# REDUCED REPRESENTATIONS FOR EFFICIENT ANALYSIS OF GENOMIC DATA; FROM MICROARRAY TO HIGH-THROUGHPUT SEQUENCING

**BY MD PAVEL MAHMUD**

**A dissertation submitted to the**

**Graduate School—New Brunswick**

**Rutgers, The State University of New Jersey**

**in partial fulfillment of the requirements**

**for the degree of**

**Doctor of Philosophy**

**Graduate Program in Computer Science**

**Written under the direction of**

**Prof. Alexander Schliep**

**and approved by**

_____

_____

_____

_____

**New Brunswick, New Jersey**

**October, 2014**

## ABSTRACT OF THE DISSERTATION

# Reduced Representations for Efficient Analysis of Genomic Data; From Microarray to High-throughput Sequencing

**by Md Pavel Mahmud**

**Dissertation Director: Prof. Alexander Schliep**

Since the genomics era has started in the '70s, microarray technologies have been extensively used for biological applications such as gene expression profiling, copy number variation (CNV) or Single Neucleotide Polymorphism (SNP) detection. To analyze microarray data, numerous statistical and algorithmic techniques have been developed over the last two decades; specially, for detecting CNV from array comparative genomic hybridization (arrayCGH) data, Hidden Markov Models (HMMs) have been successfully used. Still, due to computational reasons, the benefits of using Bayesian HMMs have been overlooked, and their use has been, at best, minimal in practice. The large demand for computational resources has also affected the analysis of high throughput sequencing (HTS) data, which, over the last few years, has started to revolutionize the field of computational biology. For example, the most sensitive tools for mapping HTS data to reference genomes are generally ignored in favor of fast, less accurate ones.

In this dissertation, we strive for reduced representations of biological data which enable us to perform efficient computations on large datasets. Since biological datasets often contain repetitive, sometimes redundant, elements, it is a natural idea to identify groups of similar elements and directly perform computations on these groups. Usually,

the relevant type of similarity is specific to the type of data and application in hand.

Specifically, we make the following four contributions in this thesis. First, we show that, by exploiting repetition in discrete sequences, Markov Chain Monte Carlo (MCMC) simulations of Bayesian HMM can be accelerated, which can then be applied to the DNA segmentation problem [1]. Second, in case of Gaussian observations representing copy number ratio data, we show that, through pre-computing similar, contiguous observations into blocks, MCMC for Bayesian HMM can be well-approximated [2]. Third, by representing sequences to multi-dimensional vectors, we introduce a nearest neighbor based novel technique for mapping HTS data to reference genome [3]. Finally, we present a highly efficient clustering approach for HTS data, which allows us to speed-up computationally demanding, sensitive tools for mapping HTS data [4].

# Acknowledgements

First of all, I would like to thank Prof. Alexander Schliep for his supervision and support during my PhD studies. He introduced me to the area of genomics and bioinformatics, specially microarray technologies and high-throughput sequencing, and encouraged me to work on challenging state-of-the-art problems. Most of the contributions presented in this dissertation are results of our combined work from 2010 to 2014 [1–4]. Without his guidance and novel ideas this dissertation would not be possible.

I would also like to thank Prof. William Steiger for his support and advices during the initial years of my studies. Being his teaching assistant for many semesters helped me tremendously in staying sharp in probability and statistics, which indirectly contributed to my research in later years. I would also like to thank Prof. Mario Szegedy, Prof. Danfeng Yao and Prof. Tina Eliassi-Rad for their help in various stages of my studies. I would like to express my gratitude to Prof. Martin Farach-Colton and Prof. Kevin Chen for their suggestions regarding this thesis.

I would like to thank my colleagues in Schliep Lab; Jonathan Shao, Eric Brugel, Rajat Roy, John Wiedenhoeft and Ivani Lopes. They often read my manuscripts, provided valuable suggestions, shared excellent ideas and often provided much needed encouragement. I express my sincere gratitude to them, and, specially to John Wiedenhoeft for his contributions in [3].

This work would not be possible without the support and motivation of friends and family. I want to specially thank Khairul Kabir, Ratan Dey and Ashique Reza Shovon for their belief in me. I am very grateful to my sister Nilufar Easmin for always making sacrifices for me. Lastly, I want to acknowledge the tremendous effort that my parents have always made for my well-being. Without their unconditional love, support and sacrifices I would achieve very little in my life.

# Dedication

I dedicate this work to my father, Noor Jalal, my mother, Farida Yasmin and my sister, Nilufar Easmin.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

In this thesis we are concerned with the computational aspects of analyzing genomic data in the form of deoxyribonucleic acid (DNA), array comparative genomic hybridization (arrayCGH) data and high throughput sequencing (HTS) reads. In this chapter, we briefly describe some basic concepts related to DNA and some biotechnological experiments that either directly or indirectly measure quantities of interest from DNA. We also discuss some computational difficulties that arise in analyzing large amount of data using powerful but complex statistical models. We conclude by outlining the contributions of this thesis and summarizing it's content.

## 1.1 DNA

DNA molecule encodes genetic information for most organisms. It has a double helix structure which consists of two long strand of nucleotides. Genetic information is stored using four possible nucleobases–guanine (G), adenine (A), thymine (T) and cytosine (C)–connected to a backbone of sugar-phosphate base (Figure 1.1). In eukaryotes, several DNA molecules, called chromosomes, are present inside a cell nucleous. Typically, each of the chromosomes has millions of base-pairs; the human genome has around 3 billion base-pairs arranged into 23 pair of chromosomes.

The coding region of DNA stores genes, which are hereditary units responsible for an organism's traits and functionalities. A process, known as transcription, generates complementary ribonucleic acid (RNA) from the DNA sequence representing a gene. Then, another process (known as translation), guided by the RNA sequence, creates amino acids, which represent the building block of proteins–responsible for carrying out functions in an organism. Although non-coding regions in a genome do not directly take

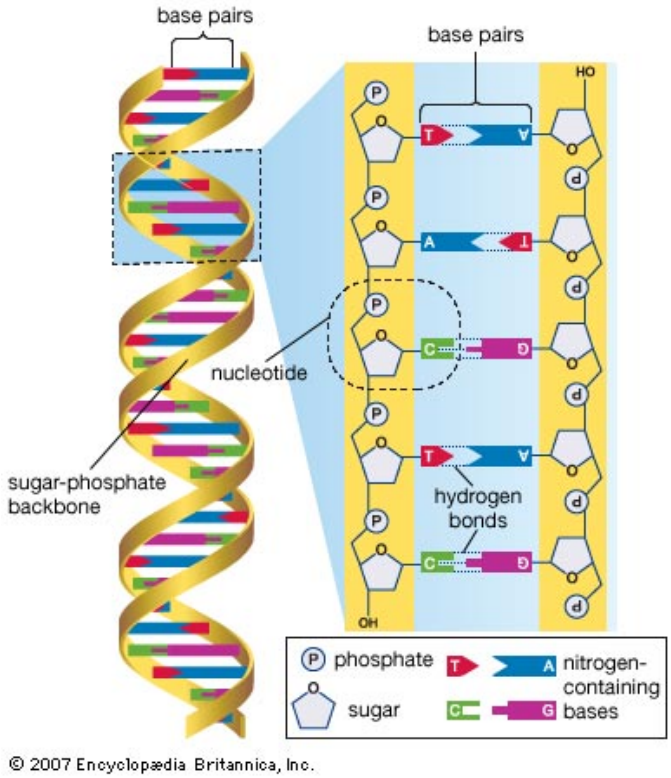



Figure 1.1: **DNA double helix structure**. Two sugar-phosphate backbone holds two complementary sequence of bases attached to each other. Figure reproduced from Encyclopaedia Britannica.

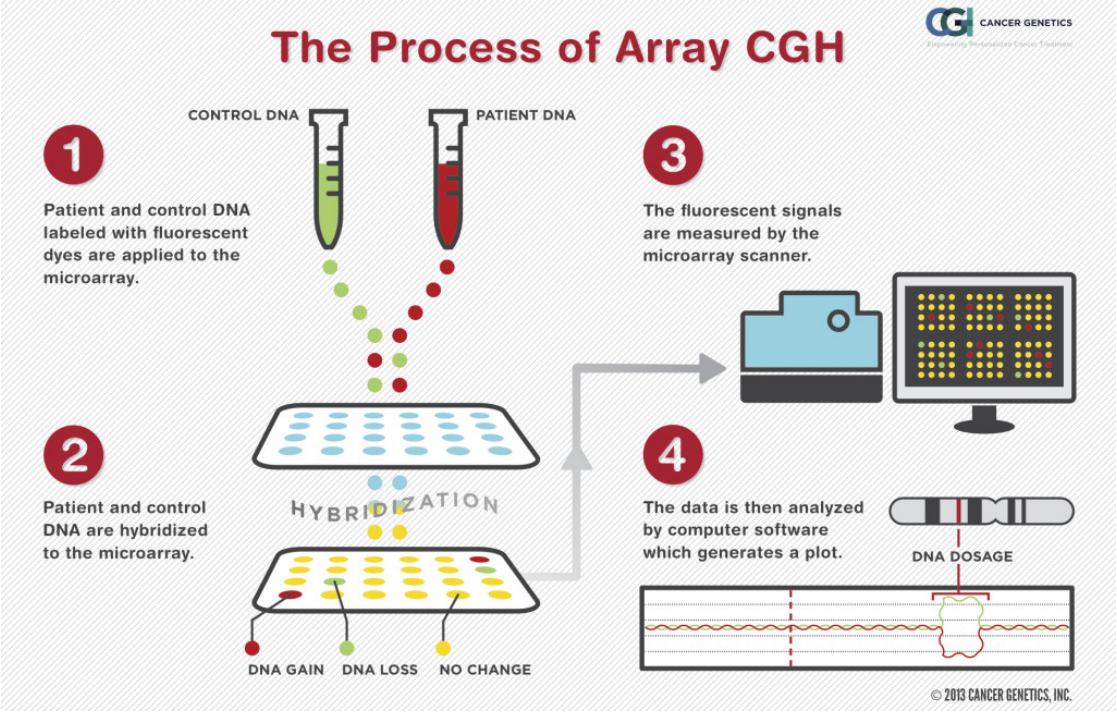



Figure 1.2: **ArrayCGH.** Control and patient DNA with different fluorescent colors get hybridized to the microarray. A high resolution scanner can detect the outcome of hybridization for each probe in the array. Figure reproduced from Cancer Genetics, Inc. (`www.cancergenetics.com`).

part in protein synthesis they also carry out some regulatory functions. Even though molecular biology is a fascinating area of research, in this thesis, our focus is limited to computational aspects of analyzing information extracted from DNA sequences in various forms.

## 1.2 Microarray Experiments

Knowing each base of a DNA sequence is not necessary to answer every biological question. Microarray experiments can be used to measure gene expression, detect SNPs, find copy numbers, etc. from DNA or RNA samples without explicitly reading every base. Here we describe a microarray experiment, known as arrayCGH, to detect copy number variation (CNV) between a patient and a control genome (see Figure 1.2). In this thesis, we will restrict our discussion of microarray technologies to arrayCGH.

