), (C,T), and (G,T) entries for BayesCall are less than the corresponding entries for Bustard or Alta-Cyclic.

(i.e., with $n_e = 0$) than did either Bustard or Alta-Cyclic. Furthermore, for $n_e > 1$, the number of reads with n_e errors was smaller in BayesCall than in either Bustard or Alta-Cyclic.

A statistic that is of interest is the percentage $c(x, y)$ of times that base x is called as base y by a basecalling algorithm. This information can be summarized in what we call a *confusion matrix* $\boldsymbol{C}$, in which the (x, y) entry corresponds $c(x, y)$. For the 76-cycle data on GA-II, the confusion matrices $\boldsymbol{C}_{\text{Bustard}}, \boldsymbol{C}_{\text{Alta-Cyclic}}$, and $\boldsymbol{C}_{\text{BayesCall}}$ of Bustard, Alta-Cyclic, and BayesCall, respectively, are shown in Table 2.2. For this data set, every diagonal entry (x, x) (i.e., the percentage of correct calls) in $\boldsymbol{C}_{\text{BayesCall}}$ is larger than that in $\boldsymbol{C}_{\text{Bustard}}$, while every off-diagonal entry (x, y) (i.e., the percentage of erroneous calls) in $\boldsymbol{C}_{\text{BayesCall}}$ is smaller than that in $\boldsymbol{C}_{\text{Bustard}}$. Also, BayesCall is less "confused" than Alta-Cyclic, in that the diagonal entries in $\boldsymbol{C}_{\text{BayesCall}}$ are higher



(a)



(b)

Figure 2.6: **Average per-cycle error rates and histograms for the number of errors per read.** BayesCall had substantially lower error rates in later cycles compared to Bustard, and the difference tended to increase with cycles. Further, BayesCall produced substantially more perfect reads than did Bustard. Alta-Cyclic was run only on the 76-cycle PhiX174 data set. BayesCall had a lower average error rate than that of Alta-Cyclic's for all cycles. Note that although Alta-Cyclic is more accurate than Bustard in later cycles, the opposite is true for earlier cycles. (a) Results for the 36-cycle data set. (b) Results for the 76-cycle data set.

Table 2.3: Joint errors in Bustard and BayesCall for the 76-cycle PhiX174 data on GA-II.

	# BayesCall Errors								
# Bustard Errors	0	1	2	3	4	5	6	7	8
0	0.5979	0.0069	0.0005	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000
1	0.1061	0.0672	0.0030	0.0005	0.0001	0.0000	0.0000	0.0000	0.0000
2	0.0354	0.0245	0.0181	0.0015	0.0003	0.0001	0.0000	0.0000	0.0000
3	0.0148	0.0123	0.0085	0.0038	0.0011	0.0002	0.0001	0.0000	0.0000
4	0.0073	0.0065	0.0052	0.0033	0.0018	0.0006	0.0002	0.0000	0.0000
5	0.0037	0.0037	0.0033	0.0027	0.0019	0.0011	0.0004	0.0002	0.0001
6	0.0019	0.0022	0.0021	0.0019	0.0016	0.0011	0.0007	0.0003	0.0002
7	0.0012	0.0013	0.0015	0.0013	0.0011	0.0010	0.0007	0.0005	0.0002
8	0.0007	0.0008	0.0011	0.0009	0.0009	0.0008	0.0007	0.0005	0.0004

The (x, y) entry corresponds to the percentage of reads with x errors in Bustard and y errors in BayesCall. For $x < y$, the upper diagonal entry (x, y) is generally much smaller than the corresponding the lower diagonal entry (y, x), thus indicating that BayesCall generally produces sequence reads with substantially fewer errors than does Bustard. See Figure 2.7 for a heat plot of joint error a larger range of x and y.

than the corresponding entries in $\boldsymbol{C}_{\text{Alta-Cyclic}}$, except for the (T,T) entry.

Recall that the "anomalous T" effect refers to calling abnormally high percentages of other bases erroneously as T. In Table 2.2, the T column of $\boldsymbol{C}_{\text{Bustard}}$ is abnormally high, suggesting that Bustard suffers from the "anomalous T" effect. BayesCall suffers much less from this effect than does Bustard or Alta-Cyclic; i.e., the (A,T), (C,T), and (G,T) entries of $\boldsymbol{C}_{\text{BayesCall}}$ are significantly smaller than the corresponding entries in $\boldsymbol{C}_{\text{Bustard}}$ or $\boldsymbol{C}_{\text{Alta-Cyclic}}$.

To compare further the performance of Bustard and BayesCall, we examined the percentage of reads with x errors in Bustard and y errors in BayesCall. For $x, y \leq 8$, these numbers are summarized in the joint error matrix shown in Table 2.3. Figure 2.7 illustrates the overall pattern for a larger range of x and y. What is abundantly clear is that the upper diagonal entries (x, y), for $x < y$, are much smaller than the corresponding lower diagonal entries (y, x). This indicates that BayesCall generally produces sequence reads with substantially fewer errors than does Bustard.

2.5.7 Gain in accuracy from modeling extra residual effects

In BayesCall, extra residual effects are captured by the parameter α_t. To examine the advantage of modeling residual effects, we considered the following three cases of BayesCall: (1) No residual effect (i.e., impose $\alpha_t = 0$ for all t). (2) Constant residual

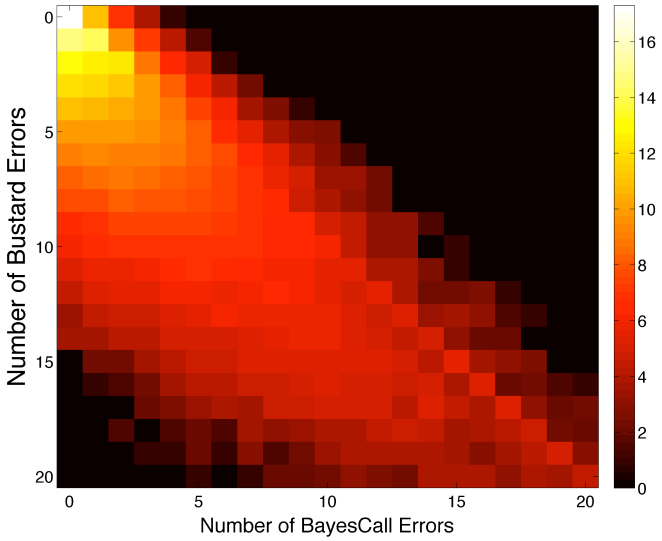


Figure 2.7: **Heat plot of joint errors in Bustard and BayesCall for the 76-cycle PhiX174 data.** This plot depicts the joint error matrix shown in Table 2.3. The (x, y) entry in the plot corresponds to $\log_2$ of the number of reads with x errors in Bustard and y errors in BayesCall. This plot clearly illustrates that BayesCall generally produces sequence reads with substantially fewer errors than that produced by Bustard.

Table 2.4: Gain in accuracy from different levels of modeling extra residual effects.

Method	By-Base Error Rate	By-Read Error Rate
Bustard	0.01737	0.44431
BayesCall ($\alpha_t = 0$ for all t)	0.01076	0.31328
BayesCall ($\alpha_t =$ some α for all t)	0.00965	0.28748
BayesCall (cycle-dependent α_t)	0.00783	0.24398

In BayesCall, extra residual effects are captured by the parameter α_t. We used cycle-dependent parameters $d_t, \sigma_t^2, \boldsymbol{X}_t$, and $\boldsymbol{\Sigma}_t$ in all three cases of BayesCall. Even without modeling extra residual effects (i.e., with $\alpha_t = 0$ for all t), BayesCall achieved an improvement of 38% over Bustard in the by-base error rate. When the full model with cycle-dependent α_t was used, the improvement over Bustard increased to 55%.

effect (i.e., assume α_t is equal to some constant α for all t). (3) Cycle-dependent residual effect (i.e., α_t allowed to change over cycles). For this study, we considered a single tile from the 76-cycle data set; the resulting test set contained $68,272$ reads and $5,188,672$ bases. Parameter estimation was performed for each case allowing $d_t, \sigma_t^2, \boldsymbol{X}_t$, and $\boldsymbol{\Sigma}_t$ to be cycle-dependent.

Results from this study are shown in Table 2.4. Even without modeling extra residual effects (i.e., with $\alpha_t = 0$ for all t), BayesCall achieved an improvement of 38% (respectively, 29%) over Bustard in the by-base (respectively, by-read) error rate. This gain in accuracy illustrates the utility of using cycle-dependent parameters $d_t, \sigma_t^2, \boldsymbol{X}_t$, and $\boldsymbol{\Sigma}_t$. BayesCall's improvement over Bustard increased slightly when a model with a constant residual effect was used. When the full model with cycle-dependent α_t was used, the improvement over Bustard further increased to 55% in the by-base error rate and 45% in the by-read error rate.

2.5.8 Discrimination ability of quality scores

To compare the utility of quality scores, we define the discrimination ability $D(\epsilon)$ at error tolerance ϵ as follows: Sort the bases according to their quality scores, from the highest to the lowest. Then, go down that sorted list until the error rate surpasses ϵ. The number of correctly called bases up to that point corresponds to $D(\epsilon)$. This notion is essentially the same as the discrimination ability defined in [13]. Shown in Figure 2.8 are the plots of $D(\epsilon)$ for the data sets considered in this chapter. We see that BayesCall not only reduces the error rate, but also produces base-specific quality scores with a high discrimination ability that consistently outperforms both Bustard's and Alta-Cyclic's.

An error tolerance ϵ implies a corresponding quality score cutoff for each basecaller;

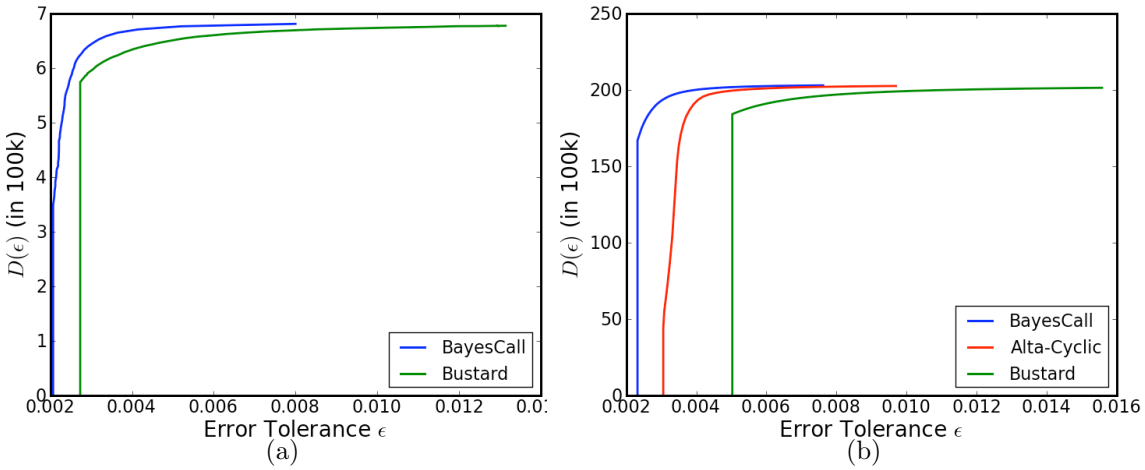


Figure 2.8: **Discrimination ability $D(\epsilon)$ of quality scores at error tolerance ϵ.** We define $D(\epsilon)$ as the number of correctly called bases at error tolerance ϵ. BayesCall maintains a high discrimination ability which outperforms both Bustard's and Alta-Cyclic's. (a) Results for the 36-cycle PhiX174 data set. (b) Results for the 76-cycle PhiX174 data set.

bases with quality scores below the cutoff may be considered unreliable and get thrown out. This means that for all reasonable error tolerances, BayesCall produces a much larger number of bases and consequently a lower cost per base. This is a much stronger result than simply having a lower error rate at a single cutoff.

2.5.9 Difference between GA-I and GA-II

The known upgrades between the GA-I and GA-II platforms include an imaging system with a wider field of view, a larger flow cell, and a larger CCD. Additional Peltier devices were employed to regulate the flow cell temperature better throughout the sequencing process. Further, a slight modification to the flow cell design also reduced the number of poorly imaged tiles caused by displaced fluid from the oil immersion microscope.

The sequencing chemistry was improved, resulting in a noticeable decrease in the phasing rate p. This change is illustrated in Table 2.5, which shows our estimates of the phasing and prephasing rates for the PhiX174 data on GA-I and GA-II. As a consequence of the chemistry improvement, the rate d_t also decreased in magnitude and in the amount of fluctuation. However, we still observed a marked residual effect (parametrized by α_t), as illustrated in Figure 2.3(b). Residual effects contribute significantly to the cycle-to-cycle decrease in the platform's signal-to-noise ratio. To

Table 2.5: Average estimates of phasing and prephasing rates for the PhiX174 data on GA-I and GA-II.

Platform	Phasing rate p	Prephasing rate q
GA-I	6.8×10^{-4}	3.4×10^{-3}
GA-II	3.0×10^{-8}	3.3×10^{-3}

Because of improvement in the sequencing chemistry, the phasing rate p for GA-II is substantially lower than that for GA-I. However, prephasing and residual effects seem to persist in GA-II.

the extent we can better model it, we can obtain longer reads for a given error tolerance.

2.6 The naiveBayesCall Algorithm

We now describe the naiveBayesCall algorithm. As mentioned in Chapter 2.1, it is based on the same graphical model as in BayesCall, and we employ the method detailed in Chapter 2.4.3 to estimate the parameters in the model. The main novelty of naiveBayesCall is in the base-calling part of the method. We divide the presentation into two parts. First, we propose a hybrid algorithm that combines the model described in Chapter 2.4 with the matrix inversion approach employed in Bustard described in Chapter 2.3. Then, we use the hybrid algorithm to initialize an optimization procedure that both improves the base-call accuracy and produces useful per-base quality scores.

2.6.1 A hybrid base-calling algorithm

We present a new inference algorithm for the model described in Chapter 2.4. The main strategy is to avoid direct inference of the continuous random variables $\Lambda_{t,k}$. First, for each cycle t, we estimate the average concentration c_t of templates within each tile. In Chapter 2.5, we showed that the magnitude of the fluctuation rate d_t (c.f., (2.2)) is typically very small (less than 0.03) for all $1 \leq t \leq L$. Hence, assuming that d_t is close to zero for all t, we estimate the tile-wide average concentration c_t using

$$c_t = \frac{1}{K} \sum_{k=1}^{K} \sum_{b=1}^{4} \max(0, [\boldsymbol{X}_t^{-1}(\boldsymbol{I}_{t,k} - \alpha_t \boldsymbol{I}_{t-1,k})]_b), \tag{2.17}$$

where K denotes the total number of clusters in the tile and $[\boldsymbol{y}]_b$ denotes the bth component of vector $\boldsymbol{y}$. The above c_t serves as our estimate of $\Lambda_{t,k}$ for all clusters k within the same tile. Using this estimate, we define

$$\tilde{\boldsymbol{I}}_{t,k} = \left(\boldsymbol{I}_{t,k} - \alpha_t \frac{c_t}{c_{t-1}} \boldsymbol{I}_{t-1,k} \right)_+, \tag{2.18}$$

where $(\boldsymbol{y})_+$ denotes the vector obtained from $\boldsymbol{y}$ by replacing all negative components with zeros. Note that subtracting $\alpha_t \frac{c_t}{c_{t-1}} \boldsymbol{I}_{t-1,k}$ from $\boldsymbol{I}_{t,k}$ accounts for the residual effect modeled in (2.8). The ratio $\frac{c_t}{c_{t-1}}$ rescales $\boldsymbol{I}_{t-1,k}$ so that its norm is similar to that of $\boldsymbol{I}_{t,k}$.

After determining $\tilde{\boldsymbol{I}}_{t,k}$, the rest of the hybrid base-calling algorithm resembles Bustard. First, for each cycle t, we estimate the cluster-specific normalized concentration of four different bases using

$$\boldsymbol{\mathcal{Z}}_{t,k} = (\mathcal{Z}^A_{t,k}, \mathcal{Z}^C_{t,k}, \mathcal{Z}^G_{t,k}, \mathcal{Z}^T_{t,k})' = \frac{1}{c_t} (\boldsymbol{X}_t^{-1} \tilde{\boldsymbol{I}}_{t,k})_+, \tag{2.19}$$

where $\boldsymbol{X}_t$ is the 4-by-4 crosstalk matrix at cycle t (see previous section). Normalizing by the tile-wide average c_t is to make the total concentration stay roughly the same across all cycles. Note that $\boldsymbol{\mathcal{Z}}_{t,k}$ is an estimate of the concentration vector shown in (2.4). Now, we let $\boldsymbol{\mathcal{Z}}_k = (\boldsymbol{\mathcal{Z}}_{1,k}, \ldots, \boldsymbol{\mathcal{Z}}_{L,k})$ and use the following formula to correct for phasing and prephasing effects:

$$\boldsymbol{\mathcal{Z}}_k (\boldsymbol{Q}^w)^{-1}, \tag{2.20}$$

where $\boldsymbol{Q}^w = (\boldsymbol{Q}^w_1, \ldots, \boldsymbol{Q}^w_L)$ is the L-by-L phasing-prephasing matrix defined in Chapter 2.4. Finally, for $t = 1, \ldots, L$, the row index of the largest value in column t of (2.20) is called as the tth base of the DNA templates in cluster k:

$$\boldsymbol{S}^H_{t,k} = \underset{b \in \{A,C,G,T\}}{\operatorname{argmax}} \, [\boldsymbol{\mathcal{Z}}_k (\boldsymbol{Q}^w)^{-1}]_{b,t}. \tag{2.21}$$

Algorithm 1 summarizes the hybrid base-calling algorithm just described.

The performance of the hybrid algorithm will be discussed in Chapter 2.5. We will see that, with the parameters estimated in BayesCall, our simple hybrid algorithm already outperforms Bustard.

Algorithm 1 Hybrid Algorithm

for all tiles **do**
  for all cycles $1 \leq t \leq L$ **do**
    Estimate concentration c_t for each cycle t according to (2.17).
  end for
  for all clusters $1 \leq k \leq K$ **do**
    Compute residual-corrected intensities $\tilde{\boldsymbol{I}}_k$ using (2.18).
    Compute concentration matrix $\boldsymbol{\mathcal{Z}}_k$ according to (2.19).
    Correct for phasing and prephasing effect using (2.20).
    Infer $\boldsymbol{S}_{1,k}^H, \ldots, \boldsymbol{S}_{L,k}^H$ using (2.21) and output the associated sequence.
  end for
end for

2.6.2 Estimating Λ_k via optimization and computing quality scores

In this section, we devise a method to improve the hybrid algorithm described above and to compute base-specific quality score. The Viterbi algorithm [49] has been widely adopted as a dynamic programming algorithm to find the most probable path of states in a hidden Markov Model. There are two source of difficulty in applying the Viterbi algorithm to our problem:

1. Our model is a high order Markov model, so path tracing can be computationally expensive. This complexity arises from modeling phasing and prephasing effects. Recall that the observation probability at a given cycle t depends on all hidden random variables $\boldsymbol{S}_{i,k}$ with i within a window w about t. In [26], we used 11 for the window size.

2. In addition to the discrete random variables $\boldsymbol{S}_k = (\boldsymbol{S}_{1,k}, \ldots, \boldsymbol{S}_{L,k})$ for the DNA sequence, our model contains continuous hidden random variables $\boldsymbol{\Lambda}_k = (\Lambda_{1,k}, \ldots, \Lambda_{L,k})$, but the Viterbi algorithm cannot handle continuous variables. One might try to address this problem by marginalizing out $\boldsymbol{\Lambda}_k$, but it turns out that the maximum a posteriori (MAP) estimate of $\boldsymbol{\Lambda}_k$ is useful for computing quality scores.

To address the first problem, we obtain a good initial guess of hidden variables $\boldsymbol{S}_k$ and use it to break the high order dependency. To cope with the second problem, we adopt a sequential approach. Algorithm 2 summarizes our naiveBayesCall algorithm and a detailed description is provided below.

Our algorithm iteratively estimates $\Lambda_{t,k}$ and updates $\boldsymbol{S}_{t,k}$, starting with $t = 1$ and ending at $t = L$. Let $\boldsymbol{S}_k^{(i)}$ denote the sequence matrix after the ith iteration.

Algorithm 2 naiveBayesCall Algorithm

for all clusters k **do**
  Initialize $\boldsymbol{S}_k^{(0)} = (\boldsymbol{S}_{1,k}^H, \ldots, \boldsymbol{S}_{L,k}^H)$ using Algorithm 1.
  for $1 \leq t \leq L$ **do**
    for $b \in \{A, C, G, T\}$ **do**
      Find $\lambda_{t,k}^b = \text{argmax}_\lambda \, L_{t,k}^b(\lambda)$, where $L_{t,k}^b(\lambda)$ is defined as in (2.22).
      Compute base-specific quality score $Q(b)$ using (2.26) and (2.27).
    end for
    Set $s_{t,k} = \text{argmax}_{b \in \{A,C,G,T\}} \, L_{t,k}^b(\lambda_{t,k}^b)$.
    Update $\boldsymbol{S}_k^{(t)} = \mathcal{R}_{t,s_{t,k}}(\boldsymbol{S}_k^{(t-1)})$.
  end for
  Call $s_{1,k}, \ldots, s_{L,k}$ as the inferred sequence and output base-specific quality scores.
end for

We initialize $\boldsymbol{S}_k^{(0)} = \boldsymbol{S}_k^H$, where $\boldsymbol{S}_k^H = (\boldsymbol{S}_{1,k}^H, \ldots, \boldsymbol{S}_{L,k}^H)$ is obtained using the hybrid algorithm described in Chapter 2.6.1. Let $s_{t,k}$ denote the base (i.e., A, C, G, or T) called by naiveBayesCall for position t of the DNA sequence in cluster k. At iteration t, the first $t-1$ bases $s_{1,k}, \ldots, s_{t-1,k}$ have been called and the vectors $\boldsymbol{S}_{1,k}, \ldots, \boldsymbol{S}_{t-1,k}$ have been updated accordingly. The following procedures are performed at iteration t:

Optimization

Our inference of $\Lambda_{t,k}$ depends on the base at position t, which has not been called yet. We use $\lambda_{t,k}^b$ to denote the inferred value of $\Lambda_{t,k}$, given that the base at position t is b. For a given base $b \in \{A, C, G, T\}$, we define the log-likelihood function

$$L_{t,k}^b(\lambda) = \begin{cases} \log \mathbb{P}[\boldsymbol{I}_{t,k} | \boldsymbol{I}_{t-1,k}, \mathcal{R}_{t,b}(\boldsymbol{S}_k^{(t-1)}), \lambda], & \text{if } t = 1, \\ \log \mathbb{P}[\lambda | \lambda_{t-1,k}^{s_{t-1,k}}] + \log \mathbb{P}[\boldsymbol{I}_{t,k} | \boldsymbol{I}_{t-1,k}, \mathcal{R}_{t,b}(\boldsymbol{S}_k^{(t-1)}), \lambda], & \text{if } t > 1, \end{cases} \tag{2.22}$$

where $\mathcal{R}_{t,b}(\boldsymbol{S}_k^{(t-1)})$ denotes the sequence matrix obtained by replacing column t of $\boldsymbol{S}_k^{(t-1)}$ with the unit column vector $\boldsymbol{e}_b$, the probability $\mathbb{P}[\lambda | \lambda_{t-1,k}^{s_{t-1,k}}]$ is defined in (2.6), and observation likelihood $\mathbb{P}[\boldsymbol{I}_{t,k} | \boldsymbol{I}_{t-1,k}, \mathcal{R}_{t,b}(\boldsymbol{S}_k^{(t-1)}), \lambda]$ is defined by (2.5)–(2.7), More exactly,

$$\mathbb{P}[\boldsymbol{I}_{t,k} | \boldsymbol{I}_{t-1,k}, \mathcal{R}_{t,b}(\boldsymbol{S}_k^{(t-1)}), \lambda] \approx \phi(\boldsymbol{I}_{t,k}; \lambda \boldsymbol{X}_t \boldsymbol{z}_{t,k}^{w,b} + \alpha_t (1 - d_t) \boldsymbol{I}_{t-1,k}, \|\lambda \boldsymbol{z}_{t,k}^{w,b}\|^2 \boldsymbol{\Sigma}_t), \tag{2.23}$$

where $\boldsymbol{z}_{t,k}^{w,b} = \mathcal{R}_{t,b}(\boldsymbol{S}_k^{(t-1)}) \boldsymbol{Q}_t^w$ is an unscaled concentration vector and $\phi(\cdot; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the probability density function of a multivariate normal distribution with mean vector

$\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. For each $b \in \{A, C, G, T\}$, we estimate $\lambda_{t,k}^b$ using the following optimization:

$$\lambda_{t,k}^b = \underset{\lambda}{\operatorname{argmax}}\, L_{t,k}^b(\lambda). \tag{2.24}$$

Our implementation of naiveBayesCall uses the golden section search method [27] to solve the 1-dimension optimization problem in (2.24).

Base-calling

The nucleotide at position t is called as

$$s_{t,k} = \underset{b\in\{A,C,G,T\}}{\operatorname{argmax}} \max_{\lambda} L_{t,k}^b(\lambda) = \underset{b\in\{A,C,G,T\}}{\operatorname{argmax}} L_{t,k}^b(\lambda_{t,k}^b), \tag{2.25}$$

and the sequence matrix is updated accordingly: $\boldsymbol{S}_k^{(t)} = \mathcal{R}_{t,s_{t,k}}(\boldsymbol{S}_k^{(t-1)})$.

Quality score

For position t, the probability of observing b is estimated by

$$\mathbb{P}(b) = \frac{\phi(\boldsymbol{I}_{t,k}; \lambda_{t,k}^b \boldsymbol{X}_t \boldsymbol{z}_{t,k}^{w,b} + \alpha_t(1-d_t)\boldsymbol{I}_{t-1,k}, \|\lambda_{t,k}^b \boldsymbol{z}_{t,k}^{w,b}\|^2 \boldsymbol{\Sigma}_t)}{\sum_{x\in\{A,C,G,T\}} \phi(\boldsymbol{I}_{t,k}; \lambda_{t,k}^x \boldsymbol{X}_t \boldsymbol{z}_{t,k}^{w,x} + \alpha_t(1-d_t)\boldsymbol{I}_{t-1,k}, \|\lambda_{t,k}^x \boldsymbol{z}_{t,k}^{w,x}\|^2 \boldsymbol{\Sigma}_t)}, \tag{2.26}$$

and the quality score for base b is given by

$$Q(b) = 10 \log_{10} \left[\frac{\mathbb{P}(b)}{1-\mathbb{P}(b)} \right]. \tag{2.27}$$

2.7 Empirical Studies of the naiveBayescall Algorithm

2.7.1 Improvement in running time

The experiments were done on a Mac Pro with two quad-core 3.0GHz Intel Xeon processors, utilizing all eight cores. Table 2.6(a) shows the training time and the prediction time of Alta-Cyclic, BayesCall, and naiveBayesCall. The times reported in Table 2.6(a) are for the full-lane of data. The training time of naiveBayesCall is the same as that of BayesCall, since naiveBayesCall currently uses the same parameter estimation method as in BayesCall. Although the training time of BayesCall is longer than that of Alta-Cyclic, we point out that, in principle, the cycle-dependent

Table 2.6: Comparison of overall performance results (a) Running times (in hours). (b) Base-call error rates.

(a)

	Training Time	Testing Time for Full-Lane
Alta-Cyclic	10	4.4
BayesCall	19	362.5
naiveBayesCall	19	6

(b)

	4 Tiles		Full-Lane	
	By-base	By-read	By-base	By-read
Bustard	0.0098	0.2656	0.0103	0.2705
Alta-Cyclic	0.0097	0.3115	0.0101	0.3150
BayesCall	0.0076	0.2319	NA	NA
naiveBayesCall	0.0080	0.2348	0.0088	0.2499

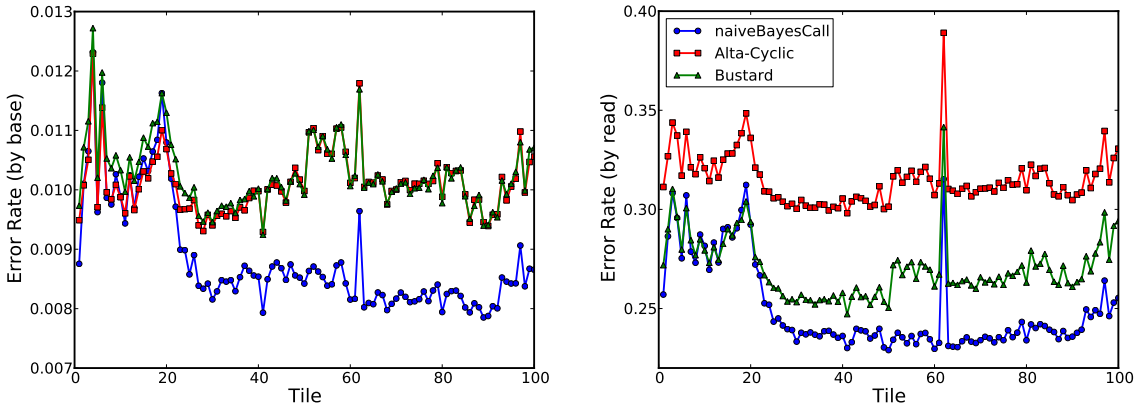
BayesCall's testing time was estimated from that for 4 tiles of data. The "by-base" error rate refers to the ratio of the number of miscalled bases to the total number of base-calls made, while the "by-read" error rate refers to the ratio of the number of reads each with at least one miscalled base to the total number of reads considered.

parameters in BayesCall can be estimated progressively as the sequencing machine runs (a run currently takes about 10 days). This advantage comes from the fact that BayesCall can be trained without labeled training data. As Table 2.6(a) illustrates, naiveBayesCall dramatically improves the base-calling time over BayesCall, delivering about 60X speedup. This improvement makes our model-based base-calling approach practical.

2.7.2 Summary of base-call accuracy

Table 2.6(b) shows the overall base-call accuracy of the four different methods. The columns under the label "4 Tiles" show the results for only 4 out of the 100 tiles in the lane. Since it would take more than 15 days for BayesCall to call bases for the entire lane, it was not used in the full-lane study. Both Bustard and Alta-Cyclic were trained on the full-lane data. To train BayesCall for the 4-tile data, we randomly chose 250 clusters from each tile to estimate tile-specific parameters, and used the same parameters in naiveBayesCall. To run naiveBayesCall on the full-lane data, we randomly chose 250 clusters from the entire lane to estimate lane-wide parameters.

From Table 2.6(b), we see that the performance of naiveBayesCall is comparable



(a) (b)

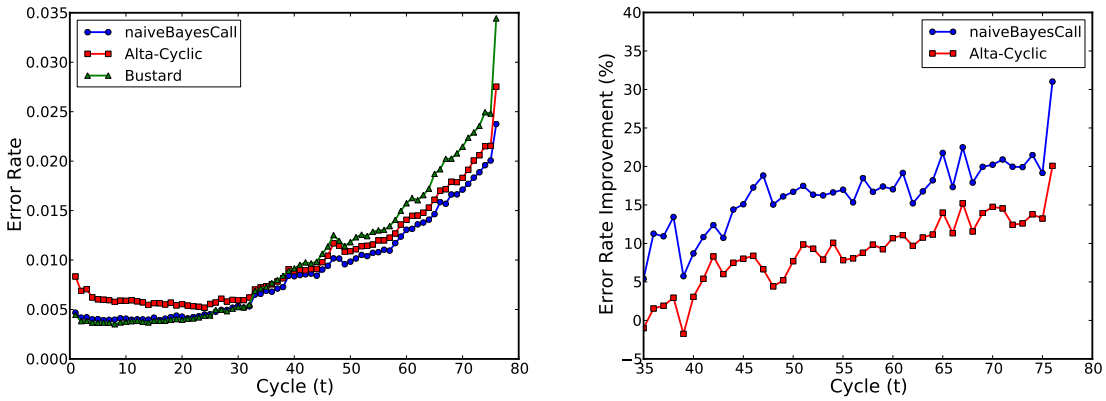
Figure 2.9: **Title-specific error rates.** Our algorithm naiveBayesCall clearly outperforms both Bustard and Alta-Cyclic for tiles 21 to 100, but has comparable error rates for tiles 1 to 20. It is possible to improve naiveBayesCall's base-call accuracy for the first 20 tiles by using tile-specific parameter estimates. (a) By-base error rate for each tile. (b) By-read error rate for each tile.

to that of BayesCall. Figure 2.9 shows the tile-specific average error rate for each tile of the full-lane data. Note that naiveBayesCall clearly outperforms both Bustard and Alta-Cyclic for tiles 21 to 100, but has comparable error rates for tiles 1 to 20. It is possible to improve naiveBayesCall's accuracy for the first 20 tiles by using tile-specific parameter estimates (see Discussion).

Figure 2.10(a) illustrates the cycle-specific average error rate. Note that naiveBayesCall's average accuracy dominates Alta-Cyclic's for all cycles. Furthermore, the improvement of naiveBayesCall over Bustard increases with cycles, as illustrated in Figure 2.10(b). This suggests that it is possible to run the sequencing machine for longer cycles and still obtain useful sequence information for longer reads by using an improved base-calling algorithm such as ours. Furthermore, we believe that fewer errors in later cycles may facilitate *de novo* assembly. We return to this point in Chapter 2.7.4.

2.7.3 Discrimination ability of quality scores

To compare the utility of quality scores, we follow the idea in [13] and define the discrimination ability $D(\epsilon)$ at error tolerance ϵ as follows. First sort the called bases according to their quality scores in decreasing order. Then go down that sorted list until the error rate surpasses ϵ. The number of correctly called bases up to this



(a) (b)

Figure 2.10: **Comparison of average base-call accuracy for the full-lane data.** Note that naiveBayesCall's average accuracy dominates Alta-Cyclic's for all cycles. Further, The improvement of naiveBayesCall over Bustard increases with cycles. (a) Cycle-specific error rate. (b) Improvement of naiveBayesCall and Alta-Cyclic in cycle-specific error rate over Bustard.

point is defined as $D(\epsilon)$. Hence, $D(\epsilon)$ corresponds to the number of bases that can be correctly called at error tolerance ϵ, if we use quality scores to discriminate bases with lower error probabilities from those with higher error probabilities. For any given ϵ, a good quality score should have a high $D(\epsilon)$. Shown in Figure 2.11 is a plot of $D(\epsilon)$ for naiveBayesCall, Alta-Cyclic, and Bustard. As the figure shows, naiveBayesCall's quality score consistently outperforms Alta-Cyclic's. For $\epsilon < 0.0017$ and $\epsilon > 0.0032$, naiveBayesCall's quality score has a higher discrimination ability than Bustard's, while the opposite is true for the intermediate values $0.0017 < \epsilon < 0.0032$.

2.7.4 Effect of base-calling accuracy on the performance of *de novo* assembly

Here, we demonstrate how improved base-calling accuracy may facilitate *de novo* assembly. Because of the short read length and high sequencing error rate, *de novo* assembly of the next-generation sequencing data is a challenging task. Recently, several promising algorithms [7, 35, 54, 48, 5] have been proposed to tackle this problem. In our study, we used the program Velvet [54] to perform *de novo* assembly of the reads called by different base-calling algorithms. First, we randomly chose a set of clusters from the 4-tile data without doing any filtering. Then, we base-called those clusters using each of Bustard, Alta-Cyclic, BayesCall, and naiveBayesCall, producing four different sets of base-calls on the same data set. For each set of base-called

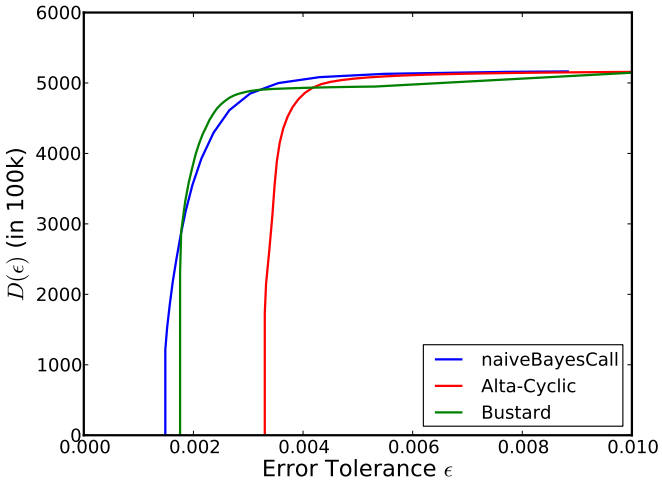


Figure 2.11: **Discrimination ability $D(\epsilon)$ of quality scores for the full-lane data.** Roughly, $D(\epsilon)$ corresponds to the number of correctly called bases at error tolerance ϵ.

reads, Velvet was run with the k-mer length set to 55. For a given choice of coverage, we repeated this experiment 100 times. The results are summarized in Table 2.7, which shows the N50 length, the maximum contig length, and the total number of contigs produced; these numbers were averaged over the 100 experiments. On average naiveBayesCall led to better *de novo* assemblies than did Bustard or Alta-Cyclic: For 5X and 10X coverages, the performance of naiveBayesCall was similar to that of Bustard's in terms of the N50 and maximum contig lengths, but naiveBayesCall produced significantly more contigs than did Bustard. For 15X and 20X, naiveBayesCall clearly outperformed Bustard in all measures, producing longer and more contigs. The results for BayesCall and naiveBayesCall were comparable.

2.7.5 Effect of base-calling accuracy on SNP detection

In this section, we assess how the improvement in base-calling accuracy affects the performance of single nucleotide polymorphism (SNP) detection. SNP detection is usually achieved by mapping short-reads onto a reference genome and by noting the difference between the reads and the reference. In the absence of sequencing errors, all mismatches would correspond to SNPs. When there are sequencing errors, however, SNP detection algorithms need to distinguish between sequencing errors and SNPs. We show below that reducing base-calling errors can significantly improve SNP detection quality.

In our experiment, we used a standard resequencing data of PhiX174 virus, pro-

Table 2.7: Average contig lengths resulting from *de novo* assembly of the 76-cycle PhiX174 data, when different base-calling algorithms are used to produce the input short-reads.

	Bustard			Alta-Cyclic			BayesCall			naiveBayesCall		
Cov	N50	Max	#Ctgs	N50	Max	#Ctgs	N50	Max	#Ctgs	N50	Max	#Ctgs
5X	145	153	277	140	146	251	146	156	358	146	158	349
10X	203	368	2315	200	353	2148	203	368	2435	203	365	2467
15X	352	685	4119	331	637	4047	368	712	4249	371	716	4263
20X	675	1162	4941	674	1119	4893	752	1246	5004	750	1259	5015

The length of the PhiX174 genome is 5386 bp. Velvet [54] was used to perform *de novo* assembly for varying sequence coverage depths (denoted "Cov"). N50 is a statistic commonly used to assess the quality of *de novo* assembly. It is computed by sorting all contigs by their size in decreasing order and adding the length of these contigs until the sum is greater than 50% of the total length of all contigs. The length of the last added contig is reported as N50. A larger N50 indicates a better assembly. Also shown are the maximum contig length (denoted "Max") and the total number of contigs (denoted "#Ctgs").

vided to us by the DPGP Sequencing Lab at UC Davis. The data were obtained from a 36-cycle run on the Genome Analyzer I platform. The same data set was also used in [26]. We produced four different sets of base-calls using Bustard, Alta-Cyclic, BayesCall, and naiveBayesCall.

We artificially introduced SNPs into the PhiX174 reference genome by using the "fakemut" function of MAQ [31]. We tried using 0.001 and 0.01 for SNP rates. MAQ was also used to perform read mapping and SNP-calling. We supplied MAQ with the artificially mutated reference genome, the short-reads called by each base-calling algorithm, and their associated quality scores. The detected SNP positions were then compared to the positions of artificially generated SNPs. To evaluate the quality of SNP detection, we considered precision and recall of the detected SNPs. Precisions and recalls are widely used metrics in accessing the quality of information retrieval tasks. In this context, precision is the portion of true SNPs in the set of all called SNPs, and recall is the portion of called SNPs in the set of all true SNPs introduced into the reference genome.

Table 2.8 summarizes the precision and recall of SNP detection with reads called by the four base-calling algorithms. In the unfiltered case, the reads called by BayesCall and naiveBayesCall led to significant improvements in precision, while generally leading to a similar level of recall as that for the reads called by Bustard. Precision could be further improved by imposing a threshold on read mapping quality scores when

using MAQ to call SNPs. Shown in Table 2.8 are precision and recall results under the commonly-used quality threshold of 30. Overall, our results illustrate that, when the sequence coverage depth is low (< 10), performing error correction prior to using MAQ can significantly improve the quality of SNP detection.

Table 2.8: SNP detection efficiency.