In arrayCGH experiment, at first, patient and control DNA are labelled with different fluorescent colors and separated into single strands. Then both sets of DNA materials are brought onto a microarray chip where they hybridize to complementary genetic material known as probes. By reading the fluorescent image, created from fluorescently labelled sample DNA bound to probes, with high resolution scanner we can detect the level of comparative difference between the control and sample DNA. Loss or gain in copy numbers manifest themselves by different colors while hybridization intensity informs about the magnitude of change.

ArrayCGH technology has experienced tremendous success in cancer research and detecting genetic aberrations. The datasets produced by these experiments involve manual intervention, imaging and chemical processes which lead to noisy measurements. As a result, statistical methods are required to analyze the data. In this thesis, we will discuss an alternate representation of arrayCGH data and show that a highly used statistical model can gain significant advantage from that representation.

## 1.3 High Throughput Sequencing (HTS) Experiments

The first methods for DNA sequencing—Maxam-Gilbert sequencing [5] and Sanger sequencing [6, 7]—arrived in the early '70s. These methods had been laborious and highly expensive; The first human genome was sequenced only about a decade ago using Sanger sequencing at a cost of nearly 3 billion dollars [8]. However, in about a decade, the cost of sequencing has gone down a few orders of magnitude, and, just recently, Illumina has announced the first DNA sequencer to sequence human genome at a cost of 1,000$ per genome. In this thesis, we will focus on sequencing data produced by HTS based methods such as Illumina's.

The sequencing process used by Illumina can be described in two steps; DNA fragment library preparation (Figure 1.3) and sequencing by synthesis (Figure 1.4). At first, genomic DNA is randomly sheared into many fragments and unique adapter sequences are attached to the ends. Through hybridization the adapter sequences bind single DNA strands to an array of primers. Then the attached segments are amplified

and clusters of DNA sequences are formed. In the second step, bases with fluorescent labels are added, which attach to the DNA strands in the clusters. Unattached bases are washed away and a high-resolution image informs about the first base of the DNA strands. This step is repeated until the full strands are sequenced. We refer to the collection of sequences generated in this process as HTS *reads*.

Although high-throughput nature of this experiment allows inexpensive sequencing at a very fast rate, due to the nature of the chemical process involved, base call quality degrades over time. As a result, the sequenced reads occasionally contain errors and are reliable only up to a certain length (for Illumina, depending on their specific platform, up to 150-300 bp). This poses some critical challenges in analyzing HTS reads. A fundamental task in bioinformatics is mapping the sequenced reads back to a reference genome–known as the read mapping problem. Without errors, mapping reads would be equivalent to solving exact string matching problem. Because of errors and the sheer scale of the datasets, sophisticated approximate string matching algorithms, known as read mappers, have to be designed. The situation even gets worse due to the genetic differences between the sample genome and the reference genome. In this thesis, we propose a radically different read mapper capable of dealing with large amount of errors and genetic variations, and argue that a different compressive approach is required for computationally demanding read mappers.

## 1.4 Thesis Overview

The main focus of this thesis is to find reduced representations of biological data which enables us to perform efficient computations on large datasets. Since large biological datasets often contain repetitive, sometimes redundant, elements it is a natural idea to identify groups of elements with respect to some similarity criteria and perform computations on these groups. Usually, the type of similarity is specific to the type of data and application for which it will be used. In this thesis, we will explore three fundamentally different kind of data produced from biological experiments and show how a particular representation can be exploited for one or more applications.

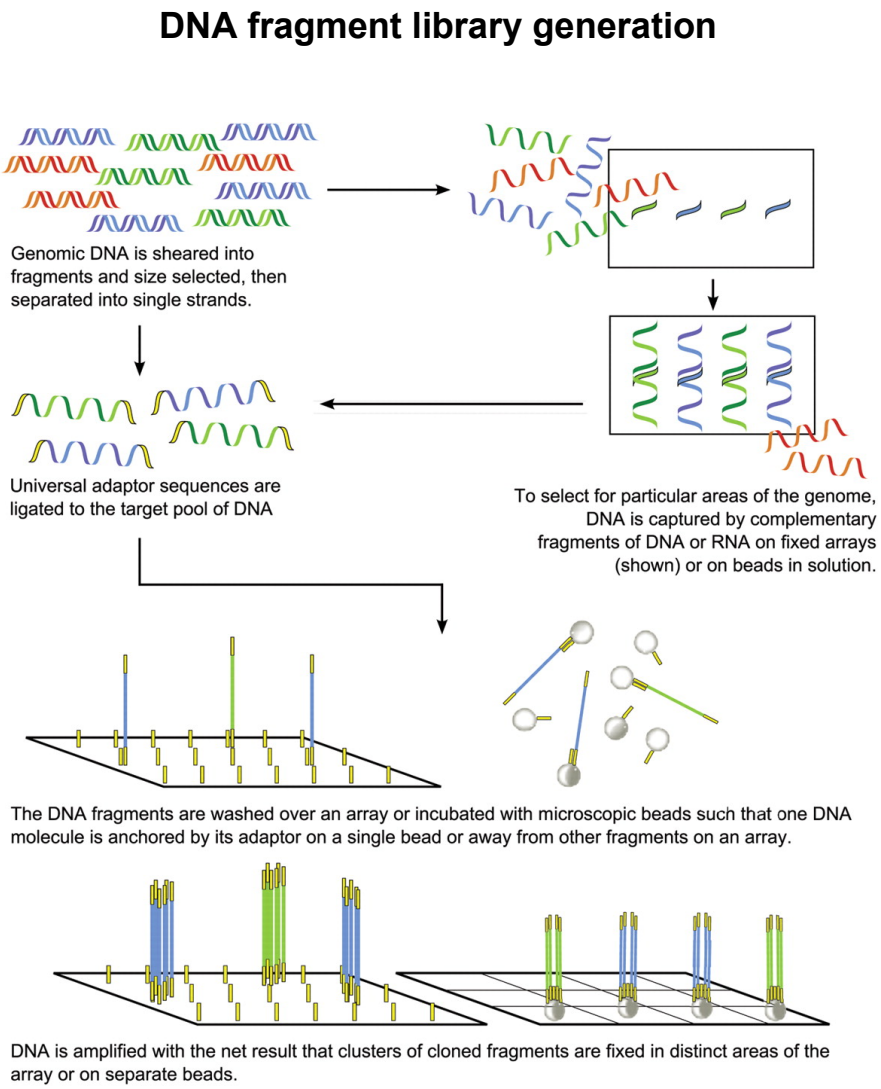



Figure 1.3: **HTS - library preparation.** After shearing DNA into random fragments adapters are attached, which help to bind these fragments onto an array. Some particular DNA fragments can be targeted for sequencing through complementary DNA. In whole genome sequencing, there is no need for this step. The generated fragments are then amplified. Note that beads are used by some other non-Illumina platforms. Figure reproduced from [9].

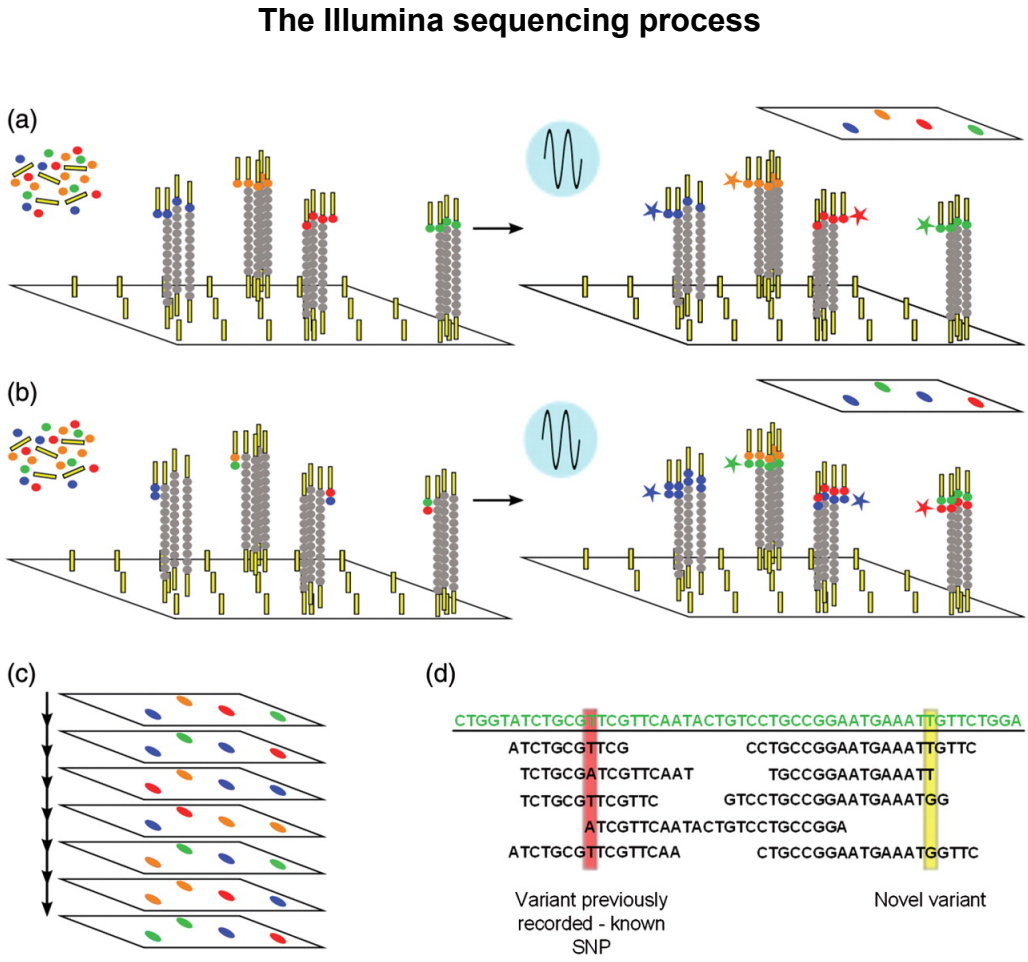



Figure 1.4: **HTS - sequencing.** Bases with fluorescent labels are poured onto the clusters. Through a bio-chemical process the labelled bases get attached to the first base of the strands. A high resolution camera identifies the first bases. This process is repeated for subsequent bases in the strands. Figure reproduced from [9].

In case of a DNA sequence, since it comprises of only four possible bases, some bases may appear together more than once. The longer a sequence is, the greater is the chance of this kind of repetition. This observation has led to efficient computations for Hidden Markov models before [10, 11]. We have extended this approach to even more complex statistical computations. With copy number ratio data from arrayCGH experiments, we shift our focus from the discrete to the continuous domain where exploiting sequence repetition is not an option anymore. Instead we look into the Markov property of the biological process in question. In particular, copy numbers have strong spatial dependency, which can be successfully modeled using HMMs . Moreover, observations in a particular HMM state can be modeled using Gaussian distribution, which, under some assumptions, can be grouped together and represented in a computation-friendly way. Unlike DNA sequence, this reduced representation leads to computations that produce approximate results.

Although high resolution microarray experiments are getting cheaper and statistical analysis of these datasets have improved a lot, the future of biological data analysis belongs to HTS. To overcome sequencing errors and increase statistical confidence of analysis, HTS datasets are often generated with high coverage, which means more redundant reads. This redundancy has been exploited in many applications in a somewhat ad-hoc way. We propose a cluster based approach to produce reduced representations of HTS data that can be directly used for biological applications. We extensively discuss it's applicability to read mapping problem.

In chapter 2, we introduce basic computations using Hidden Markov models, and extensively discuss the details of MCMC approach in the context of Bayesian HMM. We also discuss the basic characteristics of HTS datasets which are relevant in analysis. We start our discussion on this thesis's contribution with the DNA segmentation problem in chapter 3. We motivate the use of Bayesian HMMs for this problem and show how sequence repetition can be exploited to improve MCMC in discrete observation settings. In chapter 4, we use continuous valued observations with Bayesian HMMs for copy number variation detection. We show how similar consecutive observations can be grouped together and exploited for approximating MCMC simulations.

We start our exploration of HTS datasets with a close look at the fundamental problem of read mapping. In chapter 5, we introduce a new read mapper and discuss how the scale and variation of the dataset inspires a different approach in read mapping. We propose in chapter 6 that the massive HTS datasets can be tackled through clustering similar reads together. We show the effect of using a reduced representation of HTS dataset through extensive experimentation on read mapping. Finally, in chapter 7, we conclude with final remarks and future directions.

# Chapter 2

# Hidden Markov Models and HTS Reads

Hidden Markov Model (HMM) is a class of directed graphical models which are specially useful when successive observations are correlated or a Markov process is assumed to have generated the observations. HMMs have been used extensively for sequence classification tasks in many areas including speech recognition [12], natural language processing [13], and bioinformatics [14]. For analyzing biological sequences, HMMs are particularly useful, for example in sequence alignment problems [14], gene finding [15], CpG island detection [16], DNA segmentation [16–19] and promoter detection [20]. In this chapter, we introduce the basic statistical and computational aspects of HMM. All of these discussion has been extensively covered in the literature [21]. Additionally, we introduce basic characteristics of HTS reads which are relevant for analysis.

This chapter is organized as follows: We start with the basic definitions and inference in HMM (Section 2.1.1). Then, we introduce Bayesian HMM and a special kind of MCMC named Forward-backward Gibbs sampling (Section 2.1.2). Finally, we discuss characteristics of HTS reads and some necessary definitions related to their analysis (Section 2.2.2).

## 2.1 Hidden Markov Model

We consider HMM with both discrete and continuous emission distributions; see [22] for an introduction. We will use the following notation: $N$ denotes the number of states, $S = \{s_1, s_2, \ldots, s_N\} \equiv \{1, 2, \ldots, N\}$ the set of states, $\mathcal{O}$ the set of possible observations, $T$ the length of the observation sequence, $O = (o_1, o_2, \ldots, o_T) \in \mathcal{O}^T$ the observation sequence, $Q = (q_1, q_2, \ldots, q_T) \in S^T$ the hidden state sequence, $A = \{a_{i,j}\}_{1 \leq i,j \leq N}$ the transition matrix, $B = \{b_{i,o}\}_{1 \leq i \leq N, o \in \mathcal{O}}$ the emission matrix, and $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$

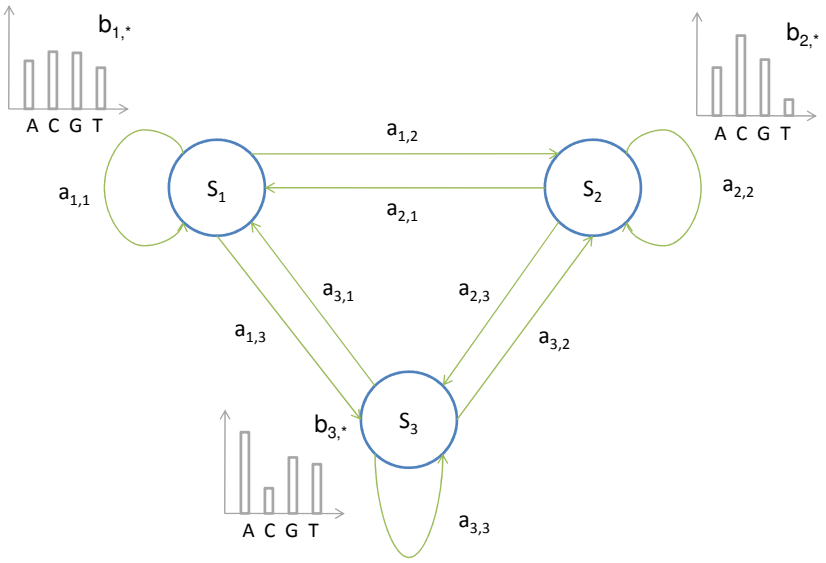



Figure 2.1: Discrete HMM with 3 states, $S_1, S_2$ and $S_3$, over alphabet $\{A, C, G, T\}$. Here $a_{i,j}$ is the probability of making a transition from state $i$ to $j$, and $b_{i,o}$ is the probability of observing the alphabet $o$ in state $i$. $\pi$, the initial distribution over states, is not shown here.

the initial distribution over states; see Figure 2.1 for an example of a 3-state HMM. We represent a consecutive set of observations with $O_{i,j}= o_i, \ldots, o_j$ and, similarly, a consecutive set of state sequences with $Q_{i,j}= q_i, \ldots, q_j$. We will use two different sets for $\mathcal{O}$; for discrete observations $\mathcal{O} = \{A, C, G, T\}$ and for continuous observations $\mathcal{O} = \mathbf{R}$.

The hidden state sequence $Q$ follows a first-order Markov chain and observations solely depends on the current state. The following equations fully describe the behavior of an HMM, represented by $\theta= (A, B, \pi)$.

$$P(q_1) = \pi_{q_1} \ , \tag{2.1}$$

$$P(q_t|Q_{1,t-1}) = P(q_t|q_{t-1}) = a_{q_{t-1},q_t} \ , \tag{2.2}$$

$$P(o_t|q_t) = b_{q_t,o_t} \ . \tag{2.3}$$

Thus, the joint probability of the hidden state sequence $Q$ and the observation sequence $O$, is

$$P(O, Q|A, B, \pi) = \pi_{q_1} b_{q_1,o_1} \prod_{t=2}^{N} a_{q_{t-1},q_t} b_{q_t,o_t} \ . \tag{2.4}$$

### 2.1.1 Inference in HMM

There are three fundamental questions related to HMM. Given a model $\theta = (A, B, \pi)$ and an observation sequence $O$, (1) what is the most likely hidden state sequence (known as Viterbi path) and (2) how likely is the observation sequence? (3) Given $O$, what are the most likely parameters $\theta_{ML}$ of the HMM? We discuss these questions below in detail.

#### Viterbi Path

Given $\theta = (A, B, \pi)$ and $O$, the most likely state sequence $Q_{ML}$ is defined as,

$$Q_{ML} = \underset{Q}{\operatorname{argmax}} P(Q|O, \theta). \tag{2.5}$$

Although there are $O(N^T)$ number of possible state-paths in a HMM, utilizing the first-order property of the Markov chain, a simple dynamic programming algorithm can find out the Viterbi path in $O(TN^2)$ time.

#### Forward Variables

Given $\theta = (A, B, \pi)$, for state $j$ at time $t$, the forward variable $\alpha_t(j)$ is defined as

$$\alpha_t(j) = P(O_{1,t}, q_t = j|\theta) \tag{2.6}$$

$$= \sum_{i=1}^{N} \alpha_{t-1}(i) a_{i,j} b_{j,o_t}. \tag{2.7}$$

It follows from (2.7) that forward variables can be computed iteratively in $O(TN^2)$ time.

**Likelihood**

Given $\theta = (A, B, \pi)$, the likelihood of observing $O$ is

$$P(O|\theta) = \sum_{Q} P(O|Q, \theta)P(Q|\theta) \tag{2.8}$$

$$= \sum_{i=1}^{N} \alpha_T(i) \tag{2.9}$$

Since the likelihood of observing $O$ is a sum over the forward variables at time $t$, it can be computed in $O(TN^2)$.

**Baum-Welch Algorithm**

Given the observation sequence $O$, the most likely model $\theta_{ML}$ is

$$\theta_{ML} = \underset{\theta}{\operatorname{argmax}}\, P(O|\theta), \tag{2.10}$$

which can be computed using an expectation maximization based algorithm known as the Baum-Welch algorithm [23]. If the algorithm converges in $M$ iterations, the total runtime complexity will be $O(MTN^2)$.

### 2.1.2 Bayesian HMM

In Bayesian setting, instead of relying on one point estimate such as $\theta_{ML}$, we model the uncertainty of the parameters using standard conjugate prior distributions; for multinomial likelihood we choose Dirichlet priors, and for Gaussian likelihood we choose Gaussian and Gamma priors for the sufficient statistics—mean and precision respectively. In particular,

$$\pi \sim \text{Dirichlet}(\theta^{\pi}) \tag{2.11}$$

$$A_{i,*} \sim \text{Dirichlet}(\theta^{A_i}), \tag{2.12}$$

for discrete observations,

$$B_{i,*} \sim \text{Dirichlet}(\theta^{B_i}), \tag{2.13}$$

and for Gaussian observations,

$$\mu_i|\sigma_i \sim \mathcal{N}(\tilde{\mu}_i, \tilde{\sigma}_i) \tag{2.14}$$

$$\sigma_i^{-2}|\mu_i \sim \text{Gamma}(a_i, b_i), \tag{2.15}$$

where $\tilde{\mu}_i$, $\tilde{\sigma}_i$, $a_i$, $b_i$, $\theta^{A_i}$, $\theta^{B_i}$, and $\theta^{\pi}$ are the hyperparameters of the model. See [24] and [21] for detail.

**Forward-Backward Gibbs Sampling**

As we are interested in computing the distribution $P(Q|O)$, and a closed form solution of the Bayesian integral $\int_\theta P(Q,\theta|O)\,d\theta = \int_\theta P(Q|\theta,O)\,P(\theta|O)\,d\theta$ is not feasible, the use of MCMC techniques like Gibbs sampling or Metropolis-Hastings becomes mandatory [24, 25].

In Gibbs sampling, we iteratively sample $Q$ and $\theta$ from the conditional distributions $P(\theta|Q,O)$ and $P(Q|\theta,O)$, which creates a Markov chain with the desired distribution $P(Q,\theta|O)$ as its stationary distribution. After an appropriate burn-in period, performing a random walk on the state transition graph, the states of the chain can be used as samples from the stationary distribution [25]. To successfully use Gibbs sampling we need a way to generate samples from the above two conditional distributions. Due to the use of conjugate priors, the posterior probability of $\theta$, $P(\theta|Q,O)$, has the same distribution (with different hyper-parameters) as the prior distribution of $\theta$ (see [24]). In contrast, sampling $Q$ from $P(Q|\theta,O)$ requires a computational approach. Scott [26] compares various techniques for sampling $Q$ and strongly argues in favor of using forward-backward recursions for it's excellent convergence characteristics. Combinedly, we will call this approach forward-backward Gibbs sampling (FBG-sampling) and briefly summarize it for a HMM $\equiv (A, B, \pi) = \theta$ in Alg. 1; see [26, 27] for details.

**Algorithm 2.1** FBG-Sampling($O$)

1: Choose initial parameters $\theta^0 = (A^0, B^0, \pi^0)$.
2: Perform the following steps for $0 \leq m < M$.

1. $Q^m$ = *StateSampler(O, $\theta^m$)* [See Alg. 2.2]
2. Sample HMM parameters,
   $\theta^{m+1} \sim PriorDistribution(hyperparameters, O, Q^m, \theta^m)$

3: **return** $Q^0, Q^1, \ldots, Q^{M-1}$.