		SNP rate = 0.001				SNP rate = 0.01			
		Unfiltered		Quality ≥ 30		Unfiltered		Quality ≥ 30	
Cov	Reads	Prec.	Recall	Prec.	Recall	Prec.	Recall	Prec.	Recall
2	Bustard	0.17	0.62	0.44	0.50	0.54	0.64	0.84	0.58
	Alta-Cyclic	0.28	0.62	0.31	0.62	0.71	0.64	0.72	0.62
	BayesCall	0.33	0.62	0.45	0.62	0.78	0.62	0.84	0.60
	naiveBayesCall	0.31	0.62	0.33	0.62	0.76	0.62	0.77	0.60
4	Bustard	0.26	1.00	0.54	0.88	0.62	0.80	0.85	0.73
	Alta-Cyclic	0.33	0.88	0.37	0.88	0.75	0.80	0.77	0.80
	BayesCall	0.33	0.88	0.54	0.88	0.76	0.78	0.85	0.78
	naiveBayesCall	0.35	0.88	0.39	0.88	0.76	0.78	0.80	0.78
6	Bustard	0.33	1.00	0.57	1.00	0.73	0.89	0.87	0.87
	Alta-Cyclic	0.42	1.00	0.50	1.00	0.80	0.89	0.83	0.87
	BayesCall	0.44	1.00	0.62	1.00	0.82	0.89	0.89	0.89
	naiveBayesCall	0.42	1.00	0.50	1.00	0.80	0.89	0.85	0.89
8	Bustard	0.40	1.00	0.62	1.00	0.78	0.93	0.89	0.89
	Alta-Cyclic	0.47	1.00	0.53	1.00	0.84	0.93	0.86	0.93
	BayesCall	0.53	1.00	0.62	1.00	0.86	0.93	0.89	0.93
	naiveBayesCall	0.50	1.00	0.57	1.00	0.84	0.93	0.88	0.93
10	Bustard	0.50	1.00	0.62	1.00	0.84	0.96	0.89	0.91
	Alta-Cyclic	0.50	1.00	0.57	1.00	0.86	0.96	0.88	0.96
	BayesCall	0.53	1.00	0.62	1.00	0.86	0.96	0.90	0.96
	naiveBayesCall	0.53	1.00	0.57	1.00	0.86	0.96	0.88	0.96
15	Bustard	0.53	1.00	0.62	1.00	0.86	0.98	0.90	0.96
	Alta-Cyclic	0.57	1.00	0.57	1.00	0.88	0.98	0.88	0.98
	BayesCall	0.62	1.00	0.62	1.00	0.90	0.98	0.90	0.98
	naiveBayesCall	0.62	1.00	0.62	1.00	0.90	0.98	0.90	0.98

MAQ [31] was used to carry out read mapping and SNP detection. Precision (denoted "Prec." in the table) is the portion of true SNPs in the set of all called SNPs, and recall is the portion of called SNPs in the set of all true SNPs introduced to the reference genome. "Quality ≥ 30" corresponds to the results obtained by imposing a threshold of 30 on read mapping quality scores when using MAQ to call SNPs. These results indicate that performing error correction prior to using MAQ can significantly improve precision of SNP detection when the sequence coverage depth is low.

Chapter 3

A Reference-Free Error Correction Algorithm for Short Reads

3.1 Introduction

To improve data quality for next generation sequencing, it is well recognized that development of accurate, scalable computational tools must parallel the rapid advancement in sequencing technology. As mentioned in Chapter 1, there are two main approaches to addressing this challenge: 1) One approach is to develop improved image analysis and base-calling algorithms. This line of work has been pursued by several researchers in the past, including ourselves; see [50, 12, 41, 26, 24, 28]. Indeed, by employing more sophisticated statistical methods, it has been demonstrated that it is possible to deliver significant improvements over the tools developed by the manufacturers of the sequencing platforms. 2) An alternative approach is to correct for potential errors after base-calling has been performed by leveraging on the fact that each position in the genome on average is sequenced multiple times. Several methods have been developed for this approach; e.g., see [15, 7, 44, 45, 8, 42, 40, 51, 52].

The goal of the present chapter is to introduce a novel, efficient computational method for the latter approach. Our error correction algorithm is called ECHO, and it has the following notable features, most of which are unique to our method:

1. Whereas previous reference-free error correction algorithms [7, 44, 45, 42] typically rely on the user to specify some of the key parameters, of which optimal values are typically unknown *a priori*, ECHO automatically finds the optimal parameters and estimates error characteristics specific to each sequencing run.

2. ECHO is based on a probabilistic framework and can assign a quality score to each corrected base. Quality scores are useful and sometimes necessary in

downstream analysis.

3. Previous reference-free error correction algorithms [7, 44, 45, 42] are based on k-mer or substring frequencies. ECHO instead relies on finding overlaps between reads. Despite being more computationally intensive, this approach retains more of the information contained in the reads without aggregating away potentially useful relationships. Although read overlap error correction has been done previously with Sanger reads [2], ECHO is specialized for high-throughput short reads.

4. ECHO explicitly models heterozygosity in diploid genomes and allows one to process diploid data in a novel way. Specifically, for each length-l haplotype (or short-read) in the input data, ECHO can infer a likely length-l genotype sequence from which the haplotype may have originated, and position-specific quality scores can be assigned to each genotype. For each input short-read, this approach provides a way, without the help of a reference genome, to detect the bases that originated from heterozygous sites in the sequenced diploid genome and to infer the corresponding genotypes.

ECHO performs error correction by first finding overlaps between reads. This stage of the algorithm requires not only computational considerations to handle the enormous number of reads typically generated by NGS platforms, but requires an effective method to filter out false overlaps. ECHO achieves this by imposing a maximum error tolerance in the overlaps and a minimum overlap length. These stringent requirements typically filter out many of the potential false overlaps, assuming the genome is not repeat-rich. The optimal error tolerance and overlap requirement, however, are usually not known *a priori*. Whereas previous error correction algorithms require the user to specify key parameters, which may greatly affect the performance of the algorithm, ECHO automatically determines the optimal values for the error tolerance and minimum overlap length by utilizing assumptions on the coverage distribution of the reads.

Once ECHO determines these parameters and finds the overlaps between reads, it estimates the error characteristics of the data using an expectation maximization procedure. Statistically modeling the error behavior on a per-base basis more accurately considers the error behavior of the sequencing platform. This is in contrast to other error correction algorithms that typically use an error threshold or coverage cutoff, which may not be statistically motivated. Furthermore, in ECHO, error modeling is performed for each input data set to characterize the unique properties of different sequencing runs more effectively. Finally, ECHO utilizes the read overlaps and the

error characterization to perform error correction on the reads using a maximum *a posteriori* procedure. Because the sequencing behavior is modeled statistically, this leads directly to a method for assigning meaningful quality scores to the bases of the corrected reads.

Repeats in the genome are challenging to handle since they may lead to false overlaps. In general, ECHO is less sensitive to repeats than are previous error correction methods that rely on k-mer or substring frequencies. By retaining the relationships among reads more explicitly, rather than computing aggregate statistics on substring frequencies, ECHO is better able to handle repeat patterns in the genome. However, the obstacles imposed by repeat regions are still significant, and more sophisticated methods that incorporate additional information will be necessary to overcome these challenges adequately.

To test the performance of our algorithm, we consider real short-read data generated by Illumina's Genome Analyzer (GA) I and II, as well as synthetic short-read data simulated to mimic GA's error characteristics. For GA, most of the sequencing errors are miscall errors (as opposed to indels), with the property that the error rate generally increases toward the end of the read. We show that ECHO is able to improve the accuracy of previous error correction methods by several folds to an order of magnitude, depending on the sequence coverage depth and the position in the read. In particular, provided that the sequence coverage depth is moderate to high (roughly, 15 or higher), ECHO remains effective throughout the entire read length, typically reducing the error rate at the end of the read from over 5% to under 1%. ECHO can process relatively small genomes such as that of yeasts or fungi on a desktop computer. To demonstrate this point and to show that ECHO is capable of coping with non-uniform coverage that may arise in real data, we apply our method on a whole-genome yeast data set.

In addition to the improvement in data quality, we also examine here the effects of error correction on *de novo* assembly. Assembly is particularly challenging when the sequence coverage depth is low to moderate, and it may benefit considerably from the improved data quality provided by error correction. In this chapter, we show that performing error correction as a preprocessing step facilitates *de novo* assembly significantly.

3.2 Notation

Throughout, we adopt the notational convention described below. Denote the genome being sequenced by S and $\overline{S}$, where the latter is the reverse complement of the former. For ease of discussion, we assume that S and $\overline{S}$ are length-L strings over

$\{A, C, G, T\}$; if there is more than one chromosome, then S and $\overline{S}$ correspond to the strings obtained by concatenating the chromosomes.

We assume that all reads have the same length, denoted by l. We use N to denote the total number of reads in the input data, and define the sequence coverage depth as $c = \frac{Nl}{L}$. The set of all reads obtained from a sequencing run is denoted by $\boldsymbol{R} = \{r^1, \ldots, r^N\}$, with r^i, for $1 \leq i \leq N$, corresponding to a length-l string over $\{A, C, G, T\}$. In general, two reads r^i and r^j in $\boldsymbol{R}$ may be identical, and therefore $\boldsymbol{R}$ actually is a multiset. Note that a read r^i may be have originated from either S or $\overline{S}$. With $\bar{r}^i$ denoting the reverse complement of r^i, we define $\overline{\boldsymbol{R}} = \{\bar{r}^1, \ldots, \bar{r}^N\}$ and augment the set of reads as $\mathcal{R} = \boldsymbol{R} \bigcup \overline{\boldsymbol{R}}$. Given a read $r \in \mathcal{R}$, let r_m denote the mth character of r and, for $1 \leq m \leq n \leq l$, let $r_{m:n}$ denote the substring of r starting at the mth character and ending at the nth character, inclusive; similar notation is used for S and $\overline{S}$. By a k-mer, we mean a length-k string over $\{A, C, G, T\}$. Given a k-mer K, for $1 \leq k \leq l$, and a read r, we write $K \subset r$ to denote that K is a substring of r. Finally, we use $\{A, C, G, T\}$ and $\{1, 2, 3, 4\}$ interchangeably as indices.

3.3 The ECHO Algorithm

In this section, we describe our error-correction algorithm ECHO, which is divided into two stages: "neighbor finding" and "maximum *a posteriori* error correction." Since we assume that the reference genome is not available, we first devise a clustering algorithm to find "neighboring" reads that presumably cover the same region of the genome. After the neighbors of each read are found, we then correct for sequencing errors by employing a maximum *a posteriori* estimation procedure for each base of the read.

3.3.1 Neighbor finding

In a sequencing run with a moderate to high coverage depth, each position in the genome on average gets sequenced multiple times. The basic idea behind error correction is to leverage on this redundant information. When there is a reference genome, identifying the reads that cover the same region can be done via "read mapping" [31, 30], which involves aligning reads against the reference genome. However, in the absence of a reference genome, the task of identifying neighbors becomes more challenging. Below, we devise a neighbor-finding algorithm based on hashing. The key observation we utilize is that, for a small enough k, given a pair of reads that cover the same region of a reasonable size, the chance that they do not have any common k-mer is small, even if the reads have a few sequencing errors.

For a given positive integer $k < l$, let $\mathcal{K}$ denote the set of all distinct k-mers present in $\mathcal{R}$; i.e.,

$$\mathcal{K} = \{k\text{-mer } K \mid K \subset r \text{ for some } r \in \mathcal{R}\}.$$

For each $K \in \mathcal{K}$, let $C(K) \subset \mathcal{R}$ denote the set of reads that *contain* the k-mer K as a substring. Then, given a read $r \in C(K)$ that contains K, we define the starting position of K as

$$x_K(r) = \begin{cases} \min\{m \mid r_{m:m+k-1} = K\}, & \text{if } r \in \boldsymbol{R}, \\ \max\{m \mid r_{m:m+k-1} = K\}, & \text{if } r \in \overline{\boldsymbol{R}}. \end{cases} \tag{3.1}$$

The intuition behind (3.1) is that the quality of sequencing is usually better in the early cycles of a sequencing run, and we want to utilize those good data. The maps x_K, for all $K \in \mathcal{K}$, can be constructed in $O(Nl)$ time by iterating through the reads and using hashing.

Note that the reads in $C(K)$ should have a good chance of covering the same region in genome. To refine this idea further and select those reads which may have originated from the same region, we perform pairwise alignments involving the reads in $C(K)$, for each $K \in \mathcal{K}$. In the most general case, a pairwise alignment can be done with the Smith-Waterman algorithm [47], resulting in the computational cost of $O(l^2)$ per alignment. However, for a sufficiently large k, if indel errors are rare, as is true in the case of the Illumina platform, it is possible to obtain a reasonable alignment in $O(1)$ time by using the position map x_K. Further, the quality of that alignment can be determined in $O(l)$ time. We adopt this simplified approach in what follows.

Given two reads $r, s \in C(K)$, we align $K \subset r$ with $K \subset s$ as shown in Figure 3.1, so that position $x_K(r)$ in r is aligned with position $x_K(s)$ in s. In this alignment, position m in r gets identified with position $I_K^{r \to s}(m)$ in s, where

$$I_K^{r \to s}(m) = \begin{cases} m - x_K(r) + x_K(s), & \text{if } 1 \le m - x_K(r) + x_K(s) \le l, \\ \varnothing, & \text{otherwise.} \end{cases} \tag{3.2}$$

The length $\omega_K(r, s)$ of overlap between r and s is given by

$$\omega_K(r, s) = l - \max(x_K(r), x_K(s)) + \min(x_K(r), x_K(s)). \tag{3.3}$$

Let $\Omega_K(r, s)$ denote the set of position indices in r that overlap with s in the above alignment. Note that $|\Omega_K(r, s)| = \omega_K(r, s)$. The quality of alignment between r and

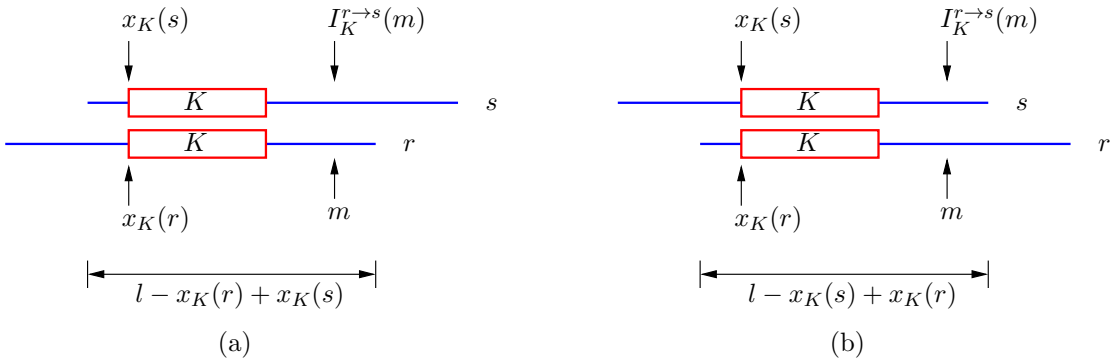


Figure 3.1: **Illustration of aligning two length-l reads r and s by identifying the common k-mer K.** Position m of r in the overlapping region gets identified with position $I_K^{r\to s}(m) = m - x_K(r) + x_K(s)$ of s. (a) A case with $x_K(r) > x_K(s)$. (b) A case with $x_K(r) < x_K(s)$.

s is measured by the Hamming distance

$$d_K(r,s) = \sum_{m \in \Omega_K(r,s)} \mathbf{1}\{r_m \neq s_{I_K^{r\to s}(m)}\}, \tag{3.4}$$

where $\mathbf{1}\{y\}$ denotes the indicator variable taking value 1 if y holds and 0 otherwise.

For a given minimum overlap threshold δ and maximum error rate tolerance ϵ, we say that two reads r, s are "neighbors" of each other if $\omega_K(r,s) \geq \delta$ and $d_K(r,s) \leq \epsilon \cdot \omega_K(r,s)$ for some $K \in \mathcal{K}$; i.e., r and s overlap by a substantially large amount and they do not differ by much in that overlapping region. We write $r \sim s$ to denote that r and s are neighbors, and use $\mathcal{N}_r = \{s \in \mathcal{R} \mid s \sim r\}$ to denote the set of all neighbors of r. Note that a neighbor $s \in \mathcal{N}_r$ may belong to either $\boldsymbol{R}$ or $\overline{\boldsymbol{R}}$. Further, $r \sim r$ and $r \in \mathcal{N}_r$ by convention.

Note that if a pair of reads, r and s, contain more than one common k-mer, then they may admit more than one alignment. In such a case, we use the alignment associated with the k-mer $K_{r,s}^*$ that gives the minimum Hamming distance normalized by the overlap length; i.e.,

$$K_{r,s}^* = \operatorname*{argmin}_{K \in \mathcal{K}} \left\{ \frac{d_K(r,s)}{\omega_K(r,s)} \right\}. \tag{3.5}$$

Then, we use

$$\Omega(r,s) = \Omega_{K_{r,s}^*}(r,s) \tag{3.6}$$

to denote the set of positions in r that overlap with s in the alignment associated

Algorithm 3 Neighbor-Finding Algorithm

Given k, δ, ϵ.
Construct $\mathcal{K}_0$ by extracting all possible k-mers from $\mathcal{R}$.
for all reads $r \in \mathcal{R}$ and $K \in \mathcal{K}_0$ **do**
  Construct $x_K(r)$ according to (3.1).
end for
for all k-mer $K \in \mathcal{K}_0$ **do**
  for all reads $r, s \in C(K)$ **do**
    if $\omega_K(r,s) \geq \delta$ and $d_K(r,s) \leq \epsilon \cdot \omega_K(r,s)$ **then**
      Update $\mathcal{N}_r$ and $\mathcal{N}_s$, and update $I_{r\to s}$ and $I_{s\to r}$ according to (3.5) and (3.7).
    end if
  end for
end for
Output the neighbor sets $(\mathcal{N}_r)_{r\in\mathcal{R}}$ and the identification maps $(I_{r\to s})_{r\in\mathcal{R},s\in\mathcal{N}_r}$

with $K^*_{r,s}$, and position $m \in \Omega(r,s)$ gets identified with the following position in s:

$$I_{r\to s}(m) = I^{r\to s}_{K^*_{r,s}}(m), \tag{3.7}$$

where the identification map $I^{r\to s}_{K^*_{r,s}}$ is defined as in (3.2). Using the algorithm described above, one can construct $(I_{r\to s})_{r\in\mathcal{R},s\in\mathcal{N}_r}$ in $O(Nl + |\mathcal{K}| \cdot l \cdot \max_K |C(K)|^2)$ time. In practice, the size $|C(K)|$ can be very large for some $K \in \mathcal{K}$, potentially increasing the running time substantially; this usually happens when K is a substring of a repetitive region in the genome being sequenced. Therefore, instead of considering all possible k-mers, we consider only those k-mers with $|C(K)| \leq 50000$, and define

$$\mathcal{K}_0 = \left\{ K \in \mathcal{K} \,\middle|\, |C(K)| \leq 50000 \right\}.$$

Our neighbor-finding algorithm is summarized in Algorithm 3.

3.3.2 Parameter selection

We now describe how the parameters k, δ and ϵ are selected. The choice of k should consider the following two competing factors: 1) The length k should not be too small, so that $\max_K |C(K)|$ is linear in the coverage depth c, independent of the length L of the genome. Otherwise, our algorithm will take more than $O(Nl+|\mathcal{K}_0|lc^2)$ time to construct the identification maps $(I_{r\to s})_{r\in\mathcal{R},s\in\mathcal{N}_r}$. 2) On the other hand, k should not be too large, so that, even with sequencing errors, the reads that cover the same region can still have a common k-mer and be grouped as neighbors by our neighbor-finding algorithm. The number of distinct k-mers in a genome of size L is

upper bounded by $2(L-k+1)$. However, sequencing errors may introduce additional distinct k-mers. Hence, if the order of magnitude L of the genome size is known, we suggest using

$$k = \lfloor \log_4 10L \rfloor. \tag{3.8}$$

If the genome size is completely unknown, then, assuming that the sequence coverage depth ranges from tens to hundreds, we approximate (3.8) with

$$k = \lfloor \log_4 Nl \rfloor. \tag{3.9}$$

We show in Chapter 3.6 that the performance of our algorithm is robust with respect to the choice of k.

The quality of the identification maps $(I_{r\to s})_{r\in\mathcal{R},s\in\mathcal{N}_r}$ greatly depends on the choice of δ and ϵ. If we use δ and ϵ that impose too strict constraints for the neighbor relation, then the reads that cover the same region of the genome will not be grouped together, and the subsequent error-correction algorithm (described later) will lose its power to correct for sequencing errors. Furthermore, constraints that are too loose will lead to incorrectly inferring reads as neighbors when they do not in fact cover the same region of genome, and hence negatively affect the accuracy of our error-correction algorithm.