FBG-sampling starts with an initial choice of parameters $\theta^0$ and alternatively keeps sampling state sequence $Q^m$ and parameters $\theta^{m+1}$. See [27] for a proof that $Q^m$ returned by Alg. 2.2 is indeed sampled from the marginal distribution $P(Q^m|O, \theta^m)$.

**Algorithm 2.2** StateSampler($O$, $\theta$)

1: **Forward Variables:**

- Compute $\alpha_1(j) = P(o_1, q_1 = j|\theta) = \pi_j b_{j,o_1}$ for all $j$.
- For $2 \leq t \leq T$:
  Compute $\alpha_t(j) = P(O_{1,t}, q_t = j|\theta) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{i,j} b_{j,o_t}$ for all $j$.

2: **Backward Sampling:**

- Sample $q_T$ s.t. $P(q_T = i) \propto \alpha_T(i)$.
- For $T > t \geq 1$:
  Sample $q_t$ s.t. $P(q_t = i) \propto \alpha_t(i) a_{i,q_{t+1}}$.

3: **return** $Q$

Algorithm *StateSampler* uses $O(TN^2)$ space and runs in $O(TN^2)$ time; step 1 (forward variables) runs in $O(TN^2)$ time and step 2 (backward sampling) in $O(T \log N)$. It is obvious from the above algorithm that all the pre-computed forward variables are not used for sampling the state sequence $Q^m$. In the next chapter we will see that even without computing all the forward variables $Q^m$ can be sampled accurately.

## 2.2 HTS Reads

High-throughput sequencing is an emerging technology. Currently, there are at least five different biotechnology companies—454 Life Sciences, Ion Torrent, Pacific Bio, Solid,

and Illumina—marketing high-throughput sequencing machines. The bio-chemical processes involved in these sequencers vary significantly. As a result, the characteristics of the HTS reads also differ among the platforms; see [28,29] for a review. In this thesis, we will restrict our discussion on HTS reads produced by Illumina or Illumina-like sequencing technologies. We choose Illumina because of widespread use and cost-efficiency.

### 2.2.1 Read Characteristics

Although, from computational point, it is convenient to think of reads as a set of random substrings originating from a long genomic string, there are other factors that also affect HTS read analysis. Here we discuss some of the relevant features.

#### Sequencing errors

Mapping reads back to a reference genome is affected by two sources of variation: One, genomic difference—substitutions, indels, etc.—between the sample genome from where HTS reads are sequenced and the reference; two, errors introduced during the sequencing process. Although reads produced by Illumina has non-negligible amount of errors—about 2% per base in HiSeq systems—, it is mostly substitution error rather than indel errors. As a result, the dominant source of indel error in mapping Illumina reads are genetic variants. The methods that we will introduce later in this thesis utilizes this fact. Moreover, these reads have the characteristics that error rate is low in the starting positions but gradually increases towards the end.

#### Quality score

One measure of the reliability of a base call is known as 'Phred quality score', which is reported by the sequencer as $Q_{phred} = -10 \log_{10} e$, where $e$ is the estimated probability of a wrong base call. This extra information about the reliability can be used by methods working on these reads. Additionally, if the sequencer cannot unambiguously resolve a base it reports that base with a special letter $N$.

#### Length

One fundamental characteristics of Illumina reads are their short lengths. Although their most recent platform can produce reads of length 300 bp, reads from the most popular platform HiSeq2500 are limited to 150 bp. Short reads present two difficulties in read mapping: One, due to repetitiveness of complex genomes finding the optimal mapping location of a short read is often very difficult; two, computational tools usually cannot overcome the impact of sequencing errors in short reads. However, Illumina is gradually increasing the maximum read length of their sequencers.

#### Coverage

To increase the probability of reliable mapping and, in general, the power of statistical analysis, HTS reads are usually sequenced with coverage more than one. Here coverage $C$ means that any random base of the sample genome is expected to be covered by $C$ number of reads. Based on some parameters usually the sequencing process allows one to vary $C$ within a range. Along with other advantages, higher coverage allows us to find genetic variants with more confidence. Unfortunately, it also means more computation for downstream analysis tools.

#### Paired-end reads

So far we have discussed reads as a set of independent random samples from the sample genome. The reads produced by this process are known as single-end reads. Illumina, and others, also produce a different kind of reads known as paired-end reads. In this process, instead of reading a genomic segment from one end, both ends are simultaneously read and reported as a pair. This pairing information can be exploited in read mapping to overcome sequencing errors and ambiguity in mapping or alignment.

### 2.2.2 Genomic Characteristics

Previously, we have mentioned that microarray based arrayCGH technologies are limited to a few specific type of large variations between genomes. HTS reads expand

our capability to detect a wider range of small to large variations. In particular, one base pair differences between genomes (known as SNPs), short indels and inversions present considerable challenges to analysis, but ultimately can be detected using HTS reads. Another factor that affects analysis is the complexity of organism's genomes. A genome is not a random set of nucleotides. It consists of many sub-structures such as genes, non-coding regions, etc., which, due to evolutionary pressure and other biological reasons, are maintained and sometimes repeated. Mapping reads back to a reference genome is often complicated due to the nature of these structures; for example, for most tasks, analyzing bacterial genomes are comparatively easy to human genome.

# Chapter 3

# Exploiting Repetition in Discrete Sequences

Although the number of bases in a DNA sequence significantly varies between species—from thousands of bp in bacterial genomes to billions of bp in the human genome [8]—,it consists of only four bases–guanine (G), adenine (A), thymine (T), and cytosine (C). Due to the small finite alphabet size, as a consequence of the pigeon hole principle, some combination of bases must be repeated in a long sequence. This observation has led to identification of repeated contiguous sub-sequences in DNA and their utilization for computational benefits. As a contribution of this thesis we show an application of this observation to Bayesian HMMs in the context of DNA segmentation problem.

HMMs have been used extensively for biological sequence classification tasks such as sequence alignment problems [14], gene finding [15], CpG island detection [16], DNA segmentation [16–19], and promoter detection [20]. These application problems all lead to the computational task of segmenting the input, an observation sequence, based on the most likely assignment of hidden states. Specifically, in the context of DNA segmentation problem [18,30–32], a *segment* is defined to be a contiguous region of DNA sequence, where nucleic acid composition is assumed to follow the same distribution. For example, isochore classes can be identified by solving the DNA segmentation problem using HMMs [16]; see [30, 31] for a fully Bayesian approach.

For simplicity and efficiency reasons, point estimates such as maximum likelihood (ML) or maximum a posterior (MAP), computed with Baum-Welch [23] and variants, have traditionally been used for learning HMM parameters. Based on these estimates segmentations have been computed with the Viterbi path. This ignores uncertainty in model parameters and consequently predictions based on ML or MAP trained models often turn out to be inferior in practice. In contrast, a full Bayesian approach integrates

out model parameters and thus removes dependency on one parameter estimate to improve HMM based prediction. As closed form solutions are not available for HMMs, one frequently uses Markov Chain Monte Carlo (MCMC) sampling techniques like Gibbs sampling or Metropolis-Hastings [24] instead of integration. Forward-backward Gibbs sampling [26, 27], a particular form of Gibbs sampling for HMM (see section 2.1.2), is popular in several communities [33–38] for its improved convergence rate through use of forward and backward recursions. However, depending on the problem, forward-backward Gibbs sampling can still take many iterations to converge. Careful choice of prior distributions and corresponding hyper-parameters can sometimes increase the convergence rate but it remains computationally inefficient compared to using point estimates. In this chapter, we propose a method to speedup an iteration of MCMC sampling by exploiting sequence repetitions.

This chapter is organized as follows. First, we discuss related work and then introduce an extension of classical HMM where observations are time dependent. Then, in Section 3.3 we discuss a simple compression technique known as Four Russian's method and show how compressed observations can be used in MCMC. In Section 3.4 we evaluate our method and show the computational benefits achieved using repetitive discrete observations in compressed form.

## 3.1 Related Work

Using text compression techniques (LZ78, byte pair encoding, four Russians, etc.), Mozes *et al.* [10, 11] have exploited repetitions in long biological sequence to improve the running time of the Viterbi algorithm which computes the most likely hidden state sequence given the observation sequence as well as forward, backward algorithms. Their main idea is to find contiguous repetitive sub-sequences and pre-compute all quantities of interest for these sub-sequences so that these quantities can be used multiple times without repeating the computation. Mozes *et al.* have shown that, despite being one of the simplest compression techniques, the four Russians method yields a logarithmic improvement over the traditional Viterbi algorithm. That the four Russians method

improves dynamic programming algorithms for other applications has been shown previously [39–42]. Moreover, Mozes *et. al.* have shown that for an HMM with few states Baum-Welch training can be improved using partially computed forward and backward variables.

While [10, 11] shows asymptotic speed up for the Viterbi algorithm and improved Baum-Welch training, we focus on Bayesian analysis of HMM using MCMC simulations. Following their idea, we pre-compute quantities of interest for all possible $\log T$-sized sub-sequences (Note: in the following we assume sub-sequences to be contiguous) and use these quantities to compute $O\left(\frac{T}{\log T}\right)$ forward variables. While forward-backward Gibbs sampling needs $T$ forward variables, we show that, because of the conditional dependency structure in an HMM, one can use the partially computed forward variables to implement a modified, but exact, version of forward-backward Gibbs sampling. As forward variable computations dominate the running time we achieve a $O(\log T)$ speed-up.

## 3.2 Time Dependent HMMs

We will follow the definitions related to HMM from Section 2.1. Additionally, we introduce some new definitions for time-dependent observations inspired by their application in DNA segmentation problem [30, 31]. In particular, we define $\gamma$ as the order of the observation process and re-define the emission matrix as $B = \{b^{\beta}_{i,j}\}$, where $\beta \in [\Sigma \cup \Sigma^2 \ldots \cup \Sigma^{\gamma}], 1 \leq i \leq N, 1 \leq j \leq |\Sigma|$. The hidden state sequence $Q$ still follows a first-order Markov chain but, in contrast to the usual literature, where emissions only depend on the state, we consider the case of higher order emissions [43]. Specifically, the probability of an observation sequence $o_t$ is described using the following equation

$$P(o_t|Q_{1,t}, O_{1,t-1}) = P(o_t|q_t, O_{\max(1,t-\gamma),t-1}) = b^{o_t^{'}}_{q_t,o_t}, \tag{3.1}$$

where $o_t^{'} = O_{\max\{1,t-\gamma\},t-1}$, in other words, $o_t^{'}$ is the sequence of previous $\gamma$ observations before time $t$. Fig. 3.1 shows the dependency structure in HMM using graphical models for regular ($\gamma = 0$) and first-order ($\gamma = 1$) emission HMMs.

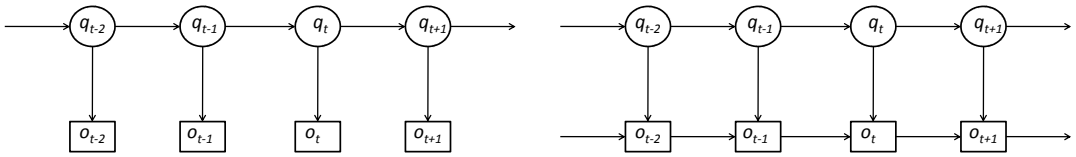



Figure 3.1: Graphical model showing conditional dependency for HMM; $\gamma = 0$ (left) and $\gamma = 1$ (right). An arrow from $X$ to $Y$ means $Y$ is dependent on $X$.