To select (δ, ϵ) automatically, we randomly choose a subset $\boldsymbol{Q} \subseteq \boldsymbol{R}$ and define $\mathcal{Q} = \boldsymbol{Q} \cup \overline{\boldsymbol{Q}}$, where $\overline{\boldsymbol{Q}}$ contains the reverse complements of the reads in $\boldsymbol{Q}$; we suggest using a subset of size $100,000$ or $|\boldsymbol{R}|$, whichever is smaller. Then, we construct the neighbor sets $\mathcal{N}_r$ and the identification maps $I_{r\to s}$ for the reads $r \in \mathcal{Q}$. We assume that, for any position m in read $r \in \mathcal{Q}$, the number $C_{r,m}$ of neighbors of r that overlap with the mth character r_m is Poisson distributed. We further assume that $C_{r,m}$, for $r \in \mathcal{Q}$ and $1 \le m \le l$, are independent and identically distributed. Therefore, when Algorithm 3 is used with parameters (δ, ϵ), the empirical probability mass function for the distribution of $C_{r,m}$ can be obtained by

$$\hat{f}_{\delta,\epsilon}(i) = \frac{1}{l|\mathcal{Q}|} \sum_{r\in\mathcal{Q}} \sum_{1\le m\le l} \mathbf{1}\{i = c_{r,m}(\delta,\epsilon)\}, \tag{3.10}$$

where $c_{r,m}(\delta, \epsilon)$ is the empirical number of neighbors in $\mathcal{N}_r$ that overlap with the mth character r_m. Our goal is then to find the best (δ, ϵ) such that $\hat{f}_{\delta,\epsilon}$ resembles a Poisson probability mass function as closely as possible. The mean of the Poisson distribution can be estimated by a maximum *a posteriori* estimator. However, we find that it is

more robust to match the mode and use the following as the mean:

$$\mu(\delta, \epsilon) = \underset{i>1}{\operatorname{argmax}}\, \hat{f}_{\delta,\epsilon}(i) + 1. \tag{3.11}$$

Finally, we apply a grid search algorithm to obtain

$$(\delta^*, \epsilon^*) = \underset{\delta,\epsilon}{\operatorname{argmin}}\, d_{TV}(\hat{f}_{\delta,\epsilon}, g_{\mu(\delta,\epsilon)}), \tag{3.12}$$

where $g_{\mu(\delta,\epsilon)}$ denotes the probability mass function of a Poisson distributed random variable with mean $\mu(\delta, \epsilon)$, and $d_{TV}(f, g)$ denotes the total variation distance

$$d_{TV}(f, g) = \frac{1}{2} \sum_i |f(i) - g(i)|. \tag{3.13}$$

The above parameter selection method can be implemented efficiently by noting that one can obtain the neighbor relationships under more restrict constraints from that under looser constraints. More precisely, if we have already computed $(\mathcal{N}_r)_{r\in\mathcal{Q}}$ and $(I_{r\to s})_{r\in\mathcal{Q},s\in\mathcal{N}_r}$ for some δ and ϵ, then $(\mathcal{N}_r)_{r\in\mathcal{Q}}$ and $(I_{r\to s})_{r\in\mathcal{Q},s\in\mathcal{N}_r}$ for $\delta' \geq \delta$ and $\epsilon' \leq \epsilon$ can be obtained by re-examining the reads in $(\mathcal{N}_r)_{r\in\mathcal{Q}}$ for (δ, ϵ), instead of re-running the entire algorithm for (δ', ϵ') from scratch.

3.3.3 Maximum *a posteriori* error correction for haploid genomes

Maximum *a posteriori* error correction

We now describe our error-correction algorithm, which is applied to the neighbors $(\mathcal{N}_r)_{r\in\mathcal{R}}$ and identification maps $(I_{r\to s})_{r\in\mathcal{R},s\in\mathcal{N}_r}$ found using $(\delta, \epsilon) = (\delta^*, \epsilon^*)$ in Algorithm 3, where δ^* and ϵ^* are as shown in (3.12). The basic idea is to formulate a maximum *a posteriori* estimation procedure for each base in each read and correct for possible sequencing errors.

In the Illumina GA platform, most of the sequencing errors are miscall errors (as opposed to indels), with the property that the error rate generally increases towards the end of the read. For simplicity, we assume that miscall errors are made independently within each read and across different reads. To be more precise, we assume that each character in a given read is distributed as a multinomial distribution with parameters that depend on the position of that character within the read and the true corresponding nucleotide in the original genome. If the true nucleotide at position m of a given read in $\boldsymbol{R}$ is b, the probability that it is called as b' is denoted by $\Phi_{b,b'}^{(m)}$.

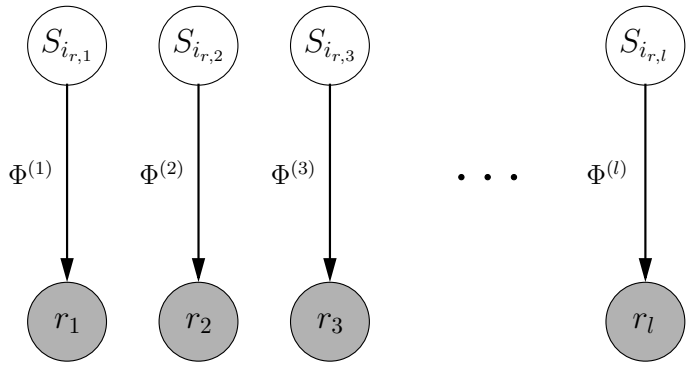


Figure 3.2: **A graphical representation of our model (3.16)–(3.18) for generating a read r from the genome S.** Open circles correspond to unobserved random variables, while filled circles represent observed variables. The symbol $\mathbf{\Phi}^{(m)}$ denotes the *confusion matrix* for position $1 \leq m \leq l$; this matrix characterizes sequencing errors. See text for details.

Following [26, 24], we refer to $\mathbf{\Phi}^{(m)} = (\Phi^{(m)}_{b,b'})_{b,b' \in \{A,C,G,T\}}$ as the *confusion matrix* for position $1 \leq m \leq l$; this matrix characterizes sequencing errors. We later provide an algorithm for estimating $\mathbf{\Phi}^{(m)}$ for $1 \leq m \leq l$.

Now, given a read $r \in \boldsymbol{R}$, suppose that the nucleotide at position m of r originated from position $i_{r,m}$ of the genome. To be explicit, suppose that S, rather than $\overline{S}$, is the source. Then, in the absence of any sequencing error, read r will be identical to the substring $S_{i_{r,1}:i_{r,l}}$. For a read $r \in \boldsymbol{R}$, the entry $\Phi^{(m)}_{b,b'}$ in the confusion matrix $\mathbf{\Phi}^{(m)}$ can be formally defined as

$$\Phi^{(m)}_{b,b'} = \mathbb{P}(r_m = b' \mid S_{i_{r,m}} = b). \tag{3.14}$$

For a read $r \in \overline{\boldsymbol{R}}$, since it is the reverse complement of a read in $\boldsymbol{R}$, the error characteristic at position m of r is captured by

$$\overline{\Phi}^{(m)}_{b,b'} = \Phi^{(l-m+1)}_{\bar{b},\bar{b}'}, \tag{3.15}$$

where $\bar{b}$ and $\bar{b}'$ are the complementary bases of b and b', respectively.

We use the following model for the distribution of observed reads:

$$\begin{aligned}
S_i &\sim \text{Unif}\{A, C, G, T\}, && (3.16)\\
r_m | S_{i_{r,m}} = b &\sim \text{Mult}(1; \Phi^{(m)}_{b,A}, \Phi^{(m)}_{b,C}, \Phi^{(m)}_{b,G}, \Phi^{(m)}_{b,T}), \quad \text{if } r \in \boldsymbol{R}, && (3.17)\\
r_m | S_{i_{r,m}} = b &\sim \text{Mult}(1; \overline{\Phi}^{(m)}_{b,A}, \overline{\Phi}^{(m)}_{b,C}, \overline{\Phi}^{(m)}_{b,G}, \overline{\Phi}^{(m)}_{b,T}), \quad \text{if } r \in \overline{\boldsymbol{R}}. && (3.18)
\end{aligned}$$

A graphical representation of this model is shown in Figure 3.2 for the case $r \in \boldsymbol{R}$. Based on this model, we can express the posterior distribution $\mathbb{P}(S_{i_{r,m}} = b \mid r_m = b')$, for $b, b' \in \{A, C, G, T\}$, as

$$\mathbb{P}(S_{i_{r,m}} = b \mid r_m = b') = \begin{cases} \dfrac{\Phi^{(m)}_{b,b'}}{\sum_{a \in \{A,C,G,T\}} \Phi^{(m)}_{a,b'}}, & \text{if } r \in \boldsymbol{R}, \\ \dfrac{\overline{\Phi}^{(m)}_{b,b'}}{\sum_{a \in \{A,C,G,T\}} \overline{\Phi}^{(m)}_{a,b'}}, & \text{if } r \in \overline{\boldsymbol{R}}, \end{cases} \tag{3.19}$$

where $\overline{\Phi}$ is as defined in (3.15). For a read $r \in \boldsymbol{R}$, we assume that every neighbor in $\mathcal{N}_r$ is distributed according to (3.16)–(3.18). Hence, assuming that the neighbor reads in $\mathcal{N}_r$ cover the same region in the original genome, the posterior log likelihood of $S_{i_{r,m}}$ being base b is

$$\ell_{r,m}(b) = \sum_{s \in \mathcal{N}_r} \mathbf{1}\{m \in \Omega(r, s)\} \times \log \mathbb{P}(S_{i_{r,m}} = b \mid r_m = s_{I_{r \to s}(m)}) \tag{3.20}$$

where recall that $\Omega(r, s)$ is the set of positions in r that overlap with s, and $I_{r \to s}(m)$ is the position in s with which position m in r is identified. (Also recall that r itself is in $\mathcal{N}_r$, and that its neighbor $s \neq r$ may belong to either $\boldsymbol{R}$ or $\overline{\boldsymbol{R}}$.) The maximum *a posteriori* (MAP) estimate of $S_{i_{r,m}}$ is

$$S^{\text{MAP}}_{i_{r,m}} = \operatorname*{argmax}_{b \in \{A,C,G,T\}} \ell_{r,m}(b). \tag{3.21}$$

Coverage checks

An abnormally large set of overlapping reads usually indicates the reads are sampled from a repetitive or repeat region, and suggests that our assumptions on overlapping reads most likely do not hold. From the overlaps found in the neighbor finding stage, we have an estimate of the expected coverage μ, as described in Chapter 3.3.2. If the coverage at a position is greater than $\mu + \alpha\sqrt{\mu}$ (α standard deviations greater than the average coverage), we do *not* perform a correction. Our empirical study suggests that the precise value of α does not generally affect the accuracy of the algorithm; $4 \leq \alpha \leq 6$ is typically effective. This "filtering" allows ECHO to be robust against parts of the genome that have repetitive structure or have highly non-uniform coverage. We refer to this filtering as a "coverage check," because it ensures that the estimated coverage for a read looks reasonable before correction is attempted.

Algorithm 4 Maximum *A Posteriori* Error-Correction Algorithm (Haploid Version)

Given $(\mathcal{N}_r)_{r\in\mathcal{R}}, (I_{r\to s})_{r\in\mathcal{R},s\in\mathcal{N}_r}$, and $\mathbf{\Phi}^{(m)}$, for $1 \le m \le l$.
for all reads $r \in \mathcal{R}$ **do**
  $\ell_{r,m}(b) \leftarrow 0, \forall b \in \{A, C, G, T\}$ and $\forall 1 \le m \le l$
  for all $1 \le m \le l$ **do**
    for all reads $s \in \mathcal{N}_r$ **do**
      Update posterior log likelihood via (3.20)
    end for
    Correct for sequencing errors with MAP estimator via (3.21).
    Update r_m^{ECHO} via (3.22).
  end for
  Output corrected read r^{ECHO}.
end for

Recall that, in (3.10), $c_{r,m}$ denotes the number of neighbor reads in $\mathcal{N}_r$ that overlap with the mth position in r. We want to check whether $c_{r,m}$ is consistent with the expected value μ obtained using (3.11) and (3.12). If the difference $c_{r,m} - \mu$ is too large, it indicates a potential problem with neighbor-finding, and hence the MAP error correction might be unreliable. Therefore, in such a case we output the original base r_m instead of the MAP estimate $S_{i_{r,m}}^{\text{MAP}}$. Following the same Poisson distribution assumption as in Chapter 3.3.2 and choosing $[1, \mu + 5\sqrt{\mu}]$ as the region of acceptance, the base output by our algorithm is

$$r_m^{\text{ECHO}} = \begin{cases} S_{i_{r,m}}^{\text{MAP}}, & \text{if } c_{r,m} \in [1, \mu + 5\sqrt{\mu}], \\ r_m, & \text{otherwise.} \end{cases} \tag{3.22}$$

A summary of the MAP error-correction algorithm can be found in Algorithm 4.

3.3.4 Quality scores

Providing an accurate measure of per-base quality has practical importance. For instance, MAQ [31], a widely-used read mapping algorithm, utilizes base-specific quality scores to produce mapping quality scores and to call variants. A widely adopted definition of quality score is that used in *Phred* [13]; it is a transformed error probability of a base, defined more precisely as follows: Let $e_{r,m}$ denote the probability that the mth base r_m of read r is erroneously called. Then, the Phred quality score $q_{r,m}$ for r_m is given by

$$q_{r,m} = -10 \log_{10} e_{r,m}. \tag{3.23}$$

Since ECHO provides an estimate of the posterior probability of each base, the error probability $e_{r,m}$ of base r_m^{ECHO} can be estimated by

$$e_{r,m} = 1 - \frac{\exp\{\ell_{r,m}(r_m^{\text{ECHO}})\}}{\sum_{b\in\{A,C,G,T\}} \exp\{\ell_{r,m}(b)\}}, \tag{3.24}$$

where $\ell_{r,m}(b)$ is defined in (3.20). The Phred quality score for r_m^{ECHO} is then obtained by plugging (3.24) into (3.23). In Chapter 3.6, we show that, combined with this quality score, our error correction method can significantly improve SNP detection.

3.4 Generalization to diploid genomes

In this section, we show how our approach can be generalized to handle short-reads from diploid genomes, in which case the main challenge lies in distinguishing heterozygotes from sequencing errors. We use the same neighbor-finding algorithm as in the haploid case, but modify the maximum *a posteriori* error-correction step as described below.

The unordered genotype at a given position in a diploid genome is either a homozygote or a heterozygote. We use $\text{Hom} = \{\{A,A\},\{C,C\},\{G,G\},\{T,T\}\}$ to denote the set of homozygous genotypes and $\text{Het} = \{\{A,C\},\{A,G\},\{A,T\},\{C,G\},\{C,T\},\{G,T\}\}$ to denote the set of heterozygous genotypes. The set of all possible genotypes is denoted by $\mathcal{G} = \text{Hom} \cup \text{Het}$.

Given a length-l haplotype sequence $r \in \{A,C,G,T\}^l$, let $D_{i_{r,1}:i_{r,l}}$ denote the true unphased genotype sequence in the diploid genome from which r originated. Our goal is to transform r into a length-l unphased genotype sequence $\mathcal{D}(r) \in \mathcal{G}^l$, such that $\mathcal{D}(r)$ is as close to $D_{i_{r,1}:i_{r,l}}$ as possible. For each position m in $r \in \boldsymbol{R}$, we assume that the two alleles in $D_{i_{r,m}}$ are equally likely to be the source of the mth nucleotide r_m. Hence, sequencing errors at that position are characterized by the following generalized confusion matrix $\boldsymbol{\Psi}^{(m)} = (\Psi^{(m)}_{g,b})_{g\in\mathcal{G},b\in\{A,C,G,T\}}$:

$$\Psi^{(m)}_{\{a,a'\},b} = \mathbb{P}(r_m = b \mid D_{i_{r,m}} = \{a,a'\}) = \frac{1}{2}(\Phi^{(m)}_{a,b} + \Phi^{(m)}_{a',b}),$$

where Φ is defined in (3.14). Similarly, for a read $r \in \overline{\boldsymbol{R}}$, the error characteristic at position m of r is captured by

$$\overline{\Psi}^{(m)}_{\{a,a'\},b} = \frac{1}{2}(\overline{\Phi}^{(m)}_{a,b} + \overline{\Phi}^{(m)}_{a',b}),$$

where $\overline{\Phi}^{(m)}$ is defined in (3.15).

Now, we use h to denote the prior probability that a given site in the diploid genome is heterozygous. Then, for a particular genotype $g \in \mathcal{G}$, the prior is

$$\mathbb{P}(g) = \begin{cases} \dfrac{1-h}{4}, & \text{if } g \in \text{Hom}, \\ \dfrac{h}{6}, & \text{if } g \in \text{Het}. \end{cases}$$

That is, given the zygosity, we assume the uniform distribution over all possible genotypes. Analogous to (3.19), the posterior distribution is given by

$$\mathbb{P}(D_{i_{r,m}} = g \mid r_m = b) = \begin{cases} \dfrac{\Psi^{(m)}_{g,b}\mathbb{P}(g)}{\sum_{g' \in \mathcal{G}} \Psi^{(m)}_{g',b}\mathbb{P}(g')}, & \text{if } r \in \boldsymbol{R}, \\ \dfrac{\overline{\Psi}^{(m)}_{g,b}\mathbb{P}(g)}{\sum_{g' \in \mathcal{G}} \overline{\Psi}^{(m)}_{g',b}\mathbb{P}(g')}, & \text{if } r \in \overline{\boldsymbol{R}}. \end{cases}$$

Further, analogous to (3.20), the posterior log likelihood of $D_{i_{r,m}}$ being genotype g is

$$\mathcal{L}_{r,m}(g) = \sum_{s \in \mathcal{N}_r} \mathbf{1}\{m \in \Omega(r,s)\} \times \log \mathbb{P}(D_{i_{r,m}} = g \mid r_m = s_{I_{r \to s}(m)}),$$

and the maximum *a posteriori* estimate of $D_{i_{r,m}}$ is

$$D^{\text{MAP}}_{i_{r,m}} = \operatorname*{argmax}_{g \in \mathcal{G}} \mathcal{L}_{r,m}(g).$$

Finally, analogous to (3.22), our algorithm outputs the following genotype for the mth position in r:

$$\mathcal{D}^{\text{ECHO}}(r_m) = \begin{cases} D^{\text{MAP}}_{i_{r,m}}, & \text{if } c_{r,m} \in [1, \mu + 5\sqrt{\mu}], \\ \{r_m, r_m\}, & \text{otherwise.} \end{cases}$$

Replacing the error probability (3.24) with

$$e_{r,m} = 1 - \frac{\exp\{\mathcal{L}_{r,m}(\mathcal{D}^{\text{ECHO}}(r_m))\}}{\sum_{g \in \mathcal{G}} \exp\{\mathcal{L}_{r,m}(g)\}},$$

the same formula (3.23) can be used for the quality score of $\mathcal{D}^{\text{ECHO}}(r_m)$. Incidentally, note that the above algorithm for the diploid case with $h = 0$ reduces to the algorithm

described earlier for the haploid case.

3.5 Estimating position-specific confusion matrices

If a reference genome is available, the confusion matrices $\mathbf{\Phi}^{(m)}$, for $1 \leq m \leq l$, may be estimated by first aligning reads to the reference genome, treating mismatches as errors. However, this estimation can be biased, since mismatches may also originate from single nucleotide polymorphism or other structural variations. Another method of estimating the confusion matrices is to use a control lane that sequences a known genome. This method avoids the potential biases caused by variations between the sampled genome and the reference genome, but this solution is costly and error characteristics can still differ between different lanes or runs.

Based on our probabilistic model (3.16)–(3.18), we adopt the expectation-maximization (EM) algorithm to estimate confusion matrices. The advantage of the EM algorithm is that estimation can be performed without a reference genome. To start the estimation of $\mathbf{\Phi}^{(m)}$, for $1 \leq m \leq l$, we first obtain $(\mathcal{N}_r)_{r\in\mathcal{R}}$ and $(I_{r\to s})_{r\in\mathcal{R},s\in\mathcal{N}_r}$, with k, δ, ϵ as described in Chapter 3.3.2. Then we fix the neighbor relationship and apply the EM algorithm. For all $1 \leq m \leq l$, we initialize the confusion matrix as

$$\mathbf{\Phi}^{(m)}_{b,b'} = \begin{cases} 0.99, & \text{if } b = b', \\ 0.01/3, & \text{otherwise.} \end{cases} \tag{3.25}$$

The EM algorithm typically terminates after a few iterations.

3.6 Results

In this section, we evaluate the performance of our algorithm ECHO and compare it to previous error correction methods. In particular, we consider the Spectral Alignment (SA) algorithm used in the preprocessing step of the *de novo* assembler EULER-USR [7]. This error correction algorithm is based on examining the k-mer spectrum of the input reads [39, 6]. Because SA is sensitive to the choice of k, for each experiment, we ran SA over a range of k and selected the k that generated the optimal results (this was typically around $k = 15$).

SHREC [44] is a recent error correction algorithm based on a generalized suffix trie data structure and essentially considers the substring spectrum of the input reads, i.e. a range of k-mers where k is over a given interval. It requires the user to specify

the size of the genome, and we provided it with the true genome size for each data set we considered. Further, SHREC has a parameter called "strictness," with the default value being 7. For coverage depths $c < 40$, SHREC failed to run with the default parameter value, so we used the largest value less than 7 that the program accepted. As discussed in Chapter 3.3, ECHO automatically chooses its parameters k, ω^*, and ϵ^*.

In addition to the improvement in data quality, we also examine the effects of error correction on *de novo* assembly.

3.6.1 Data and experiment setup

We tested the performance of our method on both real and simulated data, with a range of sequence coverage depths. Specifically, the following data types were considered:

D1 (PhiX174): We used standard resequencing data of PhiX174 virus, provided to us by the DPGP Sequencing Lab at UC Davis. The data were obtained from the Illumina GA-II platform, with the viral sample in a single lane of the flow cell. The lane consisted of 100 tiles, containing a total of 14,820,478 reads, each being 76 bp in length. The length of the PhiX174 genome is 5386 bp, and the short-read data corresponds to a coverage depth of around 210,000. Various coverage depths were obtained by sampling without replacement from the actual data set. We used Illumina's standard base-calling software, Bustard, to produce base-calls. To create labeled test data, we aligned the reads against the PhiX174 reference sequence and regarded every mismatch as a sequencing error.

D2 (*D. mel.*): Another real data set we used is that of *Drosophila melanogaster* inbred line RAL-399, sequenced and assembled by the Drosophila Population Genomics Project (DPGP, `http://www.dpgp.org/`). The short-reads were downloaded from the NCBI Sequence Read Archive (`http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi`). The data consisted of 36 bp reads sequenced on the Illumina GA-I platform and base-called by Bustard. The overall coverage depth of this data set was roughly 13. To create a labeled test set, we aligned the reads against the genome assembly (release 1.0) produced by DPGP. Then, we selected those reads that mapped to chromosome 2L and regarded mismatches as sequencing errors.

D3 (Simulated *D. mel.*): In the test described above in D2, errors in the assembled genome (i.e., differences between the assembled genome and the true genome)

may confound the assessment of the accuracy of error-correction algorithms. To avoid this potential problem, we used the following procedure to simulate a set of 76 bp reads from a known sequence: We took a 100 kbp region from chromosome 2L of the RAL-399 inbred line mentioned above. Let S denote that sequence and $\overline{S}$ its reverse complement. To simulate each 76 bp read, we chose a strand (either S or $\overline{S}$) and a starting position uniformly at random, and introduced errors by selecting a random "error template" as follows: First, we selected a read randomly from the PhiX174 data set D1 and aligned it to the PhiX174 reference genome to determine the positions of errors. Then, we took that error template and added errors to the simulated Drosophila read at the corresponding locations. The miscalled base was determined according to the position-specific empirical confusion matrices estimated from the PhiX174 data, because the "error template" from the sampled PhiX174 read only provides the *positions* of errors. This approach allowed us to simulate more realistic error patterns; e.g., the correlation of errors in a read and the clustering of errors on a subset of the reads. Our simulation method permitted miscall errors but no indels.

D4 (Simulated human data with repeats and duplicated regions): To evaluate the performance of ECHO on sequences with repeats and duplicated regions, we generated 76 bp reads from a 250 kbp region of human chromosome 16 known to have a duplicated gene [34]. More specifically, the region spans chromosomal coordinates $20240000 - 20490000$ of the GRCh37 reference assembly of chromosome 16, and was downloaded from the NCBI Genome database. We followed the same procedure as in D3 to introduce errors into reads randomly selected from the 250 kbp region.

D5 (Simulated diploid data): To generate a known diploid genome, we took the 100 kbp region mentioned in D3 and mutated each base with probability 0.001, with the new base drawn uniformly at random from the alternative bases. To simulate each 76 bp read, we first chose a strand uniformly at random from the four possible strands. Then, we randomly chose a starting position and generated observed reads according to (3.17)–(3.18), using position-specific empirical confusion matrices estimated from the PhiX174 data. For this diploid data set, the sequence coverage depth is given by $\frac{Nl}{2L}$, where N denotes the number of reads, $l = 76$, and $L = 100,000$.

D6 (*Saccharomyces cerevisiae*): This real data set consisted of 7.1 million reads from the whole-genome sequencing of a laboratory-evolved yeast strain derived

from DBY11331 [16], Accession Number SRR031259 in the NCBI Sequence Read Archive. Because the yeast strain differed slightly from the S. cerevisiae reference genome, we created a new reference sequence using MAQ and the standard reference sequence. We used this modified reference genome to evaluate error-correction performance in our experiments. The data set contained single-end 36 bp reads from the Illumina GA II platform, covering all 16 chromosomes and the mitochondrial DNA of *S. cerevisiae*. The overall coverage depth of the data set was roughly 21.

3.6.2 Error correction accuracy for haploid genomes

Here, we compare the error correction accuracy of SA [7], SHREC [44], and ECHO on the data sets described in the previous section. The output of SHREC is partitioned into "corrected" and "discarded" reads. We measured the performance of SHREC by considering only the corrected reads.

In what follows, we use the term *by-base* error rate to refer to the ratio of the total number of incorrect bases in the reads to the total combined length of the reads. We also consider the *by-read* error rate, which corresponds to the ratio of the number of reads each with *at least* one miscalled base to the total number of reads. Table 3.1 shows the by-base and by-read error rates for haploid data sets D1–D4 before and after running the three error-correction algorithms. Table 3.2 shows the numbers of corrected errors and introduced errors, as well as the *gain*, for the same data sets after error correction. The gain is defined as the number of corrected errors minus the number of introduced errors, divided by the number of actual errors, in the same manner as in [52]. On all data sets, ECHO substantially outperformed both SA and SHREC. In general, SA did not seem effective in reducing the by-base error rate. SHREC was more effective than SA, while our algorithm outperformed SHREC by several folds on real data and by up to two orders of magnitude on simulated data. Even in the case of low coverage depth (e.g., 5), ECHO was quite effective in reducing the sequencing error rate. Observe that all three algorithms generally improved the by-read error rates noticeably, with ECHO outperforming both SA and SHREC significantly. It is interesting that while SA only moderately improved the by-base error rate, it was actually quite effective in improving the by-read error rate.

To compare the performance of the three error-correction algorithms in more detail, we examined the *position-specific* by-base error rates; i.e., the by-base error rate at position i is the ratio of the number of reads with an incorrect base at position i to the total number of reads. For PhiX174 data D1 with sequence coverage depth 30, Figure 3.3 illustrates the position-specific by-base error rates before and after

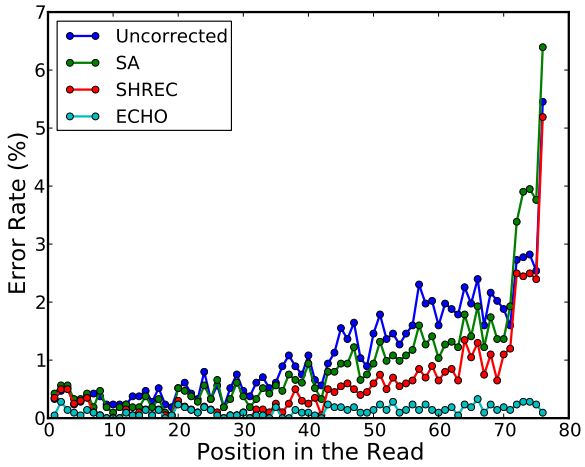


Figure 3.3: **Position-specific by-base error rates for 76 bp PhiX174 data D1 with sequence coverage depth 30.** Error rates before and after applying the three error-correction algorithms are shown. Spectral Alignment (or SA; see [7]) and SHREC [44] are able to improve the error rate in intermediate positions, but they both become less effective for later positions. In contrast, our error-correction algorithm ECHO remains quite effective throughout the entire read length, reducing the error rate at the end of the read from about 5% to a small fraction of 1%.

applying the error-correction algorithms. It is well-known that the error rate of the Illumina platform generally increases toward the end of the read, reaching as high as 5% in the example shown. Although SA and SHREC were able to reduce this effect to a certain degree, they both became less effective for later positions. In contrast, ECHO remained quite effective throughout the entire read length. In particular, the error rate at the end of the read was reduced from about 5% to a small fraction of 1%.

We considered the effect of coverage depth on the performance of our algorithm. Figure 3.4 shows the position-specific by-base error rates before and after applying ECHO on PhiX174 data with varying sequence coverage depths. At a coverage depth of 15, the error rate could be controlled throughout the entire read length, resulting in only a slight increase in the error rate toward the end of the read.

Although ECHO outperformed the other error correction methods on the simulated human data set D4 (see Tables 3.1 and 3.2), the improvement was not as pronounced as for the Drosophila and PhiX174 data sets. This demonstrates that ECHO is most effective on genomes with limited repetitive structure. Although the read overlap approach might be less susceptible to repeat regions than a k-mer spec-

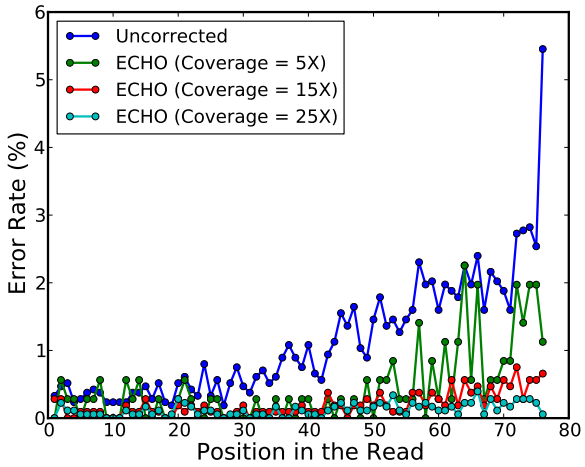


Figure 3.4: **Position-specific by-base error rates before and after running our error-correction algorithm ECHO on PhiX174 data with varying coverage depths.** The ability of our algorithm to correct sequencing errors improves as the sequence coverage depth increases. A coverage depth of 15 seems sufficient to control the error rate quite well throughout the entire read length.

trum method, it does not adequately overcome this obstacle. Further work is required to handle repeat regions, which are common in more complex genomes, such as mammalian genomes.

Table 3.1: Error rates for haploid data before and after running error-correction algorithms.

		By-base error rate (%)				By-read error rate (%)			
Data	Cov	Uncorr	SA	SHREC	ECHO	Uncorr	SA	SHREC	ECHO
(D1) PhiX174	5	1.00	0.72	0.81	0.45	31.27	21.13	25.36	15.49
	10	0.99	0.83	0.64	0.38	29.48	18.48	21.27	10.16
	15	1.01	0.72	0.65	0.27	28.85	17.48	19.75	8.55
	20	1.00	0.81	0.66	0.25	28.00	17.35	18.53	7.97
	25	1.08	0.96	0.64	0.30	28.33	18.23	18.35	8.75
	30	1.02	1.00	0.56	0.25	28.02	17.82	17.86	8.18
(D2) *D. mel.*	13	2.93	2.93	2.57	1.15	48.85	48.85	25.68	16.99
(D3) Simulated	5	0.96	0.91	0.58	0.53	15.60	15.59	15.94	11.25
D. mel.	10	0.98	0.91	0.52	0.36	12.34	12.34	13.38	3.78
	15	1.00	0.92	0.50	0.21	12.52	12.51	12.74	1.84
	20	0.99	0.90	0.49	0.10	12.58	12.58	12.57	0.89
	25	1.00	0.93	0.50	0.10	12.71	12.70	12.49	0.86
	30	0.99	0.93	0.50	0.09	12.79	12.79	12.45	0.88
(D4) Simulated	5	1.01	1.00	0.68	0.61	26.94	15.62	18.06	12.52
Human	10	1.01	1.00	0.60	0.41	27.13	13.15	15.58	6.02
	15	1.00	1.01	0.53	0.28	27.08	13.15	13.07	5.53
	20	1.01	0.99	0.53	0.27	27.06	13.23	12.86	5.57
	25	1.00	0.98	0.52	0.27	26.93	13.33	12.73	5.77
	30	1.00	0.97	0.53	0.26	26.95	13.45	12.84	5.88

Cov: sequence coverage depth. Uncorr: uncorrected reads. SA: spectral alignment method used in EULER-USR [7]. SHREC: method introduced by [44]. ECHO: our error-correction method. "By-base" error rate refers to the ratio of the total number of incorrect bases in the reads to the total length of the reads. "By-read" error rate corresponds to the ratio of the number of reads each with *at least* one miscalled base to the total number of reads considered.

Table 3.2: Numbers of corrected errors and introduced errors after running error correction.

		# Corrected Errors			# Introduced Errors			Gain %		
Data	Cov	SA	SHREC	ECHO	SA	SHREC	ECHO	SA	SHREC	ECHO
(D1) PhiX174	5	78	43	148	2	0	0	28.1	15.9	54.8
	10	196	181	328	109	0	0	16.3	34.0	61.5
	15	300	288	605	64	0	0	28.8	35.1	73.8
	20	376	363	811	169	0	0	19.2	33.6	75.2
	25	476	509	1043	320	1	0	10.8	35.1	72.0
	30	558	618	1245	524	0	0	2.1	37.6	75.8
(D2) *D. mel.*	13	0	2931051	5098551	0	1919856	40684	0.0	12.2	61.1
(D3) Simulated	5	1682	1813	3226	1401	26	0	5.8	36.9	44.9
D. mel.	10	4083	4272	7701	3346	17	0	7.5	43.4	63.5
	15	6304	7220	14235	4999	11	11	8.9	47.8	78.6
	20	8317	9539	18395	6453	12	45	9.4	47.9	89.2
	25	10304	11916	22355	8577	16	22	6.9	47.7	89.6
	30	12195	14232	28012	10481	21	27	5.8	47.8	90.6
(D4) Simulated	5	4082	4658	5419	3870	1047	365	1.7	28.5	39.9
Human	10	10117	10805	15833	9834	1189	713	1.1	38.0	59.7
	15	14950	17220	27881	15120	397	830	−0.5	44.8	72.1
	20	19731	23190	37933	19013	444	886	1.4	45.1	73.4
	25	24228	28954	46191	23169	585	714	1.7	45.5	72.9
	30	29038	34912	55706	26758	722	491	3.1	45.7	73.9

Cov: sequence coverage depth. Uncorr: uncorrected reads. SA: spectral alignment method used in EULER-USR [7]. SHREC: method introduced by [44]. ECHO: our error correction method. "# Corrected Errors" is the number of errors in the reads that were corrected. "# Introduced Errors" is the number of bases in the reads that were correct but were miscorrected by the error correction. Gain is defined as (#Corrected Errors − #Introduced Errors)/(#Actual Errors). The read length for D1, D3, and D4 was 76 bp. The read length for D2 was 36 bp.

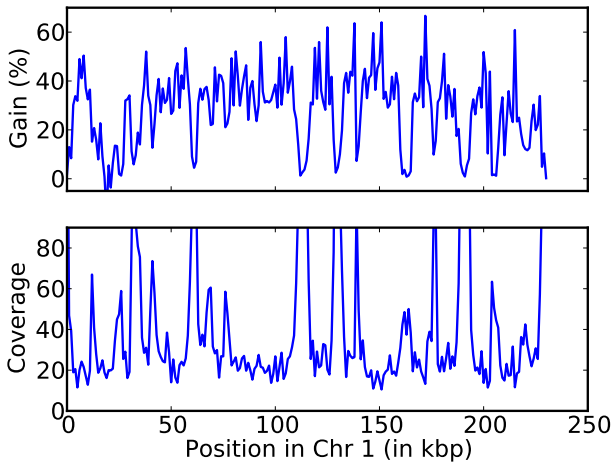


Figure 3.5: **The gain of ECHO and the position-specific coverage for Chromosome 1 of the yeast data D6.** Each plot uses bins of 1000 bp. The top plot shows the gain of ECHO, defined as the number of corrected errors minus the number of introduced errors, divided by the number of actual errors. The bottom plot shows the position-specific coverage.

3.6.3 Whole-Genome Error Correction

To evaluate the performance of ECHO on whole genome data, we ran an experiment on yeast data D6 from the NCBI Sequence Read Archive. The data set consisted of 7.1 million reads from a close variant of *S. cerevisiae*. Results from the experiment are shown in Table 3.3 and Table 3.4. The performance of all evaluated error correction algorithms was significantly worse for this data set than the others. Closer examination of the data revealed that the mitochondrial DNA (mtDNA) had much higher coverage than the other chromosomes, 210x compared to 19.5x. Because ECHO compares a read's estimated coverage to the estimated coverage over the *entire* genome, nearly every read from the mtDNA was not corrected. This contrasts with SHREC, which does not account for such drastic differences in coverage across the genome.

This suggests that although ECHO assumes uniform coverage of the genome when determining its parameters, when it encounters data with highly non-uniform coverage, it avoids confusion caused by the non-uniform coverage by requiring the reads to pass the coverage checks described in Chapter 3.3.3. It should be noted that repetitive structure would also cause a read not to pass the coverage check, which would cause ECHO to be more conservative around repeat regions.

In order to better quantify the performance of the error correction algorithms, we

Table 3.3: Error rates for yeast data before and after running error correction on the whole-genome yeast data.

	By-base error rate (%)				By-read error rate (%)			
	SHREC		ECHO		SHREC		ECHO	
	Before	After	Before	After	Before	After	Before	After
Entire D6	0.90	0.83	0.9	0.71	14.9	10.8	14.9	10.9
Conditional D6	0.89	0.79	1.28	0.83	14.8	10.6	17.3	8.4

SA results are not shown because it did not perform any error correction for this data set. *Conditional* refers to the subset of data on which error correction was performed by each algorithm. Specifically, for ECHO it refers to the reads that passed the coverage checks. For SHREC, it refers to the reads that did not get discarded.

Table 3.4: Numbers of corrected errors and introduced errors after running error correction on the whole-genome yeast data D6.

	# Corrected Errors		# Introduced Errors		Gain %	
Data	SHREC	ECHO	SHREC	ECHO	SHREC	ECHO
Entire D6	743434 out of 2302181	498588 out of 2302181	502726	12087	10.5	21.1
Conditional D6	743434 out of 2236313	498588 out of 1393181	502726	12087	10.8	35.8

SA results are not shown because it did not perform any error correction for this data set. As explained in Table 3.3, *Conditional* refers to the subset of data on which error correction was performed by each algorithm. Gain is defined as (# Corrected Errors − #Introduced Errors)/(# Actual Errors). In general, ECHO is more conservative than SHREC. See Figure 3.5 for position-specific gain information.

divided the data set into two groups: reads that passed the coverage checks and those that did not. When we considered only those reads that passed the coverage checks, the performance of ECHO improved significantly.

Figure 3.5 shows the gain for ECHO and the coverage along chromosome 1. As mentioned before, the gain corresponds to the number of corrected errors minus the number of introduced errors, divided by the number of actual errors. The plots illustrate that when the coverage increases significantly, the gain tends to drop. This is because the reads from these positions generally do not pass the coverage checks ECHO imposes.

Table 3.4 illustrates that SHREC corrected about 1.5 times more bases than did ECHO, but the number of errors introduced by SHREC was 41 times greater than that introduced by ECHO. As a result, ECHO outperforms SHREC in terms of gain.

Table 3.5: Robustness of our algorithm ECHO with respect to the choice of the keyword length k.

k	By-base error rate (%)
7	0.107
8	0.107
9	0.107
10	0.108
11	0.108
12	0.109
13	0.111
14	0.115
15	0.193

Our algorithm was run on simulated data D3 with sequence coverage depth 19. The average by-base error rate in the simulated reads was around 1%. For the optimal case (i.e., $k = 13$), ECHO reduced the error rate from 1% to 0.0061%. In comparison, (3.9) suggests using $k = 10$, in which case the associated error rate was 0.0062%.

Because of the coverage checks it performs, ECHO was more conservative in correcting reads coming from highly covered regions of the genome. We expect this to be an advantageous feature of ECHO, as a high *estimated* coverage is often indicative of repeat structure.

3.6.4 Robustness with respect to the choice of the keyword length k

In Chapter 3.3.2, we provided a rough guideline for choosing the keyword length k in the neighbor-finding algorithm. In particular, if the genome size is completely unknown, we suggested using (3.9) to set k. Here, we show empirically that the performance of ECHO with k chosen as in (3.9) is comparable to the performance of the algorithm with the optimal k that maximizes the error correction ability.

For this study, we simulated a data set of type D3 consisting of 25,000 reads of length 76 bp. This corresponds to a sequence coverage depth of 19. The average by-base error rate in the simulated reads was around 1%. For each given k, we used the method described in Chapter 3.3.2 to determine δ^* and ϵ^*; see (3.12) and the surrounding discussion. Then, Algorithm 3 was run with the chosen k and the associated $(\delta, \epsilon) = (\delta^*, \epsilon^*)$ to find the neighbor sets $(\mathcal{N}_r)_{r\in\mathcal{R}}$ and the identification maps $(I_{r\to s})_{r\in\mathcal{R}, s\in\mathcal{N}_r}$. Finally, Algorithm 4 was used to perform error correction. Table 3.5 shows that the performance of ECHO is robust with respect to the choice

of k. For the simulated data we considered, the value of k that led to the largest improvement in the error rate was $k = 7$, in which case ECHO reduced the error rate from 1% to 0.107%. In comparison, (3.9) suggests using $k = 10$; the associated error rate for this case was 0.108%, which is comparable to the optimal case.

3.6.5 Error correction accuracy for diploid genomes

As described in Chapter 3.3, our algorithm ECHO can handle diploid data as well as haploid data. Recall that ECHO explicitly models heterozygosity in diploid genomes and has a parameter (denoted h) corresponding to the probability that a given site is heterozygous. Setting $h = 0$ is equivalent to reducing the model to the haploid case. Neither SA nor SHREC explicitly models diploidy, so they may be compared with the case of $h = 0$ in our model.