## 3.3 Fast Sampling Using Four Russian's Method

In this section we will present a modified version of algorithm 2.2 taking compressed observations into account. We start with reformulating the forward variables $\alpha$ using matrix notation following [10,11]. Let $M^u(v)$, where $u \in [\Sigma \cup \Sigma^2 \cup \ldots \cup \Sigma^\gamma]$ and $v \in \Sigma$, be a $N \times N$ matrix with elements $M^u_{i,j}(v) = a_{i,j}b^u_{j,v}$. Forward variables at time $t$, $\alpha_t$, can be rewritten as a row vector,

$$\alpha_t = \pi \cdot M^{o_1'}(o_1) \cdot M^{o_2'}(o_2) \cdot \cdots \cdot M^{o_{t-1}'}(o_{t-1}) \cdot M^{o_t'}(o_t) \tag{3.2}$$

$$= \alpha_{t-1} \cdot M^{o_t'}(o_t) \ . \tag{3.3}$$

It is important to note that the matrix formulation does not change the running time of the algorithm.

### 3.3.1 Compression and Forward Variables

We define the matrix $M^{o_i'}(O_{i,j})$ as

$$M^{o_i'}(O_{i,j}) = M^{o_i'}(o_i) \cdot M^{o_{i+1}'}(o_{i+1}) \cdot \cdots \cdot M^{o_{j-1}'}(o_{j-1}) \cdot M^{o_j'}(o_j) \ . \tag{3.4}$$

We assume that the length of the observation sequence, $T$, is a multiple of $k$ such that $d = \frac{T}{k}$ and create groups of fixed size from the observation sequence $O = O_{1,k} \ldots O_{(d-1)k+1,dk}$. Pre-computing all possible matrices $M(X)$, where $|X| \leq k$, for future use is informally

known as the *four Russians method*. Now $\alpha_{lk}$ can be expressed using (3.4) as

$$\alpha_{lk} = \pi \cdot M^{o_1^{'}}(O_{1,k}) \cdot M^{o_{k+1}^{'}}(O_{k+1,2k}) \cdot \cdots \cdot M^{o_{(l-1)k+1}^{'}}(O_{(l-1)k+1,lk}) \tag{3.5}$$

$$= \alpha_{(l-1)k} \cdot M^{o_{(l-1)k+1}^{'}}(O_{(l-1)k+1,lk}) \ . \tag{3.6}$$

The compressed sequence allows us to skip computing forward variables inside a group, which results in significant time savings. Note that we cannot directly use the backward sampling in Alg. 2.2 in this setting. In the remaining part of this section we will explain how we can overcome this problem.

### 3.3.2 Backward-forward State Sequence

Now we will modify the order of state sampling, turning *backward sampling* step of Alg. 2.2 into *backward-forward sampling*, and express the distribution $P(Q|O,\theta)$ in a way that helps us to sample $Q$ accurately and efficiently. We write

$$P(Q|O,\theta) = \underbrace{P(Q_{1,k-1}|Q_{k,T},O,\theta)}_{\text{Part A}} \underbrace{P(Q_{k,T}|O,\theta)}_{\text{Part B}} \ . \tag{3.7}$$

By repeated application of Bayes theorem we can show that part $B$ is proportional to

$$\underbrace{P(q_T|O,\theta)}_{\text{Part } B_1} \prod_{\substack{d \geq i \geq 2 \\ s=(i-1)k \\ e=ik}} \left( \underbrace{P(q_s|Q_{e,T},O,\theta)}_{\text{Part } B_2} \prod_{j=s+1}^{e-1} \underbrace{P(q_j|Q_{s,j-1},Q_{e,T},O,\theta)}_{\text{Part } B_3} \right) . \tag{3.8}$$

Part $B_1$, $B_2$, and $B_3$ can be sampled using the following relations.

**Sampling $B_1$:**

$$P(q_T|O,\theta) \propto P(q_T,O|\theta)$$

$$\propto \alpha_T(q_T) \tag{3.9}$$

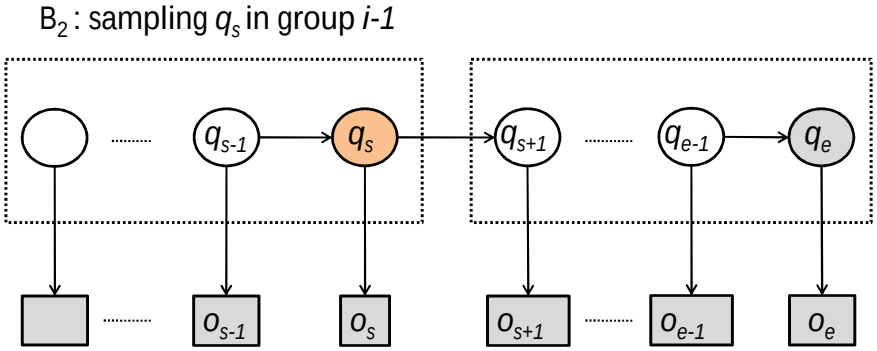



Figure 3.2: Conditional dependency shown for sampling $q_s$ using $B_2$ for $\gamma = 0$. Lightly shaded variables are either observed or already sampled. Dashed rectangle represents a group of observations.

**Sampling $B_2$:**

$$
\begin{aligned}
& P(q_s|Q_{e,T}, O, \theta) \\
& = P(q_s|Q_{e,T}, O_{1,s}, O_{s+1,T}, \theta) \\
& \propto P(q_s|O_{1,s}, \theta) P(O_{s+1,T}, Q_{e,T}|q_s, O_{1,s}, \theta) && (3.10) \\
& = P(q_s|O_{1,s}, \theta) P(O_{s+1,T}, Q_{e,T}|q_s, o'_{s+1}, \theta) && (3.11) \\
& \propto P(q_s, O_{1,s}|\theta) P(O_{s+1,e}, O_{e+1,T}, q_e, Q_{e+1,T}|q_s, o'_{s+1}, \theta) && (3.12) \\
& = \alpha_s(q_s) P(O_{s+1,e}, q_e|q_s, o'_{s+1}, \theta) P(O_{e+1,T}, Q_{e+1,T}|q_s, O_{s+1,e}, q_e, o'_{s+1}, \theta) && (3.13) \\
& = \alpha_s(q_s) P(O_{s+1,e}, q_e|q_s, o'_{s+1}, \theta) P(O_{e+1,T}, Q_{e+1,T}|O_{s+1,e}, q_e, o'_{s+1}, \theta) && (3.14) \\
& \propto \alpha_s(q_s) P(O_{s+1,e}, q_e|q_s, o'_{s+1}, \theta) && (3.15) \\
& = \alpha_s(q_s) M^{o'_{s+1}}_{q_s,q_e}(O_{s+1,e}) && (3.16)
\end{aligned}
$$

Equation (3.10), (3.12), and (3.13) are derived from Bayes theorem. The conditional dependency structure of the HMM given $Q_{e,T}$ (see Fig. 3.2) is used in (3.11) and (3.14). As the last term in (3.14) is independent of $q_s$ it is dropped in (3.15).

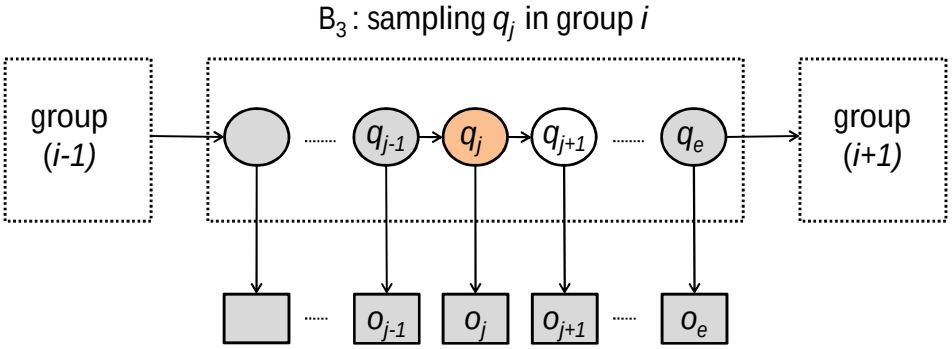



Figure 3.3: Conditional dependency shown for sampling $q_j$ using $B_3$ for $\gamma = 0$. Lightly shaded variables are either observed or already sampled. Dashed rectangle represents a group of observations.

**Sampling $B_3$:**

$$
\begin{aligned}
& P(q_j|Q_{s,j-1}, Q_{e,T}, O, \theta) \\
& \propto P(q_j, o_j, Q_{e,T}, O_{j+1,T}|Q_{s,j-1}, O_{1,j-1}, \theta) \\
& = P(q_j, o_j|Q_{s,j-1}, O_{1,j-1}, \theta) P(Q_{e,T}, O_{j+1,T}|Q_{s,j}, O_{1,j}, \theta) && (3.17) \\
& = P(q_j, o_j|q_{j-1}, o_j^{'}, \theta) P(Q_{e,T}, O_{j+1,T}|q_j, o_{j+1}^{'}, \theta) && (3.18) \\
& = P(q_j, o_j|q_{j-1}, o_j^{'}, \theta) P(q_e, O_{j+1,e}, Q_{e+1,T}, O_{e+1,T}|q_j, o_{j+1}^{'}, \theta) \\
& = P(q_j, o_j|q_{j-1}, o_j^{'}, \theta) P(q_e, O_{j+1,e}|q_j, o_{j+1}^{'}, \theta) P(Q_{e+1,T}, O_{e+1,T}|q_e, O_{j+1,e}, q_j, o_{j+1}^{'}, \theta) \\
& && (3.19) \\
& = P(q_j, o_j|q_{j-1}, o_j^{'}, \theta) P(q_e, O_{j+1,e}|q_j, o_{j+1}^{'}, \theta) P(Q_{e+1,T}, O_{e+1,T}|q_e, o_{j+1}^{'}, \theta) && (3.20) \\
& \propto P(q_j, o_j|q_{j-1}, o_j^{'}, \theta) P(q_e, O_{j+1,e}|q_j, o_{j+1}^{'}, \theta) && (3.21) \\
& = M_{q_{j-1},q_j}^{o_j^{'}}(o_j) M_{q_j,q_e}^{o_{j+1}^{'}}(O_{j+1,e}) && (3.22)
\end{aligned}
$$

Equation (3.17) and (3.19) are derived from Bayes theorem. The conditional dependency structure of the HMM given $Q_{s,j-1}$ and $Q_{e,T}$ (see Fig. 3.3) is used in (3.18) and (3.20). As the last term in (3.20) is independent of $q_j$ it is dropped in (3.21).

### 3.3.3 Fast Sampling Algorithm

Now we formally describe the algorithm *FastStateSampler* (see Alg. 3.1) and analyze its running time. Instead of using Alg. 2.2 (*StateSampler*) in step *2.a* of Alg. 2.1 (*FBG-sampling*) now we can use Alg. 3.1 for fast MCMC simulations.