We tested the methods on simulated diploid data D5. In simulating the diploid genome, the probability of heterozygosity was set to 10^{-3} for each site. Table 3.6 and Table 3.7 show a summary of error correction results on diploid data D5. As in the haploid case, SA was not effective in reducing the by-base error rate, though it was able to reduce the by-read error rate reasonably well. SHREC significantly outperformed SA on the by-base error rate, whereas their by-read error rates were comparable. ECHO with $h = 0$ outperformed both SA and SHREC by several folds for coverages 15 and higher. For these coverage depths, ECHO reduced the by-read error rate from around 27% to about 2.3%. The improvement in the by-base error rate was equally significant in proportion. For $h > 0$, ECHO displayed accuracy greater than that for $h = 0$, demonstrating the utility of explicitly modeling the diploidy of the genome.

Table 3.8 shows the detection accuracy of heterozygous sites in the diploid data. The *precision* and *recall* are shown for varying levels of coverage and for both $h = 10^{-3}$ and $h = 10^{-4}$. Precision and recall are commonly-used performance measurements in information retrieval tasks. In this context, precision is the number of correctly called heterozygous sites divided by the number of all called heterozygous sites (including incorrectly called heterozygous sites), and recall is the number of called heterozygous sites divided by all true heterozygous sites.

By filtering the bases on quality, we can trade off recall for higher precision. These results demonstrate that, with high enough coverage, ECHO provides a reliable reference-free method of detecting bases that originated from heterozygous sites and inferring the corresponding genotypes.

Table 3.6: Error rates for diploid data D5 before and after error correction.

Cov	By-base error rate (%)						By-read error rate (%)					
	Uncorr	SA	SHREC	ECHO with $h =$			Uncorr	SA	SHREC	ECHO with $h =$		
				0	10^{-4}	10^{-3}				0	10^{-4}	10^{-3}
10	1.02	1.22	0.58	0.40	0.35	0.34	27.58	13.15	15.45	5.47	5.30	4.83
20	1.02	1.12	0.54	0.21	0.16	0.16	27.22	12.83	13.27	2.84	2.02	1.88
30	1.01	1.10	0.52	0.10	0.08	0.08	27.17	13.02	13.01	2.04	1.06	1.10
40	1.00	1.01	0.50	0.10	0.07	0.08	27.12	13.21	12.76	1.99	1.16	1.17
50	1.00	1.09	0.52	0.10	0.07	0.07	26.97	13.34	13.19	2.12	1.18	1.21
60	1.00	1.09	0.52	0.10	0.08	0.08	26.99	13.63	13.00	2.43	1.41	1.43

h: prior probability of heterozygosity used in ECHO. See Table 3.1 for a full description of notation. In computing the error rate of ECHO for $h > 0$, we compared the genotype assigned by ECHO to each short-read with the true genotype sequence in the diploid genome from which the short-read originated. Errors in SA, SHREC, and ECHO for $h = 0$ were computed in the usual way by comparing with the true haplotype. The read length was 76 bp.

Table 3.7: Numbers of corrected errors and introduced errors for diploid data D5 after error correction.

Cov	# Corrected Errors					# Introduced Errors				
	SA	SHREC	ECHO with $h =$			SA	SHREC	ECHO with $h =$		
			0	10^{-4}	10^{-3}			0	10^{-4}	10^{-3}
10	4152	4311	6357	7785	7865	6128	147	178	6	18
20	8140	9488	16490	19499	19539	10185	107	320	196	221
30	12169	14348	27740	31797	31797	15074	85	464	615	697
40	15914	19183	36884	42214	42215	19405	81	600	780	855
50	19493	23808	45769	52411	52408	24141	437	819	888	976
60	22934	28547	55458	63209	63208	28007	420	1125	996	1079

h: prior probability of heterozygosity used in ECHO. See Table 3.2 for a full description of notation. In computing the error rate of ECHO for $h > 0$, we compared the genotype assigned by ECHO to each short-read with the true genotype sequence in the diploid genome from which the short-read originated. Errors in SA, SHREC, and ECHO for $h = 0$ were computed in the usual way by comparing with the true haplotype. The read length was 76 bp.

Table 3.8: Heterozygous allele detection efficiency for simulated diploid data D5.

		Unfiltered		Quality ≥ 20		Quality ≥ 30		Quality ≥ 40	
h	Cov	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
10^{-3}	10	0.97	0.77	0.99	0.59	0.99	0.48	1.00	0.37
	20	0.90	0.96	0.92	0.93	0.93	0.90	0.95	0.88
	30	0.82	0.99	0.85	0.99	0.86	0.99	0.88	0.97
	40	0.83	0.99	0.86	0.99	0.88	0.99	0.88	0.99
	50	0.84	0.99	0.86	0.99	0.87	0.99	0.88	0.98
	60	0.85	0.98	0.87	0.98	0.88	0.97	0.89	0.97
10^{-4}	10	0.99	0.69	0.99	0.48	1.00	0.37	1.00	0.27
	20	0.91	0.94	0.93	0.90	0.95	0.88	0.95	0.83
	30	0.84	0.99	0.86	0.99	0.88	0.97	0.89	0.96
	40	0.84	0.99	0.87	0.99	0.88	0.99	0.89	0.98
	50	0.85	0.99	0.87	0.99	0.88	0.98	0.89	0.98
	60	0.86	0.98	0.88	0.97	0.89	0.97	0.89	0.97

The precision and recall of the genotypes called by ECHO when modeling a diploid genome are shown. Precision is the number of true positives over the sum of true positives and false positives. Recall is the number of true positives divided by the number of positives. A true positive is a correctly identified heterozygous site. A false positive is a homozygous site mistakenly classified as a heterozygous site. The number of positives is the number of actual heterozygous sites. Two cases are shown: $h = 10^{-3}$ and $h = 10^{-4}$. Every genotype called by ECHO is given a quality score. The "Unfiltered" column shows the results when all called genotypes are considered. "Quality $\geq x$" shows the results when only genotypes with quality scores greater than x are considered.

3.6.6 Effects on *de novo* assembly

The data sets we considered were real PhiX174 data D1 and simulated data D3 from a 100 kbp region of *D. melanogaster*; these two experiments were repeated 20 times each. We also considered simulated data D3 from a 5 Mbp region of *D. melanogaster*; this experiment was repeated twice. For the PhiX174 data set, Velvet was run on the following four sets of short-reads:

- Uncorrected reads, corresponding to the original simulated reads.
- Error-corrected reads processed by SA, SHREC, or ECHO.

For the simulated *D. melanogaster* data sets, Velvet was also run on the following set of short-reads, in addition to the above four sets:

- Perfect reads, corresponding to the error-free reads from which the uncorrected reads were simulated.

The results for PhiX174 are shown in Table 3.9, where "Max" denotes the maximum contig length, and "N50," defined in the caption, is a statistic commonly used to assess the quality of *de novo* assembly. Larger values of Max and N50 indicate better assembly quality. For nearly all sequence coverage depths, note that using ECHO-corrected reads led to the largest Max and N50 values.

The assembly results for simulated 100 kbp and 5 Mbp *D. melanogaster* data are summarized in Table 3.10 and Table 3.11, respectively. There was a clear advantage to performing error correction prior to assembly, and error correction produced an appreciable improvement over uncorrected reads. In addition to Max and N50, we assessed the assembly quality using the following two new measures:

- SeqCov1000 is the percentage of the genome covered by the contigs of length $\geq$ 1000 bp.
- Error1000 is the percentage of the total number of assembly errors (measured by edit distance from the true genome, with a penalty of 1 for mismatch, insertion, or deletion) in those contigs with length $\geq$ 1000 bp, with respect to the total length of those contigs.

For coverage depths ≥ 20, SeqCov1000 values for all three error correction methods were comparable to that for perfect reads.

As Table 3.9 and Table 3.11 illustrate, the key advantage of ECHO is its consistency; i.e., the performance of ECHO is consistently good for all coverage depths and all data sets, while the other error correction methods seem more erratic (e.g., see the performance of SHREC in Table 3.9, and the performance of SA in Table 3.11 for coverages ≥ 20).

3.6.7 Running times

We ran the three error-correction algorithms on simulated data to compare their running times. We first simulated a genome sequence by drawing each base uniformly at random from $\{A, C, G, T\}$. Then, in simulating short-reads, the by-base error rate was set to 1%, and errors were drawn uniformly at random from the alternative bases. We varied the number of reads and the length of the genome, while fixing the sequence coverage depth at 20. All experiments were done on a Mac Pro with two quad-core 3.0 GHz Intel Xeon processors and 14 GB of RAM.

Shown in Table 3.12 is a summary of running times. Since our algorithm needs to determine the parameters δ^* and ϵ^* via (3.12), its running time is divided into two parts: parameter selection time, shown inside parenthesis, and the running time for neighbor finding and error correction. Although our algorithm is slower, note that its running time is within the range of practical use. Also, we point out that SA and our algorithm each used only a single core, while SHREC was allowed to use all 8 cores.

If one has enough memory to store the entire neighbor sets $(\mathcal{N}_r)_{r\in\mathcal{R}}$ and identification maps $(I_{r\to s})_{r\in\mathcal{R},s\in\mathcal{N}_r}$, the running time of ECHO scales linearly in the number N of reads, provided that k is chosen appropriately so that $\max_K |C(K)|$ is linear in the coverage depth. This behavior is demonstrated in Table 3.12 for $N \leq 2.5 \times 10^6$. However, if the available amount of memory is not sufficient, our implementation first partitions the read set $\mathcal{R}$ into smaller subsets $\mathcal{R}_1, \ldots, \mathcal{R}_n$. Then, successively for each $i = 1, \ldots, n$, the neighbor sets $(\mathcal{N}_r)_{r\in\mathcal{R}_i}$ and identification maps $(I_{r\to s})_{r\in\mathcal{R}_i,s\in\mathcal{N}_r}$ are constructed, and error correction is performed for the reads in $\mathcal{R}_i$. The current $(\mathcal{N}_r)_{r\in\mathcal{R}_i}$ and $(I_{r\to s})_{r\in\mathcal{R}_i,s\in\mathcal{N}_r}$ are discarded before moving onto the next i. In this fashion, the running time of ECHO scales quadratically in the number of reads.

Table 3.9: *De novo* assembly results for uncorrected and corrected PhiX174 data D1.

Coverage	Reads	N50 (bp)	Max (bp)
5	Uncorrected	219.4	401.5
	SA	238.0	422.4
	SHREC	148.5	246.2
	ECHO	261.2	475.7
10	Uncorrected	841.2	1452.4
	SA	1005.7	1629.6
	SHREC	608.6	1017.9
	ECHO	1246.4	1833.7
15	Uncorrected	2561.4	2704.7
	SA	3095.8	3126.9
	SHREC	2014.5	2369.8
	ECHO	3345.6	3386.7
20	Uncorrected	3725.9	3773.3
	SA	4551.3	4569.5
	SHREC	4179.2	4179.2
	ECHO	4706.7	4706.7
25	Uncorrected	4747.5	4747.5
	SA	5208.6	5208.6
	SHREC	5215.6	5215.6
	ECHO	5335.7	5335.7
30	Uncorrected	5224.1	5224.1
	SA	5337.1	5337.1
	SHREC	4855.5	4855.5
	ECHO	5337.1	5337.1

The length of the PhiX174 genome is 5386 bp. Velvet was used to carry out the assembly. Max: the maximum contig length. N50: a statistic commonly used to assess the quality of *de novo* assembly. It is computed by sorting all contigs by their size in decreasing order and adding the length of these contigs until the sum is greater than 50% of the total length of all contigs. The length of the last added contig corresponds to N50. A larger N50 indicates a better assembly. For all sequence coverage depths, the largest Max and N50 values were observed for the assembly carried out using our error-corrected reads.

Table 3.10: *De novo* assembly results for simulated data D3 from 100 kbp region of *D. melanogaster*.

Cov	Reads	N50 (bp)	Max (bp)	# Contig1000	SeqCov1000	Error1000
10	Uncorrected	838.3	2708.3	28.2	41.0%	0.009%
	SA	1402.3	4258.7	37.2	66.4%	0.003%
	SHREC	984.7	2983.8	31.1	48.8%	0.060%
	ECHO	1465.7	4235.7	38.2	69.3%	0.004%
	Perfect	1816.6	4903.7	37.9	77.2%	0.000%
15	Uncorrected	4471.0	11811.3	26.7	94.7%	0.005%
	SA	11222.6	19775.8	15.3	98.5%	0.001%
	SHREC	8082.8	16511.9	18.5	97.3%	0.003%
	ECHO	12260.2	22544.8	13.6	98.6%	0.004%
	Perfect	14657.3	24826.0	11.6	99.1%	0.000%
20	Uncorrected	32259.8	38847.4	6.7	99.9%	0.001%
	SA	64520.9	66720.6	2.7	99.9%	0.000%
	SHREC	59463.2	62277.7	3.0	99.9%	0.000%
	ECHO	69527.9	71245.0	2.8	99.9%	0.001%
	Perfect	75614.0	76890.5	2.2	99.9%	0.000%
25	Uncorrected	67886.7	71308.3	2.4	99.9%	0.000%
	SA	89654.5	89654.5	1.5	99.9%	0.000%
	SHREC	86589.4	86589.4	1.6	99.9%	0.000%
	ECHO	90380.1	90380.1	1.5	99.9%	0.000%
	Perfect	94153.8	94153.8	1.3	100.0%	0.000%
30	Uncorrected	95056.9	95056.9	1.3	100.0%	0.000%
	SA	99831.3	99831.3	1.1	100.0%	0.000%
	SHREC	99954.7	99954.7	1.0	100.0%	0.000%
	ECHO	99954.8	99954.8	1.0	100.0%	0.000%
	Perfect	99955.6	99955.6	1.0	100.0%	0.000%

Assembly was performed using Velvet with $k = 39$ and automatic coverage cutoff. Uncorrected reads are the original simulated reads. To correct for sequencing errors, we used SA, SHREC, and ECHO prior to assembly; the preprocessed reads are denoted by the corresponding error-correction algorithm employed. Perfect reads correspond to the error-free reads from which uncorrected reads were simulated. Max and N50 are as explained in Table 3.9. SeqCov1000 is the percentage of the genome covered by the contigs of length $\geq$ 1000 bp. Error1000 denotes the percentage of the total number of assembly errors (measured by edit distance from the true genome, with a penalty of 1 for mismatch, insertion, or deletion) in those contigs with length $\geq$ 1000 bp, with respect to the total length of those contigs. The results shown above are average values from 20 simulations.

Table 3.11: *De novo* assembly results for simulated data D3 from 5 Mbp region of *D. melanogaster*.

Cov	Reads	N50 (bp)	Max (bp)	# Contig1000	SeqCov1000	Error1000
10	Uncorrected	840.0	4796.0	1400.5	41.5%	0.009%
	SA	1283.0	6514.5	1835.0	63.9%	0.005%
	SHREC	996.0	5315.5	1625.0	50.8%	0.094%
	ECHO	1194.5	7338.5	1798.0	60.8%	0.006%
	Perfect	1721.5	10044.0	1871.5	75.4%	0.000%
15	Uncorrected	4476.0	20978.5	1293.5	95.0%	0.004%
	SA	9732.0	40215.5	740.0	98.8%	0.001%
	SHREC	7838.0	40424.0	866.0	98.2%	0.015%
	ECHO	10698.0	40739.0	669.0	99.0%	0.001%
	Perfect	13847.0	48053.5	532.5	99.4%	0.000%
20	Uncorrected	29377.0	96954.0	291.5	99.8%	0.001%
	SA	80231.0	233239.0	107.0	100.0%	0.000%
	SHREC	80847.5	225311.0	104.0	100.0%	0.001%
	ECHO	95977.5	242397.0	81.0	100.0%	0.000%
	Perfect	114364.0	312355.0	67.0	100.0%	0.000%
25	Uncorrected	158180.5	483918.5	53.0	100.0%	0.000%
	SA	799798.5	1232231.5	13.0	100.0%	0.000%
	SHREC	951499.0	1260863.5	11.5	100.0%	0.000%
	ECHO	931249.0	1260863.5	10.5	100.0%	0.000%
	Perfect	951499.0	1391995.0	8.5	100.0%	0.000%
30	Uncorrected	895733.5	1399461.5	8.0	100.0%	0.000%
	SA	3598579.0	3598579.0	2.5	100.0%	0.000%
	SHREC	4266643.0	4266643.0	1.5	100.0%	0.000%
	ECHO	4266643.0	4266643.0	1.5	100.0%	0.000%
	Perfect	4266643.0	4266643.0	1.5	100.0%	0.000%

Assembly was performed using Velvet with $k = 39$ and automatic coverage cutoff. Uncorrected reads are the original simulated reads. To correct for sequencing errors, we used SA, SHREC, and ECHO prior to assembly. Perfect reads correspond to the error-free reads from which uncorrected reads were simulated. Max and N50 are as explained in Table 3.9. SeqCov1000 is the percentage of the genome covered by contigs of length ≥ 1000 bp. Error1000 denotes the percentage of the total number of assembly errors (measured by edit distance from the true genome, with a penalty of 1 for mismatch, insertion, or deletion) in those contigs with length ≥ 1000 bp, with respect to the total length of those contigs.

Table 3.12: Comparison of running time, in minutes, for simulated data with 76 bp reads.

	N, Number of reads in the input data				
Method	10^4	10^5	10^6	2.5×10^6	5×10^6
SA	0.03	0.36	4.59	12.44	26.69
SHREC	0.19	1.48	17.00	39.52	90.64
ECHO	(1.52) 0.67	(7.48) 5.37	(19.97) 56.03	(42.55) 143.95	(78.2) 300.77

SA and ECHO each used only a single core, while SHREC was allowed to use all 8 cores, on a Mac Pro with two quad-core 3.0 GHz Intel Xeon processors and 14 GB of RAM. For ECHO, the numbers shown inside the parentheses correspond to the running times of parameter selection.

Chapter 4

Conclusions

4.1 Summary

In this thesis, we addressed problems that are pertinent to the next-generation short-read sequencing technologies: the shorter read lengths and the higher error rates. We adopted two complementary approaches, namely, developing improved basecallers and error correction algorithms.

In Chapter 2.4, we proposed a model based basecaller for Illumina Genome Analyzer, called BayesCall. BayesCall is the first model that considers all the known non-stationary noises which include phasing and prephasing effects, death effects, residual effects, and background noises. Because BayesCall is based on a generative model, unsupervised learning can be achieved without labeled training data. Therefore, it is applicable to *de novo* sequencing.

Empirical studies show that BayesCall improves by-base error rates by more than 39% and the improvement is more pronounced toward the end of reads. BayesCall also improves by-read error rates by more than 40%. In other words, it produces much more perfect reads without any erroneously called bases which, as shown later, is important to de novo assembly. The errors produced by Bustard are usually non-uniformly distributed over alternative bases. This bias violates what most of the variant detection algorithm assume, and it can increase the number of false positives. One of the most pronounced effet that researchers have discovered is the "anomalous T" effect in which many of the bases are erroneously called as base "T" at the end of reads. BayesCall also significantly reduces the biases by modeling the residual effect. One additional benefit of the probabilistic modeling used in BayesCall is that it can produce quality scores with high discriminating ability. In our experiment, filtering bases with the quality scores produced by BayesCall can result in more accurately called bases. It is also capable of producing base-specific quality scores.

One drawback of the BayesCall algorithm is that it requires a simulated annealing step for inferring the most probable base, and simulated annealing is computationally expensive. To solve the problem, we devised an algorithm called naiveBayesCall in Chapter 2.6. NaiveBayesCall uses optimization methods to approximate the MAP estimate of the original BayesCall model. The result is an algorithm that runs sixty times faster than the original BayesCall algorithm. Empirical studies show that while naiveBayesCall largely reduces the running time, it also preserves the accuracy of BayesCall. With this improvement, it is practicable to use naiveBayesCall on a desktop computer. We then perform a comprehensive study on the effect of an improved basecall algorithm on downstream data analysis.

We considered two different scenarios. The first scenario we considered is *de novo* assembly. In *de novo* assembly, the whole genome is to be assembled from the reads without a reference sequence. Thus, any errors in the reads can largely increase the difficulty of producing the assembly. Our study shows that with reads output by naiveBayesCall, Velvet can construct an assembly with more and longer contigs. The quality of the obtained assembly, measured by N50, is also improved. This indicates that the improved basecall quality will be very useful in a *de novo* sequencing project. The second scenario we considered is the detection of single nucleotide polymorphism. In this setting, a reference genome is available and the goal is to detect single nucleotide polymorphisms from the reads. As mentioned before, it has been discovered that there exists systematic biases in reads produced by Bustard. These biases can potentially confuse a SNP caller. BayesCall and naiveBayesCall can significantly reduce the bias. In our study, MAQ called SNPs with much higher precision and comparable recall with BayesCall and naiveBayesCall reads. Since validation of these SNPs can be expensive and time consuming, higher precision in SNP detection can be very helpful in genome researches.

In Chapter 3, we considered correcting reads as a preprocessing step before downstream sequence analysis. It is inevitable to have some errors in sequencing data even with advanced basecall algorithms. However, since each region in the genome is covered by multiple reads, sequencing errors can potentially be corrected by aggregating information in the reads. Based on this concept, we developed an algorithm called ECHO. ECHO does not require a reference sequence and is applicable to *de novo* sequencing projects. ECHO has several novelties. First, it tries to utilize as much information from the reads as possible by using read overlaps. Instead of summarizing reads with k-mer frequencies, it uses read overlaps to minimize the chance of misalignment, and ultimately, mis-correction. In addition, it also uses reads to estimate position specific substitution rates. These rates can increase the power of detecting and correcting sequencing errors especially when the coverage is low to moderate.