In the *Precompute* step of Alg. 3.1, $M^{\beta}(X)$ matrices, which are required in (3.16) and (3.22), are computed at first. To sample $q_j$ using (3.22) (in *Backward-forward Sampling* step) we need to compute $M^{o_j'}_{q_{j-1},q_j}(o_j)M^{o_{j+1}'}_{q_j,q_e}(O_{j+1,e})$ for all possible values of $q_j$, which is an $O(N)$ operation. Considering these quantities as weights for possible states we can select $q_j$ using weighted random sampling, which again takes $O(N)$ time. Interestingly, these quantities are already precomputed as intermediate parts of $M^{\beta}(O_{j,e})$. Instead of simply storing these weights, if we store the sum of these values from state 1 to $c$ in $R^{\beta}_{q_{j-1},q_e,c}(o_j, O_{j+1,e})$, we can use binary search to select $q_j$ in $O(\log N)$ time. Similarly, we store the sum of intermediate parts of the variables $\alpha$ and $\delta$ to sample $q_s$ using binary search.

**Running Time:**

As there are at most $2|\Sigma|^{k+\gamma}$ matrices to be precomputed, the pre-computation step takes $O(2|\Sigma|^{k+\gamma}N^3)$ time. Forward variables are computed in $O(\frac{T}{k}N^2)$ time. Using the stored values in $R$ and $\delta$, the state sequence is sampled in $O(T \log N)$ time (the small portion where Alg. 2.2 is used does not affect the order of the algorithm). The total running time is $O(2|\Sigma|^{k+\gamma}N^3 + \frac{T}{k}N^2 + T\log N)$. If $k$ is chosen to be $\frac{1}{2}\log_{|\Sigma|} T - \gamma$, the total running time becomes $O(2\sqrt{T}N^3 + \frac{2TN^2}{\log_{|\Sigma|} T - \gamma} + T\log N)$. Assuming $N < \frac{\sqrt{T}}{\log_{|\Sigma|} T - \gamma}$, *FastStateSampler* achieves a speed-up of $\Theta(\log_{|\Sigma|} T - \gamma)$ and uses $O(\frac{T}{\log_{|\Sigma|} T - \gamma}N^2)$ space.

## 3.4 Experiments

In this section we apply our fast sampling technique to a Bayesian analysis of DNA segmentation. We compare the performance of our method on the DNA segmentation problem with standard FBG-sampling.

**Algorithm 3.1** FastStateSampler($O$, $\theta$)

1: **Precompute:**

- $M^{\beta}(X)$ for all $X \in \cup_{i=1}^{k}\Sigma^i$ and $\beta \in \cup_{i=1}^{\gamma}\Sigma^i$.
- $R^{\beta}(x, X)$ for all $\beta \in \cup_{i=1}^{\gamma}\Sigma^i$, $x \in \Sigma$, and $X \in \cup_{i=1}^{k-1}\Sigma^i$ such that
  - $R^{\beta}_{i,j,1}(x, X) = M^{\beta}_{i,1}(x)M^{(\beta_{2\ldots|\beta|},x)}_{1,j}(X)$.
  - $R^{\beta}_{i,j,c}(x, X) = R^{\beta}_{i,j,c-1}(x, X) + M^{\beta}_{i,c}(x)M^{(\beta_{2\ldots|\beta|},x)}_{c,j}(X)$ for $1 < c \leq N$.

2: **Forward Variables:**

- Compute $\alpha_k = \pi M^{0'_1}(O_{1,k})$.
- For $1 < i \leq m$ and $1 \leq j \leq N$, compute $\alpha_{ik}$ and $\delta_{ik,j,*}$ in the following way.
  - $\delta_{ik,j,1} = \alpha_{(i-1)k}(1)M^{o'_{(i-1)k+1}}_{1,j}(O_{(i-1)k+1,e})$.
  - $\delta_{ik,j,c} = \delta_{ik,j,c-1} + \alpha_{(i-1)k}(c)M^{o'_{(i-1)k+1}}_{c,j}(O_{(i-1)k+1,e})$ for $1 < c \leq N$.
  - Set $\alpha_{ik}(j) = \delta_{ik,j,N}$.

3: **Backward-forward Sampling:**

- Sample $q_T$ from (3.9).
- For $m \geq i \geq 2$:
  - Let $s = (i-1)k$ and $e = ik$.
  - Sample $q_s$ from (3.16) by applying binary search on the monotonically increasing sequence $\delta_{s,q_e,1}, \delta_{s,q_e,2}, \ldots, \delta_{s,q_e,N}$.
  - For $s < j < e$, sample $q_j$ from (3.22) by applying binary search on the monotonically increasing sequence $R^{o'_j}_{q_{j-1},q_e,1}(o_j, O_{j+1,e})$, $R^{o'_j}_{q_{j-1},q_e,2}(o_j, O_{j+1,e})$, $\ldots$, $R^{o'_j}_{q_{j-1},q_e,N}(o_j, O_{j+1,e})$.
- Given $q_k$, sample $Q_{1,k-1}$ (part $A$) using a slightly modified version of Alg. 2.2.

4: **return** $Q$

We measure the running time of forward-backward Gibbs sampling (Alg. 2.1) using both Alg. 2.2 and Alg. 3.1 as the sampler in step *2.a*. The running time of forward-backward Gibbs sampling is proportional to the number of sampling iterations $M$ (see step *2* of Alg. 2.1). We set $M = 10$ and compare execution time of one run of the algorithms. In [30] Boys *et. al.* used $500,000$ iterations to segment *Bacteriophage lambda* DNA. They showed that $6 \leq N \leq 8$ and $0 \leq \gamma \leq 2$ produced the best segmentation for *Bacteriophage lambda*. Unlike their model we keep $\gamma$ and $N$ fixed, but it can easily be modified to variable model dimensions. Four bacterial genomes — *Bacteriophage lambda* (genome size 0.05 Mbp), *Mycoplasma leachii* (1 Mbp), *Planctomyces brasiliensis* (6 Mbp), and *Sorangium cellulosum* (13 Mbp) — are segmented and the running time for different choices of $N$ and $\gamma$ are shown in Fig. 3.4. As both algorithms converge to the same stationary distribution we do not report any segmentation error.

As expected, we see logarithmic speed-up for our method over standard FBG-sampling (see Table 3.1). As the size of the dataset increases, so does the speed-up we observe. For *Sorangium cellulosum* we achieve a speed-up of 5. For small values of $N$, the state path sampling time is comparable to the pre-computation and forward variable computation time. As a result there is no significant speed-up for small $N$. For very large $N > \frac{\sqrt{T}}{\log_{|\Sigma|} T - \gamma}$ (often impractical) the algorithm gradually loses it's advantage over standard FBG-sampling. However, this bound and overall running time can be improved by computing $M^{\beta}(X)$ matrices using fast matrix multiplication of order $o(N^3)$.

We implemented the algorithms in C++ and tested in a Linux machine with a 2.2 GHz AMD Opteron processor. As there was very little variation between the running time of two different runs of an algorithm, instead of averaging over multiple runs, we report the running time of one single run in Fig. 3.4.

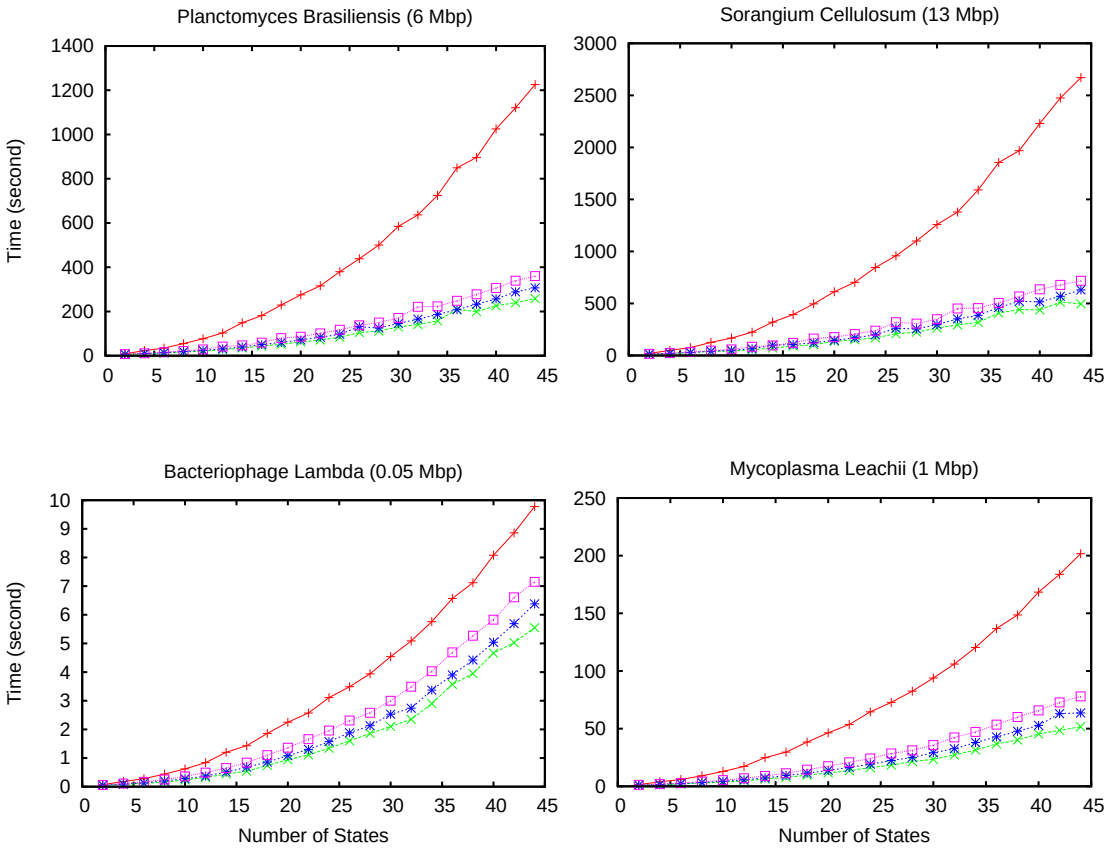



Figure 3.4: Running time comparison on four datasets. Execution times for forward-backward Gibbs (red, +) and four Russians method ($\gamma = 0$ with (green, $\times$), $\gamma = 1$ with (blue, $*$), and $\gamma = 2$ with (pink, $\square$)) are shown.

Table 3.1: Speed-up using fast sampling method for HMM with $\gamma = 0, 1, 2$.

| Dataset | Order ($\gamma$) | Number of States ($N$) | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 |
| *B. lambda (0.05 Mbp)* | 0 | 2.2 | 2.8 | 2.6 | 2.6 | 2.4 | 2.3 | 2.1 | 2.2 | 1.8 | 1.7 | 1.8 |
| | 1 | 2.0 | 2.2 | 2.3 | 2.2 | 2.1 | 2.0 | 1.9 | 1.9 | 1.7 | 1.6 | 1.5 |
| | 2 | 1.8 | 1.8 | 1.8 | 1.8 | 1.6 | 1.6 | 1.6 | 1.5 | 1.4 | 1.3 | 1.3 |
| *M. leachii (1 Mbp)* | 0 | 2.6 | 3.2 | 3.6 | 3.8 | 4.0 | 4.0 | 3.8 | 3.9 | 3.7 | 3.7 | 3.9 |
| | 1 | 2.4 | 2.7 | 3.1 | 3.2 | 3.4 | 3.3 | 3.3 | 3.3 | 3.2 | 3.2 | 3.1 |
| | 2 | 2.2 | 2.3 | 2.6 | 2.7 | 2.6 | 2.7 | 2.8 | 2.6 | 2.5 | 2.4 | 2.5 |
| *P. brasiliensis (6 Mbp)* | 0 | 2.6 | 3.4 | 3.8 | 4.2 | 4.4 | 4.5 | 4.5 | 4.5 | 4.1 | 4.5 | 4.8 |
| | 1 | 2.4 | 2.9 | 3.3 | 3.6 | 3.9 | 4.0 | 4.0 | 3.9 | 4.0 | 4.0 | 3.8 |
| | 2 | 2.3 | 2.5 | 2.8 | 3.1 | 3.1 | 3.3 | 3.4 | 3.0 | 3.2 | 3.2 | 3.3 |
| *S. cellulosum (13 Mbp)* | 0 | 2.7 | 3.0 | 4.1 | 4.4 | 4.5 | 5.0 | 4.9 | 4.7 | 4.6 | 5.1 | 5.4 |
| | 1 | 2.5 | 3.0 | 3.5 | 3.8 | 4.2 | 4.3 | 4.3 | 4.0 | 4.0 | 4.3 | 4.1 |
| | 2 | 2.3 | 2.5 | 2.9 | 3.3 | 3.4 | 3.6 | 3.8 | 3.2 | 3.5 | 3.4 | 3.7 |

# Chapter 4

# Compressed Gaussian Observations

Unlike DNA sequences, which can be represented as ordered sets of discrete values over a finite sized alphabet, sequence data from microarray experiments are real-valued observations which cannot be exploited for sequence repetition. These datasets usually contain noisy observations, and separation between different class of observations, specially for one-dimensional arrayCGH and Single Neucleotide Polymorphism Array (SNParray) data, are often low. Hence, discretization of continuous emissions, similar to vector quantization used in speech recognition [22], is not viable. Moreover, maximal compression is to be expected for small number of discrete symbols and, clearly, compression ratio conflicts with fidelity in the analysis. As an important contribution of this thesis, in this chapter, we propose a reduced representation of continuous observations from microarray data in the context of CNV detection problem.

CNVs are chromosomal aberrations of the genome which results in gaining or losing one or more copies of genomic segments of one kilobase or more. They have been shown to be prevalent in the human genome [44, 45] and their roles in diseases and evolution have been extensively studied [46–49]. Although microarray experiments can directly compute absolute copy numbers, the ratio between a patient's and control's copy number, which are computed by arrayCGH and SNParray, is often more important. Given the normalized log-ratio of copy numbers, one needs to determine the segments of similar copy number; a loss or gain in a genomic location indicates an area of biological interest; see Figure 4.1.

Segmentation problems, specially identifying segments from a biological sequence, have been extensively studied in various settings including CNV detection from microarray data. In this chapter, our focus is on detecting CNVs using HMMs, which is a

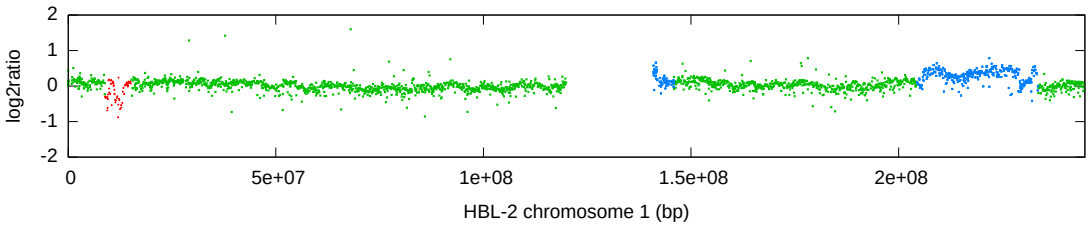



Figure 4.1: Log ratio of sample vs. control copy number data. Comparative loss is shown in red while gain is shown in green.

naturally suitable model for segmentation, and has been shown to be successful in CNV detection [50–54]. Similar to discrete sequence segmentation in the previous chapter, in practice, the parameters of a HMM are often estimated with ML and a segmentation for CNV is obtained with the Viterbi algorithm. This introduces considerable uncertainty in the segmentation, which can be avoided with Bayesian approaches integrating out parameters using MCMC sampling. While the advantages of Bayesian approaches have been clearly demonstrated, the likelihood based approaches are still preferred in practice for their lower running times; datasets coming from high-density arrays and high-throughput sequencing amplify these problems. We propose an approximate sampling technique, inspired by compression of discrete sequences in the previous chapter and by clusters leveraging spatial relations between data points in typical data sets, to speedup the MCMC sampling [24].

This chapter is organized as follows. First, we discuss related work in Section 4.1. Then in Section 4.3 we explain the approach behind compressing continuous observations into blocks and using those blocks for approximating MCMC. In Section 4.5 we evaluate our method and show the computational benefits and qualitative results on arrayCGH and SNParray datasets.

## 4.1 Related Work

A wide range of methods for CNV detection in arrayCGH data have been developed in recent years, including change-point detection based methods [55, 56], smoothing

based methods [57,58], and hierarchical clustering [59]. Here, we concentrate on HMM-based approaches which have been proposed for segmenting sequences of continuous-valued observations and shown to match or improve upon the state-of-the-art [52–54]. Typically, these models are used to describe the log ratio of normal vs. control's copy number data (generated by arrayCGH experiments); HMM states represent segments of such a sequence—normal, loss or gain in copy number w.r.t. the control—and, usually, Gaussian distributions model observations in a particular state, see Figure 4.2 (right).

For a different task, arguments about spatial relations between groups of multi-variate data points were used to achieve considerable speed-up. Moore and colleagues used modified *kd*-trees, a data structure to efficiently execute spatial queries such as determining the nearest neighbor of a given point, to accelerate $k$-means [60]. In the reassignment step of $k$-means one has to find the nearest centroid for every data point. Due to the *kd*-tree, there are groups of points contained in a node of the tree for which this decision about the nearest centroid can be made *simultaneously* by a geometrical argument about the vertices of the hyperrectangle defined by this node. A similar *kd*-tree based approach was used in speech recognition [61,62] to quickly find the most important components in a mixture of large number of Gaussians and thus approximate the full observation density in one individual HMM state with multi-variate emissions.

At the core of our approach is a similar geometrical argument about several univariate data points based on hierarchical clustering. We adaptively identify blocks of observations, cf. Fig. 4.2 (left). For *all* observations in a block we now estimate, at least conceptually, the most likely state *simultaneously* depending on the means of the Gaussians in each state to gain a considerable speed-up proportional to the average block length. Similarly, we can avoid sampling states for each individual observation in a block if we can bound the posterior. Considerable care has to be taken for combining blocks and to bound the errors introduced by the approximations based on geometric arguments.

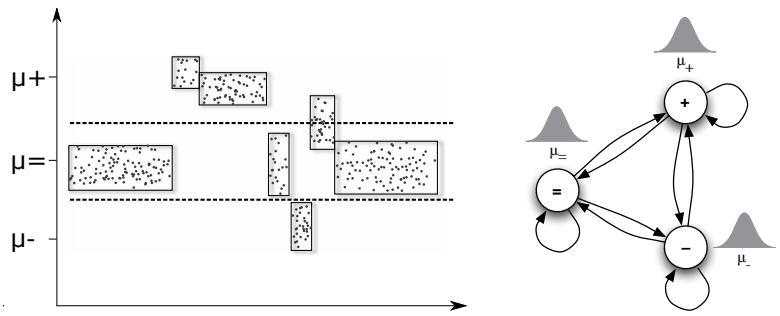



Figure 4.2: For a sequence (left) there is no overlap in $y$-direction and decisions about the most likely state can be made per block considering the means of the Gaussians of a three-state HMM (right), $\mu_-$, $\mu_=$ and $\mu_+$.

## 4.2 Bayesian HMM for CNV

We follow the definitions related to Bayesian HMM with Gaussian emissions from Section 2.1.2. Here, we restrict HMMs to at most four states—$s_1, s_2, s_3$ and $s_4$ correspond to loss, normal, gain and multiple gain states respectively. We associate the loss state $s_1$ with one or more copy loss, gain state $s_3$ with exactly one copy gain and state $s_4$ with multiple copy gain. In some cases we will combine state $s_3$ and $s_4$ into one state representing one or more gain (such as in Figure 4.2 (right)), which will be clear from the context. For prior distributions we use the conjugate priors mentioned in Section 2.1.2 in general. We also apply dataset specific domain knowledge in the form of truncated distributions following [34, 50] (explained in Section 4.5).

## 4.3 Approximate MCMC Sampling

At first, through application of a modified hierarchical clustering algorithm, we compress the observation sequence $O = o_1, \ldots, o_T$ into $T'$ number of blocks, $O' = o'_1, \ldots, o'_{T'}$, where $T' \ll T$, cf. Fig. 4.2 (left). For each block $o'_i$, we precompute $\sum_{o_j \in o'_i} o_j$ and $\sum_{o_j \in o'_i} o_j^2$, and store these statistics along with the size of the block $|o'_i|$ as part of an one time compression process. In subsequent MCMC iterations, we assume that observations compressed in a block $o'_i$ arise from the same underlying state. In other words, we ignore the contribution of the state paths that do not go through the same state for observations in $o'_i$. By ignoring those state paths, we refer to them as *weak state paths*,

when computing forward variables, and by reusing the pre-computed statistics, we are able to accelerate MCMC sampling.

### 4.3.1 Top-down Hierarchical Clustering

While a brute force compression algorithm only considers local information, a top-down hierarchical clustering approach alternately looks at both dimensions of the data—genome position and the corresponding log ratio of copy numbers—and utilizes global information such as the density of data points to create better quality blocks. We use a modified hierarchical clustering algorithm to find such blocks and discuss the details below.

Given a starting *width* parameter $w$, we create a list of nodes from the observation sequence $O = o_1, \ldots, o_T$ using the following steps.

1. Let $O' = \phi$ be the starting list, $\delta = 1.25$ (picked empirically), level $L = 1$, and dimension $d = 1$.

2. If $|\max_{o_i \in O}(o_i) - \min_{o_i \in O}(o_i)| < \frac{w}{\delta^L}$ or $|O| = 1$, create a node storing the first and second raw moments of the observations in $O$ append it to $O'$, and then go to the end step. Otherwise, go to the next step.

3. If $d = 1$, find $o_m$, the median value of the observations in $O$. Partition $O$ into maximal sets of consecutive observations $O_1, \ldots, O_i, \ldots, O_p$ such that $\forall_{o \in O_i}\ o \leq o_m$ or $\forall_{o \in O_i}\ o \geq o_m$. For each such partition $O_i$, update level to $L + 1$, set $d = 0$ and go to step 2 considering $O_i$ as the input set $O$.

4. If $d = 0$, divide the input set $O$ into two parts $O_L = o_1, \ldots, o_i$ and $O_R = o_{i+1}, \ldots, o_{|O|}$ such that $|o_i - o_{i+1}| \geq \max_{j<|O|} |o_j - o_{j+1}|$. Then for each set $O_L$ and $O_R$, go to step 2 keeping the level value $L$ unchanged, and setting $d = 1$.