Last but not least, ECHO automatically estimates all the parameters without any input from the user except for the reads. In most of the existing error correction algorithms, some parameters that are crucial to the error correction must be specified by the user. Specifying non-optimal parameters sometimes lead to dramatically worsen performances. However, it can be hard for user to specify optimal parameters without extensive knowledge and sometimes a reference sequence. ECHO solves this problem by estimating parameters automatically from a probabilistic modeling of read overlaps and sequencing errors. An additional benefit of the modeling is that ECHO is able to correct diploid genomes which is unfeasible with k-mer based correction algorithms.

In our experiments, ECHO can reduce both by-base and by-read error rates by more than 50% even at a low coverage. Moreover, ECHO also has very few mis-corrections. In an empirical study on a whole-genome yeast dataset, ECHO has 40 times less mis-corrections compared to the alternative algorithm. We also applied ECHO to a simulated diploid dataset. ECHO is the only algorithm that can distinguish heterozygous site from sequencing errors. It consistently reduces error rates by more than 50%. Heterozygous sites can also be detected simultaneously with high recall and precision. Finally, we also tested how error correction can facilitate *de novo* assembly. In general, applying error correction algorithms on the reads before running an assembly algorithm can lead to improved results. In particular, the assembled genome output by Velvet on ECHO corrected reads is almost as good as Velvet on perfect reads in terms of genome coverage, maximum contig length, and N50. Based these observations, we conclude that ECHO can be very useful in many sequencing projects.

4.2 Future Directions

In this section, I will propose several interesting and important problems that can be addressed in the near future with the extensions of progress described in this thesis.

4.2.1 Improved basecaller for Life Technologies Ion Torrent and Pacific Biosciences SMRT

Recently, there have been several attempts to develop new sequencing technologies that run faster and cheaper, and produce longer and more accurate reads. Among these so-called "third generation" sequencing technologies, there is a great interest in Pacific Biosciences' SMRT sequencing technology. SMRT is the first single molecule

real time sequencing technology. The first benefit of SMRT is that it only requires a single DNA molecule, and thus the amplification step is not necessary. Therefore, the bias that is introduced in amplification steps can be reduced. Furthermore, it also avoids the phasing and prephasing problems that most of the sequencing by synthesis technologies possess. All of the aforementioned improvements enable SMRT to produce longer reads. Finally, since the polymerization process is recorded in real time, there is much more information and potentially more statistical patterns that are yet to be discovered and utilized. Once the SMRT platform is widely available to researchers, it will be fascinating to uncover all the questions left unanswered in the realm of biology due to limitations on the sequencing technologies.

In the past few months, Ion Torrent acquired by Life Technologies has introduced a novel sequencing concept. Ion Torrent completes the entire sequencing with a specially designed disposable silicon chip. Though it has higher per-base cost and lower throughput, it runs significantly faster with much lower starting costs. It is the first technology that detects protons instead of photons. Hence, the measured signals can be quite different from what other technologies produce. Since the underlying chemistry of Ion Torrent is similar to 454 sequencing technology, the main challenge for its basecaller will be resolving long homopolymers with different lengths, and decoupling of imperfect phasing and prephasing. In the 15th Annual International Conference on Research in Computation Molecular Biology, Life Technologies announced that they will have a grand challenge seeking for a better basecaller that will reduce the error rates by one half. It will also be very interesting to see how improvements in basecaller can boost Ion Torrent's performance.

4.2.2 *De novo* assembly algorithm for reads from different sequencing technologies

As mentioned in the previous section, there are several emerging new sequencing technologies. The reads produced by these emerging technologies and various existing technologies all have very different characteristics. For example, Illumina platform has very low indel error rates with read length up to 100bp while Pacific Biosciences SMRT has higher error rates but much longer read lengths. In particular, it is possible to produce gapped reads with various gap sizes in the SMRT technology. Since each technology has its own strength and weakness, it is appealing to combine reads from different sequencing technologies. More specifically, many *de novo* sequencing projects have difficulty assemble genome regions that are repetitive such as telomere and centromere region. It is important to have an assembler that can exploit Illumina's accurate reads and SMRT's longer read lengths to produce a high quality assembly

in these regions. In Chapter 3, we developed an error correction algorithm with read overlap technique. The same technique can be applied to the aforementioned assembly problem.

4.3 Discussion

Currently, a few large-scale resequencing projects are either underway or in near inception. In early 2008, an international research consortium announced the 1000 Genomes Project[11] [1], which will resequence the genomes of at least 1000 humans around the world. This effort will provide a comprehensive picture of human genomic variation. A project called 1001 Genomes[2] will resequence a large number of Arabidopsis genomes, and the Drosophila Population Genomics Project (DPGP)[3] will resequence a similar quantity of *Drosophila melanogaster* genomes for population genomics analysis. Our studies showed that accurate reads has important implications for assembly, polymorphism detection (especially rare ones), and downstream analysis. We conclude that the algorithms described in this thesis will have direct and immediate impact on such resequencing projects.

[1] `http://1000genomes.com/`
[2] `http://www.1001genomes.org/`
[3] `http://www.dpgp.org/`

Bibliography

[1] O. T. Avery, C. M. MacLeod, and M. McCarty, "Studies on the chemical nature of the substance inducing transformation of pneumococcal types: Induction of transformation by a desoxyribonucleic acid fraction isolated from pneumococcus type iii," *Journal of Experimental Medicine*, vol. 79, no. 2, pp. 137–158, 1944.

[2] S. Batzoglou, D. B. Jaffe, K. Stanley, J. Butler, S. Gnerre, E. Mauceli, B. Berger, J. P. Mesirov, and E. S. Lander, "Arachne: a whole-genome shotgun assembler." *Genome Research*, vol. 12, no. 1, pp. 177–189, January 2002.

[3] D. R. Bentley, "Whole-genome re-sequencing," *Curr. Opin. Genet. Dev.*, vol. 16, pp. 545–552, 2006.

[4] W. Brockman, P. Alvarez, S. Young, M. Garber, G. Giannoukos, W. L. Lee, C. Russ, E. S. Lander, C. Nusbaum, and D. B. Jaffe, "Quality scores and SNP detection in sequencing-by-synthesis systems," *Genome Research*, vol. 18, pp. 763–770, 2008.

[5] J. Butler, I. MacCallum, M. Kleber, I. A. Shlyakhter, M. K. Belmonte, E. S. Lander, C. Nusbaum, and D. B. Jaffe, "ALLPATHS: De novo assembly of whole-genome shotgun microreads," *Genome Research*, vol. 18, no. 5, pp. 810–820, 2008.

[6] M. Chaisson, P. Pevzner, and H. Tang, "Fragment assembly with short reads," *Bioinformatics*, vol. 20, no. 13, pp. 2067–74, 2004.

[7] M. J. P. Chaisson, D. Brinza, and P. A. Pevzner, "De novo fragment assembly with short mate-paired reads: Does the read length matter?" *Genome research*, vol. 19, pp. 336–346, 2009.

[8] F. Chin, H. Leung, W.-L. Li, and S.-M. Yiu, "Finding optimal threshold for correction error reads in DNA assembling," *BMC Bioinformatics*, vol. 10, no. Suppl 1, p. S15, 2009. [Online]. Available: http://www.biomedcentral.com/1471-2105/10/S1/S15

[9] J. F. Degner, J. C. Marioni, A. A. Pai, J. K. Pickrell, E. Nkadori, Y. Gilad, and J. K. Pritchard, "Effect of read-mapping biases on detecting allele-specific expression from RNA-sequencing data." *Bioinformatics*, vol. 25, no. 24, pp. 3207–3212, 2009.

[10] J. C. Dohm, C. Lottaz, T. Borodina, and H. Himmelbauer, "Substantial biases in ultra-short read data sets from high-throughput dna sequencing," *Nucleic Acids Research*, vol. 36, no. 16, p. e105, 2008.

[11] R. Durbin *et al.*, "A map of human genome variation from population-scale sequencing," *Nature*, vol. 467, no. 7319, pp. 1061–1073, 2010.

[12] Y. Erlich, P. Mitra, M. Delabastide, W. McCombie, and G. Hannon, "AltaCyclic: a self-optimizing base caller for next-generation sequencing," *Nat Methods*, vol. 5, pp. 679–682, 2008.

[13] B. Ewing and P. Green, "Base-calling of automated sequencer traces using *phred*. ii. error probabilities," *Genome Res*, vol. 8, no. 3, pp. 186–94, 1998.

[14] B. Ewing, L. Hillier, M. C. Wendl, and P. Green, "Base-calling of automated sequencer traces using *phred*. i. accuracy assessment," *Genome Res*, vol. 8, no. 3, pp. 175–85, 1998.

[15] P. Gajer, M. Schatz, and S. L. Salzberg, "Automated correction of genome sequence errors," *Nucleic Acids Research*, vol. 32, no. 2, pp. 562–569, 2004.

[16] D. Gresham, M. M. Desai, C. M. Tucker, H. T. Jenq, D. A. Pai, A. Ward, C. G. DeSevo, D. Botstein, and M. J. Dunham, "The repertoire and dynamics of evolutionary adaptations to controlled nutrient-limited environments in yeast," *PLoS Genetics*, vol. 4, no. 12, 12 2008.

[17] F. Griffith, "The significance of pneumococcal types," *The Journal of Hygiene*, vol. 27, no. 2, pp. 113–159, 1928.

[18] K. D. Hansen, S. E. Brenner, and S. Dudoit, "Biases in illumina transcriptome sequencing caused by random hexamer priming," *Nucleic Acids Research*, vol. 38, no. 12, p. e131, 2010.

[19] O. Harismendy, P. C. Ng, R. L. Strausberg, X. Wang, T. B. Stockwell, K. Y. Beeson, N. J. Schork, S. S. Murray, E. J. Topol, S. Levy, and K. A. Frazer, "Evaluation of next generation sequencing platforms for population targeted sequencing studies," *Genome Biology*, vol. 10, p. R32, 2009.

[20] I. Hellmann, Y. Mang, Z. Gu, P. Li, F. M. D. L. Vega, A. G. Clark, and R. Nielsen, "Population genetic analysis of shotgun assemblies of genomic sequences from multiple individuals," *Genome Research*, vol. 18, no. 7, pp. 1020–9, 2008.

[21] A. D. Hershey and M. Chase, "Independent functions of viral protein and nucleic acid in growth of bacteriophage," *The Journal of General Physiology*, vol. 36, no. 1, pp. 39–56, 1952.

[22] R. Jiang, S. Tavare, and P. Marjoram, "Population genetic inference from resequencing data," *Genetics*, vol. 181, no. 1, pp. 187–97, 2009.

[23] W.-C. Kao, A. H. Chan, and Y. S. Song, "Echo: A reference-free short-read error correction algorithm," *Genome Research*, 2011, in press.

[24] W.-C. Kao and Y. S. Song, "naivebayescall: An efficient model-based base-calling algorithm for high-throughput sequencing," in *Proc. 14th Annual Intl. Conf. on Research in Computational Molecular Biology (RECOMB)*, 2010.

[25] W.-C. Kao and Y. S. Song, "naivebayescall: An efficient model-based base-calling algorithm for high-throughput sequencing," *Journal Computational Biology*, vol. 18, no. 3, pp. 365–77, 2011.

[26] W.-C. Kao, K. Stevens, and Y. S. Song, "BayesCall: A model-based basecalling algorithm for high-throughput short-read sequencing," *Genome Research*, vol. 19, pp. 1884–1895, 2009.

[27] J. Kiefer, "Sequential minimax search for a maximum," in *Proceedings of the American Mathematical Society*, vol. 4, 1953, pp. 502–506.

[28] M. Kircher, U. Stenzel, and J. Kelso, "Improved base calling for the Illumina Genome Analyzer using machine learning strategies," *Genome Biology*, vol. 10, no. 8, p. R83, 2009.

[29] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[30] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, no. 3, p. R25, 2009.

[31] H. Li, J. Ruan, and R. Durbin, "Mapping short DNA sequencing reads and calling variants using mapping quality scores," *Genome Res.*, vol. 18, pp. 1851–8, 2008.

[32] L. Li and T. Speed, "An estimate of the crosstalk matrix in four-dye fluorescence-based DNA sequencing," *Electrophoresis*, vol. 20, pp. 1433–1442, 1999.

[33] R. Li, Y. Li, K. Kristiansen, and J. Wang, "SOAP: short oligonucleotide alignment program," *Bioinformatics*, vol. 24, no. 5, pp. 713–714, 2008.

[34] J. Martin *et al.*, "The sequence and anslysis of duplication-rich chromosome 16," *Nature*, vol. 432, no. 7020, pp. 988–94, 2004.

[35] P. Medvedev and M. Brudno, "Ab Initio whole genome shotgun assembly with mated short reads," in *Proceedings of the 12th Annual Research in Computational Biology Conference (RECOMB)*, 2008, pp. 50–64.

[36] M. L. Metzker, "Emerging technologies in DNA sequencing," *Genome Res*, vol. 15, no. 12, pp. 1767–76, 2005.

[37] M. Metzker, "Sequencing technologiesthe next generation," *Nature Reviews Genetics*, vol. 11, no. 1, pp. 31–46, 2010.

[38] Y. Peng, H. Leung, S. Yiu, and F. Chin, "IDBA - A Practical Iterative de Bruijn Graph De Novo Assembler," in *Proc. 14th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, ser. Lecture Notes in Computer Science, vol. 6044, 2010, pp. 426–440–440.

[39] P. A. Pevzner, H. Tang, and M. S. Waterman, "An Eulerian path approach to DNA fragment assembly," *Proc Natl Acad Sci*, vol. 98, no. 17, pp. 9748–53, 2001.

[40] W. Qu, S.-i. Hashimoto, and S. Morishita, "Efficient frequency-based de novo short-read clustering for error trimming in next-generation sequencing," *Genome Research*, vol. 19, no. 7, pp. 1309–1315, July 2009.

[41] J. Rougemont, A. Amzallag, C. Iseli, L. Farinelli, I. Xenarios, and F. Naef, "Probabilistic base calling of Solexa sequencing data," *BMC Bioinformatics*, vol. 9, p. 431, 2008.

[42] L. Salmela, "Correction of sequencing errors in a mixed set of reads," *Bioinformatics*, vol. 26, no. 10, pp. 1284–1290, May 2010. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btq151

[43] F. Sanger, S. Nicklen, and A. Coulson, "DNA sequencing with chain-terminating inhibitors," *Proc. Nat. Acad. Sci.*, vol. 74, no. 12, pp. 5463–5467, 1977.

[44] J. Schröder, H. Schröder, S. J. Puglisi, R. Sinha, and B. Schmidt, "SHREC: a short-read error correction method," *Bioinformatics*, vol. 25, no. 17, pp. 2157–2163, 2009.

[45] H. Shi, B. Schmidt, W. Liu, and W. Muller-Wittig, "Accelerating error correction in high-throughput short-read DNA sequencing data with CUDA," in *Proc. IEEE International Parallel & Distributed Processing Symposium*, 2009, pp. 1–8.

[46] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol, "ABySS: a parallel assembler for short read sequence data," *Genome Research*, vol. 19, no. 6, pp. 1117–1123, June 2009. [Online]. Available: http://dx.doi.org/10.1101/gr.089532.108

[47] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, 1981.

[48] A. Sundquist, M. Ronaghi, H. Tang, P. Pevzner, and S. Batzoglou, "Whole-genome sequencing and assembly with high-throughput, short-read technologies," *PLoS One*, vol. 2, no. 5, p. e484, 2007.

[49] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *Information Theory, IEEE Transactions on*, vol. 13, no. 2, pp. 260–269, 1967. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1054010

[50] N. Whiteford, T. Skelly, C. Curtis, M. Ritchie, A. Lohr, A. Zaranek, I. Abnizova, and C. Brown, "Swift: Primary Data Analysis for the Illumina Solexa Sequencing Platform," *Bioinformatics*, vol. 25, no. 17, pp. 2194–2199, 2009.

[51] E. Wijaya, M. C. Frith, Y. Suzuki, and P. Horton, "Recount: expectation maximization based error correction tool for next generation sequencing data," *International Conference on Genome Informatics*, vol. 23, no. 1, pp. 189–201, October 2009.

[52] X. Yang, K. S. Dorman, and S. Aluru, "Reptile: representative tiling for short read error correction," *Bioinformatics*, vol. 26, no. 20, pp. 2526–2533, October 2010.

[53] Z. Yin, J. Severin, M. C. Giddings, W. A. Huang, M. S. Westphall, and L. M. Smith, "Automatic matrix determination in four dye fluorescence-based DNA sequencing," *Electrophoresis*, vol. 17, pp. 1143–1150, 1996.

[54] D. R. Zerbino and E. Birney, "Velvet: Algorithms for de novo short read assembly using de Bruijn graphs," *Genome Research*, vol. 18, no. 5, pp. 821–829, 2008.

Appendix A

Supplementary Material for BayesCall Algorithm

A.1 Proposal Distribution in the Metropolis-Hastings Algorithm

For iteration i of the Metropolis-Hastings algorithm, let $\boldsymbol{\lambda}_k = (\lambda_{1,k}, \ldots, \lambda_{L,k})$ denote the value of $\boldsymbol{\Lambda}_k = (\Lambda_{1,k}, \ldots, \Lambda_{L,k})$ from iteration $i-1$. We first sample a position $x \sim \text{Unif}\{1, 2, \ldots, L\}$ to modify and update $\boldsymbol{S}_{x,k}$ according to

$$\mathbb{P}(\boldsymbol{S}_{x,k} = \boldsymbol{e}_b) \propto (\boldsymbol{X}^{-1}\boldsymbol{I}_{x,k})_b, \tag{A.1}$$

where $\boldsymbol{X}$ is the crosstalk matrix. We wish to update $\Lambda_{x,k}$ according to a proposal distribution that is an approximation to the conditional distribution $\mathbb{P}(\Lambda_{x,k} \mid \boldsymbol{I}_k, \boldsymbol{S}_k, \mathcal{D}_x(\boldsymbol{\lambda}_k))$, where $\mathcal{D}_x(\boldsymbol{\lambda}_k)$ denotes $\boldsymbol{\lambda}_k$ with the xth component $\lambda_{x,k}$ deleted. First we note that

$$\begin{aligned} \mathbb{P}(\Lambda_{x,k} \mid \boldsymbol{I}_k, \boldsymbol{S}_k, \mathcal{D}_x(\boldsymbol{\lambda}_k)) &= \mathbb{P}(\Lambda_{x,k} \mid \boldsymbol{I}_{x-1,k}, \boldsymbol{I}_{x,k}, \boldsymbol{S}_k, \lambda_{x-1,k}, \lambda_{x+1,k}) \\ &= \frac{\mathbb{P}(\Lambda_{x,k}, \boldsymbol{I}_{x-1,k}, \boldsymbol{I}_{x,k}, \boldsymbol{S}_k, \lambda_{x-1,k}, \lambda_{x+1,k})}{\mathbb{P}(\boldsymbol{I}_{x-1,k}, \boldsymbol{I}_{x,k}, \boldsymbol{I}_{x+1,k}, \boldsymbol{S}_k, \lambda_{x-1,k}, \lambda_{x+1,k})}. \end{aligned} \tag{A.2}$$

The denominator in (A.2) will not contribute to the Metropolis-Hasting ratio, so we only need to consider the numerator, which can be decomposed as

$$\begin{aligned} \mathbb{P}(\Lambda_{x,k}, \boldsymbol{I}_{x-1,k}, \boldsymbol{I}_{x,k}, \boldsymbol{S}_k, \lambda_{x-1,k}, \lambda_{x+1,k}) &= \mathbb{P}(\boldsymbol{I}_{x,k} \mid \boldsymbol{I}_{x-1,k}, \boldsymbol{S}_k, \Lambda_{x,k}, \lambda_{x-1,k}) \\ &\quad \times \mathbb{P}(\boldsymbol{I}_{x-1,k} \mid \lambda_{x-1,k}) \\ &\quad \times \mathbb{P}(\lambda_{x+1,k} \mid \Lambda_{x,k}) \\ &\quad \times \mathbb{P}(\Lambda_{x,k} \mid \lambda_{x-1,k}) \\ &\quad \times \mathbb{P}(\lambda_{x-1,k}). \end{aligned} \tag{A.3}$$

The second and the last lines of (A.3) do not contribute to the Metropolis-Hastings ratio, so the terms relevant to our proposal distribution are

$$\mathbb{P}(\boldsymbol{I}_{x,k} \mid \boldsymbol{I}_{x-1,k}, \boldsymbol{S}_k, \Lambda_{x,k}, \lambda_{x-1,k})\mathbb{P}(\lambda_{x+1,k} \mid \Lambda_{x,k})\mathbb{P}(\Lambda_{x,k} \mid \lambda_{x-1,k}) \tag{A.4}$$

$$\propto \exp\left\{-\frac{1}{2\|\Lambda_{x,k}\boldsymbol{S}_k\boldsymbol{Q}_x^w\|_2^2}(\boldsymbol{I}_{x,k}-\boldsymbol{\mu}_{x,k})'\boldsymbol{\Sigma}^{-1}(\boldsymbol{I}_{x,k}-\boldsymbol{\mu}_{x,k})\right\} \tag{A.5}$$

$$\times \exp\left\{-\frac{(\lambda_{x+1,k}-(1-d)\Lambda_{x,k})^2}{2(1-d)^2\Lambda_{x,k}^2\sigma^2}\right\}\exp\left\{-\frac{(\Lambda_{x,k}-(1-d)\lambda_{x-1,k})^2}{2(1-d)^2\lambda_{x-1,k}^2\sigma^2}\right\}. \tag{A.6}$$

Note that $\Lambda_{x,k}$ appears in the denominator of several exponents in (A.6). To simplify the problem, we approximate $\Lambda_{x,k}\boldsymbol{S}_k\boldsymbol{Q}_x^w$ with $\lambda_{x,k}\boldsymbol{S}_k\boldsymbol{Q}_x^w$, and $(1-d)^2\Lambda_{x,k}^2\sigma^2$ with $\lambda_{x+1,k}^2\sigma^2$. The second approximation follows from the defining recursion for $\Lambda_{x,k}$:

$$\Lambda_{x+1,k} = (1-d)\Lambda_{x,k} + (1-d)\Lambda_{x,k}\epsilon,$$

where ϵ is a 1-dimensional Gaussian noise with zero mean and variance σ^2; the approximation corresponds to ignoring the noise term. In summary, our proposal distribution is approximately proportional to

$$\begin{aligned}
&\exp\left\{-\frac{1}{2\|\boldsymbol{z}_{x,k}^w\|_2^2}(\boldsymbol{I}_{x,k}-\boldsymbol{\mu}_{x,k})'\boldsymbol{\Sigma}^{-1}(\boldsymbol{I}_{x,k}-\boldsymbol{\mu}_{x,k})\right\} \\
&\quad\times\exp\left\{-\frac{[\lambda_{x+1,k}-(1-d)\Lambda_{x,k}]^2}{2\lambda_{x+1,k}^2\sigma^2}\right\}\exp\left\{-\frac{[\Lambda_{x,k}-(1-d)\lambda_{x-1,k}]^2}{2(1-d)^2\lambda_{x-1,k}^2\sigma^2}\right\} \\
\propto\ &\exp\left\{-\left[\frac{1}{2(1-d)^2\lambda_{x-1,k}^2\sigma^2}+\frac{(1-d)^2}{2\lambda_{x+1,k}^2\sigma^2}+\frac{(\boldsymbol{X}\boldsymbol{S}_k\boldsymbol{Q}_x^w)'\boldsymbol{\Sigma}^{-1}(\boldsymbol{X}\boldsymbol{S}_k\boldsymbol{Q}_x^w)}{2\|\boldsymbol{z}_{x,k}\|_2^2}\right]\Lambda_{x,k}^2\right. \\
&\quad\left.+\left[\frac{1}{(1-d)\lambda_{x-1,k}\sigma^2}+\frac{(1-d)}{\lambda_{x+1,k}\sigma^2}+\frac{(\boldsymbol{X}\boldsymbol{S}_k\boldsymbol{Q}_x^w)'\boldsymbol{\Sigma}^{-1}(\boldsymbol{I}_{x,k}-\boldsymbol{r}_{x-1,k})}{\|\boldsymbol{z}_{x,k}^w\|_2^2}\right]\Lambda_{x,k}\right\},
\end{aligned}$$

where $\boldsymbol{z}_{x,k}^w := \lambda_{x,k}\boldsymbol{S}_k\boldsymbol{Q}_x^w$ and

$$\boldsymbol{r}_{x-1,k} := \begin{cases} 0, & \text{if } x=1, \\ \alpha(1-d)\boldsymbol{I}_{x-1,k}, & \text{if } x>1. \end{cases}$$

This is a density for the normal distribution with mean $m_{x,k}$ and variance $v_{x,k}^2$ given by

$$\begin{aligned} v_{x,k}^2 &= \left[\frac{1}{(1-d)^2 \lambda_{x-1,k}^2 \sigma^2} + \frac{(1-d)^2}{\lambda_{x+1,k}^2 \sigma^2} + \frac{(\boldsymbol{X}\boldsymbol{S}_k \boldsymbol{Q}_x^w)' \boldsymbol{\Sigma}^{-1} (\boldsymbol{X}\boldsymbol{S}_k \boldsymbol{Q}_x^w)}{\|\boldsymbol{z}_{x,k}\|_2^2} \right]^{-1}, \\ m_{x,k} &= v_{x,k}^2 \left[\frac{1}{(1-d)\lambda_{x-1,k}\sigma^2} + \frac{1-d}{\lambda_{x+1,k}\sigma^2} + \frac{(\boldsymbol{X}\boldsymbol{S}_k \boldsymbol{Q}_x^w)' \boldsymbol{\Sigma}^{-1} (\boldsymbol{I}_{x,k} - \boldsymbol{r}_{x-1,k})}{\|\boldsymbol{z}_{x,k}^w\|_2^2} \right]. \end{aligned}$$

To recapitulate, our proposal distribution for $\Lambda_{x,k}$ is

$$\Lambda_{x,k} | \boldsymbol{S}_k, \lambda_{x-1,k}, \lambda_{x,k}, \lambda_{x+1,k}, \boldsymbol{I}_{x,k}, \boldsymbol{I}_{x-1,k} \sim \mathcal{N}(m_{x,k}, v_{x,k}^2). \tag{A.7}$$

A.2 Parameter Estimation

A.2.1 Details of the expectation-maximization algorithm

Recall that Θ denotes the set $\{p, q, d, \alpha, \sigma^2, \boldsymbol{X}, \boldsymbol{\Sigma}\}$ of parameters in our model. To estimate Θ, we use the EM algorithm with

$$\Theta_i = \operatorname*{argmax}_{\Theta} \mathbb{E}_{\Theta_{i-1}} \left[\log \prod_{k=1}^{K} \mathbb{P}(\boldsymbol{I}_k, \boldsymbol{\Lambda}_k, \boldsymbol{S}_k \mid \Theta) \right],$$

where K denotes the number of clusters used in the parameter estimation, Θ_i denotes the set of parameters in the ith iteration, and the expectation is taken with respect to $\mathbb{P}(\boldsymbol{\Lambda}_k, \boldsymbol{S}_k \mid \boldsymbol{I}_k, \Theta_{i-1})$. Henceforward, we omit the dependence on Θ_{i-1} when writing expectations. In the maximization step, we want to optimize the expected log-likelihood by solving the following equation:

$$\nabla_{\Theta} \mathbb{E} \left[\log \prod_{k=1}^{K} \mathbb{P}(\boldsymbol{I}_k, \boldsymbol{\Lambda}_k, \boldsymbol{S}_k \mid \Theta) \right] = 0. \tag{A.8}$$

By Lebesgue's dominated convergence theorem, we can exchange the order of taking expectation and taking derivatives, so (A.8) is equivalent to

$$\mathbb{E} \left[\nabla_{\Theta} \log \prod_{k=1}^{K} \mathbb{P}(\boldsymbol{I}_k, \boldsymbol{\Lambda}_k, \boldsymbol{S}_k \mid \Theta) \right] = 0.$$

Unfortunately, we could not obtain an analytical solution to the above equation, so we instead used the steepest ascent method, only updating one parameter at a time. Fixing the other parameters in Θ, an analytic solution for $\boldsymbol{X}$, $\boldsymbol{\Sigma}$, σ, and α can be

obtained as follows. First, for ease of notation, define $\mathcal{L} := \log \prod_{k=1}^{K} \mathbb{P}(\boldsymbol{I}_k, \boldsymbol{\Lambda}_k, \boldsymbol{S}_k \mid \Theta)$. Then, note that its partial derivatives can be written as

$$\mathbb{E}\left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}_b}\right] = \mathbb{E}\left[-\sum_{k=1}^{K}\sum_{t=1}^{L} \frac{1}{\|\boldsymbol{Z}_{t,k}^w\|^2} \frac{\partial \boldsymbol{\mu}_{t,k}}{\partial \boldsymbol{X}_b} \boldsymbol{\Sigma}^{-1}(\boldsymbol{I}_{t,k} - \boldsymbol{\mu}_{t,k})\right], \tag{A.9}$$

$$\mathbb{E}\left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}}\right] = \mathbb{E}\left[\sum_{k=1}^{K}\sum_{t=1}^{L} \left(\boldsymbol{\Sigma}^{-1} - \frac{\boldsymbol{\Sigma}^{-1}(\boldsymbol{I}_{t,k} - \boldsymbol{\mu}_{t,k})(\boldsymbol{I}_{t,k} - \boldsymbol{\mu}_{t,k})'\boldsymbol{\Sigma}^{-1}}{\|\boldsymbol{Z}_{t,k}^w\|^2}\right)\right], \tag{A.10}$$

$$\mathbb{E}\left[\frac{\partial \mathcal{L}}{\partial \sigma}\right] = \mathbb{E}\left[\sum_{k=1}^{K}\sum_{t=2}^{L} \left(-\sigma^{-1} + \frac{(\Lambda_{t,k} - (1-d)\Lambda_{t-1,k})^2}{((1-d)\Lambda_{t-1,k}\sigma)^3}\right)\right], \tag{A.11}$$

$$\mathbb{E}\left[\frac{\partial \mathcal{L}}{\partial \alpha}\right] = \mathbb{E}\left[\sum_{k=1}^{K}\sum_{t=1}^{L} \frac{1-d}{\|\boldsymbol{Z}_{t,k}^w\|^2} \boldsymbol{I}'_{t-1,k} \boldsymbol{\Sigma}^{-1}(\boldsymbol{I}_{t,k} - \boldsymbol{\mu}_{t,k})\right], \tag{A.12}$$

where $\boldsymbol{X}_b$ is the bth column of the crosstalk matrix $\boldsymbol{X}$ and $\boldsymbol{\mu}_{t,k}$ is given by

$$\boldsymbol{\mu}_{t,k} = \boldsymbol{X}\boldsymbol{Z}_{t,k}^w + \alpha(1-d)\boldsymbol{I}_{t-1,k}, \tag{A.13}$$

thus yielding

$$\frac{\partial \boldsymbol{\mu}_{t,k}}{\partial \boldsymbol{X}_b} = (Z_{t,k}^w)^b \mathbf{1}, \tag{A.14}$$

where $\mathbf{1} = (1,1,1,1)'$, $(Z_{t,k}^w)^b$ is the bth element of $\boldsymbol{Z}_{t,k}^w$, and $\boldsymbol{Z}_{t,k}^w$ is defined as the zero vector for $t < 1$. Now, set the partial derivatives in (A.9)–(A.12) to zero and solve for $\boldsymbol{X}_b, \boldsymbol{\Sigma}, \sigma^2, \alpha$, respectively, to obtain update rules. For example, the update rule for $\boldsymbol{X}_b$ can be obtained by solving the following equation.

$$\mathbb{E}\left[-\sum_{k=1}^{K}\sum_{t=1}^{L} \frac{1}{\|\boldsymbol{Z}_{t,k}^w\|^2} \frac{\partial \boldsymbol{\mu}_{t,k}}{\partial \boldsymbol{X}_b} \boldsymbol{\Sigma}^{-1}(\boldsymbol{I}_{t,k} - \boldsymbol{\mu}_{t,k})\right] = 0.$$

Now, using (A.13) and writing $\boldsymbol{X}\boldsymbol{Z}_{t,k}^w$ as a linear combination of the columns $\boldsymbol{X}_x$ of $\boldsymbol{X}$, we obtain

$$\mathbb{E}\left[\sum_{k=1}^{K}\sum_{t=1}^{L} \frac{1}{\|\boldsymbol{Z}_{t,k}^w\|^2} \frac{\partial \boldsymbol{\mu}_{t,k}}{\partial \boldsymbol{X}_b} \boldsymbol{\Sigma}^{-1} \left(\boldsymbol{I}_{t,k} - \alpha(1-d)\boldsymbol{I}_{t-1,k} - \sum_{x \in \{A,C,G,T\}} (Z_{t,k}^w)^x \boldsymbol{X}_x\right)\right] = 0, \tag{A.15}$$

Upon moving the terms with $\boldsymbol{X}_b$ to the other side of the equality in (A.15), we obtain

$$\mathbb{E}\left[\sum_{k=1}^{K}\sum_{t=1}^{L}\frac{1}{\|\boldsymbol{Z}_{t,k}^{w}\|^2}(Z_{t,k}^{w})^b\frac{\partial\boldsymbol{\mu}_{t,k}}{\partial\boldsymbol{X}_b}\boldsymbol{\Sigma}^{-1}\boldsymbol{X}_b\right] = \mathbb{E}\left[\sum_{k=1}^{K}\sum_{t=1}^{L}\frac{1}{\|\boldsymbol{Z}_{t,k}^{w}\|^2}\frac{\partial\boldsymbol{\mu}_{t,k}}{\partial\boldsymbol{X}_b}\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{I}_{t,k}-\alpha(1-d)\boldsymbol{I}_{t-1,k}-\sum_{x\in\{A,C,G,T\}\setminus\{b\}}(Z_{t,k}^{w})^x\boldsymbol{X}_x\right)\right].$$

Finally, since both $\boldsymbol{\Sigma}$ and $\boldsymbol{X}_b$ are constant with respect to the expectation, we conclude

$$\boldsymbol{X}_b = \frac{\sum_{k=1}^{K}\sum_{t=1}^{L}\mathbb{E}\left[\left(\boldsymbol{I}_{t,k}-\alpha(1-d)\boldsymbol{I}_{t-1,k}-\sum_{x\in\{A,C,G,T\}\setminus\{b\}}(Z_{t,k}^{w})^x\boldsymbol{X}_x\right)\frac{(Z_{t,k}^{w})^b}{\|\boldsymbol{Z}_{t,k}^{w}\|^2}\right]}{\sum_{k=1}^{K}\sum_{t=1}^{L}\mathbb{E}\left[\frac{((Z_{t,k}^{w})^b)^2}{\|\boldsymbol{Z}_{t,k}^{w}\|^2}\right]}. \tag{A.16}$$

The following update rules for the other parameters can be obtained in a similar fashion:

$$\boldsymbol{\Sigma} = \frac{1}{KL}\sum_{k=1}^{K}\sum_{t=1}^{L}\mathbb{E}\left[\frac{(\boldsymbol{I}_{t,k}-\boldsymbol{\mu}_{t,k})(\boldsymbol{I}_{t,k}-\boldsymbol{\mu}_{t,k})'}{\|\boldsymbol{Z}_{t,k}^{w}\|^2}\right], \tag{A.17}$$

$$\sigma^2 = \frac{1}{K(L-1)}\sum_{k=1}^{K}\sum_{t=2}^{L}\mathbb{E}\left[\frac{(\Lambda_{t,k}-(1-d)\Lambda_{t-1,k})^2}{((1-d)\Lambda_{t-1,k})^3}\right], \tag{A.18}$$

$$\alpha = \frac{\sum_{k=1}^{K}\sum_{t=1}^{L}\mathbb{E}\left[\frac{1}{\|\boldsymbol{Z}_{t,k}^{w}\|^2}\boldsymbol{I}'_{t-1,k}\boldsymbol{\Sigma}^{-1}(\boldsymbol{I}_{t,k}-\boldsymbol{X}\boldsymbol{Z}_{t,k}^{w})\right]}{(1-d)\sum_{k=1}^{K}\sum_{t=1}^{L}\mathbb{E}\left[\frac{1}{\|\boldsymbol{Z}_{t,k}^{w}\|^2}\boldsymbol{I}'_{t-1,k}\boldsymbol{\Sigma}_t^{-1}\boldsymbol{I}_{t-1,k}\right]}. \tag{A.19}$$

(A.20)

The expectations are taken with respect to $\mathbb{P}(\boldsymbol{\Lambda}_k, \boldsymbol{S}_k \mid \boldsymbol{I}_k, \Theta_{i-1})$ and are computed using Monte-Carlo integration with the Metropolis-Hastings algorithm.

A.2.2 Estimation of d, p and q

To estimate d, we adopt a slightly simplified model. Recall that $\boldsymbol{\mu}_{t,k}$ depends on d and that the additive Gaussian noise of $\Lambda_{t,k}$ scales with $(1-d)\Lambda_{t-1,k}$. For parameter estimation, we assume that $\boldsymbol{\mu}_{t,k}$ is independent of d and use

$$\Lambda_{t,k}|\Lambda_{t-1,k} \sim \mathcal{N}((1-d)\Lambda_{t-1,k}, \Lambda_{t-1,k}^2\sigma^2). \tag{A.21}$$

Then, the update rule for d is given by

$$d = 1 - \frac{\sum_{k=1}^{K}\sum_{t=2}^{L}\mathbb{E}\left[\Lambda_{t-1,k}\Lambda_{t,k}\right]}{\sum_{k=1}^{K}\sum_{t=2}^{L}\mathbb{E}\left[\Lambda_{t,k}^2\right]}. \tag{A.22}$$

To estimate the phasing and prephasing parameters p and q, respectively, we adopt an interior point method to optimize the expected log-likelihood directly.

A.2.3 Estimation of cycle-dependent parameters

The estimation of cycle-dependent parameters can be performed similarly. To avoid over-fitting and to reduce the number of clusters required for parameter estimation, we adopt a window-based approach, assuming that the parameters are constant in each window. Specifically, we divide the read length L into non-overlapping windows of size W. For simplicity, assume that L is divisible by W. Then, let $\Omega = \{[i \times W + 1, (i+1) \times W] \mid i = 0, \ldots, \frac{L}{W} - 1\}$ be the set of windows and let Θ_ω denote the set of parameters for window $\omega \in \Omega$. To estimate parameters, we update parameters for one window at a time and iterate through all windows, from left to right. For a given window $\omega = [i, j]$, parameters are estimated using (A.16)-(A.19) and (A.22) with the index t summed over from $i - W$ to $j + W$. For example, rate d_ω for window $\omega \in \Omega$ is estimated by

$$d_\omega = 1 - \frac{\sum_{k=1}^{K}\sum_{t=i-W}^{j+W}\mathbb{E}\left[\Lambda_{t-1,k}\Lambda_{t,k}\right]}{\sum_{k=1}^{K}\sum_{t=i-W}^{j+W}\mathbb{E}\left[\Lambda_{t,k}^2\right]}.$$

Empirically, this method of using information from adjacent windows reduces the fluctuation of parameters between windows.

A.3 Running Time Statistics

The running time of BayesCall depends on several factors, including the number of samples used in the Monte-Carlo algorithms, the number K of read clusters used in parameter estimation, and the number of EM iterations. We have not explored these parameters extensively. A systematic study of the tradeoff between accuracy and running time will be worthwhile. The results reported in this thesis are based on using $K = 250$ and 50 EM iterations in parameter estimation, with the Monte-Carlo algorithms using 10,000 samples for burn-in and 10,000 samples thinned by 10 for estimation.

BayesCall was run on a Mac Pro with two quad-core 3.0 GHz Intel Xeon processors. Using $K = 250$, one iteration of EM took about 25 minutes for the 76-cycle data. We remark that, in BayesCall, the estimation of cycle-dependent parameters

Table A.1: Error rates of BayesCall when K clusters are used in the training set.

K	By-base error rate	By-read error rate
150	0.00790	0.24438
250	0.00783	0.24398
500	0.00783	0.24368

This table shows that the accuracy of basecalls made by BayesCall does not depend much on the number of clusters used to estimate parameters.

can be performed progressively as the data for each cycle become available; that is, we do not need to wait until a sequencing run terminates to estimate parameters for earlier cycles. Since a typical run of the Illumina platform currently takes about 10 days to finish, BayesCall's ability to perform progressive parameter estimation will be of practical value.

We believe that the running time of BayesCall can be sped up considerably while maintaining high accuracy. Since parameter estimation can be done progressively as the sequencing machine runs, we believe that the bottleneck is in basecalling. Our current implementation requires about 18 hours to call bases for 1 million reads of length 76. This slow running time seriously restricts the practicality of BayesCall. However, we remark that the model introduced here can be used to devise a more efficient basecalling algorithm. We are currently investigating methods to achieve faster parameter estimation and basecalling.

A.4 Effects of Using a Different Number K of Clusters in the Training Set

As mentioned above, the results described in Chapter 2.5 are based on using $K = 250$. We also tried using 150 and 500 clusters in the training set. For this study, we considered a single tile from the 76-cycle PhiX174 data set; the resulting test set contained $68,272$ reads and $5,188,672$ bases. Table A.1 shows the error rate of BayesCall when a different number of clusters is used in the training set. Note that the accuracy results for $K = 250$ and $K = 500$ are very similar. Further, using $K = 150$ also seems to produce very good results. The running time of parameter estimation scaled roughly linearly with the number of clusters. These results justify our usage of $K = 250$ in parameter estimation.