5. End step.

In the above recursive algorithm, $w$ states the initial width, $\delta$ controls the rate of width shrinking in successive levels of the iterations, and $O'$ accumulates the compressed

blocks of observations. The current iteration level $L$, the current dimension $d$, and the current input set $O$ are local variables in the recursive algorithm. Notice that we start with an empty list $O'$ and at the end of the recursive procedure $O'$ contains an ordered list of compressed observations. To gain further compression of the sequence, we sequentially go through the blocks of $O'$ and combine consecutive blocks if the distance between their means is less than $w$. We also combine three consecutive blocks if the outer blocks satisfy this condition and the inner block has only one observation, which is most likely a noisy observation. In step 3 of the above algorithm, the input set is divided into two subsets and each subset contains half of the elements from the original set. Consequently, the height of the recursion tree is at most $2 \log T$ and the running time of the above algorithm is $O(T \log T)$. This overhead is negligible compared to the time that it takes to run $M$ iterations of MCMC sampling.

**Width Parameter Selection**

For increasing values of $w$ the average block size increases exponentially in the above clustering scheme. As a result, the compression ratio $\gamma = \frac{T'}{T}$ plotted as a function of $w$, has a *knee* which can inform the choice of $w$. Moreover, methods originally developed to find the optimal numbers of clusters in clustering can be used to find the knee of such a curve automatically. In particular, we use the L-method [63] which finds the knee as the intersection of two straight lines fitted to the compression curve (more discussion on this in Section 4.5.3).

### 4.3.2 Fast Approximate Sampling Algorithm

Given the compressed input sequence $O' = o'_1, o'_2, \ldots, o'_{T'}$ our goal is to adapt the FBG-sampling algorithm 2.1. With the pre-computed statistics for each block, computing forward variables and subsequent sampling in algorithm 2.2 is a straightforward modification of the uncompressed case. In particular, we make the following two changes to algorithm 2.2.

- **Approximate forward variable:** For each time position $\bar{t} = \sum_{i=1}^{t} |o'_i|$, instead of computing $\alpha_{\bar{t}}(i)$, we compute an approximation $\hat{\alpha}_{\bar{t}}(i)$ by ignoring the contributions from the non-self transition probabilities inside the block.

$$
\begin{aligned}
\hat{\alpha}_{\bar{t}}(i) &= P(q_{\bar{t}} = i, o'_1, \ldots, o'_t | \theta) \\
&= \sum_{j=1}^{N} P(q_{\bar{t}-|o'_t|} = j, o'_1, \ldots, o'_{t-1})\, a_{ji}\, a_{ii}^{|o'_t|-1} \prod_{o_k \in o'_t} P(o_k | \mu_i, \sigma_i) \\
&= \sum_{j=1}^{N} \hat{\alpha}_{\bar{t}-|o'_t|}(j)\, a_{ji}\, a_{ii}^{|o'_t|-1} \prod_{o_k \in o'_t} P(o_k | \mu_i, \sigma_i) \\
&= \left( a_{ii}^{|o'_t|-1} \prod_{o_k \in o'_t} P(o_k | \mu_i, \sigma_i) \right) \sum_{j=1}^{N} \hat{\alpha}_{\bar{t}-|o'_t|}(j)\, a_{ji} \\
&= \underbrace{f\left( i,\, |o'_t|,\, \sum_{o_k \in o'_t} o_k,\, \sum_{o_k \in o'_t} o_k^2 \right)}_{\substack{\text{constant time computation} \\ \text{using precomputed statistics}}} \sum_{j=1}^{N} \hat{\alpha}_{\bar{t}-|o'_t|}(j)\, a_{ji}.
\end{aligned}
$$

  Note that, since we assume that all observations in a block originates from the same state, it is sufficient to compute the approximate forward variables only at the end time points of the pre-computed blocks.

- **Backward sampling:** Starting with the last block, we can iteratively sample a state for each block using the approximate forward variables.

Clearly, each iteration of approximate sampling takes $O(T'N^2)$ resulting in $\frac{T}{T'}$ times speed up. Given the sampled state sequence, we count the total number of transitions $c_{i,j}$ from state $i$ to state $j$ and sample the HMM parameters using the following posterior

distributions (for prior distributions see Section 2.1.2).

$$\pi \sim \text{Dirichlet}(\theta_1^{\pi} + \sum_{i=1}^{N} c_{i,1}, \ldots, \theta_N^{\pi} + \sum_{i=1}^{N} c_{i,N})$$

$$A_{i,*} \sim \text{Dirichlet}(\theta_1^{A_i} + c_{i,1}, \ldots, \theta_N^{A_i} + c_{i,N})$$

$$\mu_i|\sigma_i^2 \sim \mathcal{N}\left(\left(\frac{\tilde{\mu_i}}{\tilde{\sigma_i}^2} + \frac{\sum_{q_{\bar{t}}=i}\sum_{o_k \in o'_t} o_k}{\sigma_i^2}\right) \Big/ \left(\frac{1}{\tilde{\sigma_i}^2} + \frac{\sum_{q_{\bar{t}}=i}|o'_t|}{\sigma_i^2}\right), \left(\frac{1}{\tilde{\sigma_i}^2} + \frac{\sum_{q_{\bar{t}}=i}|o'_t|}{\sigma_i^2}\right)^{-1}\right)$$

$$\sigma_i^{-2}|\mu_i \sim \text{Gamma}\left(a_i + \frac{\sum_{q_{\bar{t}}=i}|o'_t|}{2}, b_i + \frac{\sum_{q_{\bar{t}}=i}\sum_{o_k \in o'_t}(o_k - \mu_i)}{2}\right)\Bigg)$$

Here $\tilde{\mu_i}$, $\tilde{\sigma_i}$, $a_i$, $b_i$ and $\theta^{A_i}$ (for the state $i$), and $\theta^{\pi}$ are the hyperparameters of the prior distributions. Since $\mu_i$ and $\sigma_i$ depend on each other, we work with the $\mu_i$ from the previous iteration to compute $\sigma_i$. See [24], [21] and [34] for a detailed derivation of the posterior hyperparameters.

## 4.4 Approximation Error in Symmetric HMMs

At first, ignoring weak state paths seems to be a very crude approximation for computing forward variables. But in many applications, in particular for CNV data, this is certainly not true. We demonstrate with a symmetric Gaussian HMM that the weak state path assumption is a fairly realistic approximation and leads to faster sampling. We define a symmetric HMM, $\theta = (A, B, \pi)$, with $N$ states $s_1, \ldots, s_N$, where we set self-transition probability $a_{ii} = t$ and non-self-transition probability $a_{ij} = \frac{1-t}{N-1}$ for $1 \leq i \neq j \leq N$, and $B = \{(\mu_1, \sigma^2), \ldots, (\mu_N, \sigma^2)\}$. Given a sequence of observations $O$ (assumed to be generated by $\theta$) and its compressed form $O'$ we describe an important lemma and some remarks below.

**Lemma 1.** Let $O_{1,i-1} = O_{i-1}$, $O_{i,i+n-1} = o'$, $\min_{o_l \in o'} o_l = o'_{min}$, $\max_{o_l \in o'} o_l = o'_{max}$, $d = \min_{j \neq k} |\mu_j - \mu_k|$ and $\frac{P(q_i|O_{i-1})}{P(q_i=s_x|O_{i-1})} \leq \alpha$. Assuming there exists a state $s_x$ s.t. $\tau = \min\left(o'_{min} - \frac{\mu_{s_{x-1}} + \mu_{s_x}}{2}, \frac{\mu_{s_x} + \mu_{s_{x+1}}}{2} - o'_{max}\right) \geq 0$, we can show that $\frac{\sum_{Q_{i,i+n-1} \in S^n} P(Q_{i,i+n-1}, o'|O_{i-1})}{\sum_{s \in S} P(Q_{i,i+n-1}=s, o'|O_{i-1})} \leq$

$\alpha((1+rc)^{n-1} + (N-1)c^{\frac{2n}{N}}(1+r)^{n-1})$, where $r = \frac{1-t}{t}$ and $c = e^{-\frac{d\tau}{2\sigma^2}}$.[1]

*Proof.* Using the assumption on $\tau$, for any position $i \leq l \leq i+n-1$, we can argue that,

$$\frac{e^{-\frac{1}{2}\left(\frac{o_l-\mu_{q_l}}{\sigma}\right)^2}}{e^{-\frac{1}{2}\left(\frac{o_l-\mu_{s_x}}{\sigma}\right)^2}} \leq e^{-\frac{|\mu_{q_l}-\mu_{s_x}|\tau}{\sigma^2}} \leq \begin{cases} 1 & \text{if } q_l = s_x, \\ e^{-\frac{d\tau}{\sigma^2}} & \text{otherwise.} \end{cases} \tag{4.1}$$

For any partial state path $Q_{i,i+n-1}$,

$$\begin{aligned} P(Q_{i,i+n-1}, o'|O_{i-1}) &= P(q_i|O_{i-1})P(o_i|q_i, O_{i-1}) \prod_{k=i}^{i+n-2} a_{q_k q_{k+1}} P(o_{k+1}|q_{k+1}) \\ &= P(q_i|O_{i-1}) \frac{e^{-\frac{1}{2}\left(\frac{o_i-\mu_{q_i}}{\sigma}\right)^2}}{\sqrt{2\pi\sigma^2}} \prod_{k=i}^{i+n-2} a_{q_k q_{k+1}} \frac{e^{-\frac{1}{2}\left(\frac{o_{k+1}-\mu_{q_{k+1}}}{\sigma}\right)^2}}{\sqrt{2\pi\sigma^2}} \ . \end{aligned} \tag{4.2}$$

We partition $S^n$, the set of all possible partial state paths of length $n$, into $N$ subsets $S^{s_1} \ldots S^{s_N}$ such that, $S^{s_j} = \{\tilde{S} \in S^n : \big(\forall_{s_l \neq s_j} C(\tilde{S}, s_j) > C(\tilde{S}, s_l)\big) \vee \big(\big(\forall_{s_l \neq s_j} C(\tilde{S}, s_j) \geq C(\tilde{S}, s_l)\big) \wedge \tilde{S}_1 = s_j\big)\}$ for $1 \leq j \leq N$, where $C(\tilde{S}, s) = \sum_{q_k \in \tilde{S}} 1(q_k = s)$. We again partition $S^{s_j} = \cup_{k=0}^{n-1} S_k^{s_j}$ such that, $S_k^{s_j} = \{\tilde{S} \in S^{s_j} : \left(\sum_{l=1}^{n-1} 1(\tilde{S}_l \neq \tilde{S}_{l+1})\right) = k\}$. The size of $S^n$ can be expressed in terms of total number of non-self-transitions present in a path $|S^n| = N^n = N \sum_{k=0}^{n-1} \binom{n-1}{k}(N-1)^k$.

As the sets $S^{s_j}$ are equal sized partitions of $S^n$, $|S^{s_j}| = \sum_{k=0}^{n-1} \binom{n-1}{k}(N-1)^k$. Also notice that, by definition, the partial state paths in $S^n$ with exactly $k$ number of non-self-transitions are equally distributed among the subsets $S^{s_j}$. As a result, $|S_k^{s_j}| = \binom{n-1}{k}(N-1)^k$.

Now we define $S^{[s]} = \{Q_{i,i+n-1} : Q_{i,i+n-1} = s\}$. For the remaining part of the proof, if $Y$ is a set of partial state paths, we use $P(Y, o'|O_{i-1})$ in place of $\sum_{Q_{i,i+n-1} \in Y} P(Q_{i,i+n-1}, o'|O_{i-1})$

[1]For simplicity of the notation, we follow the convention that $\mu_{x_0} = -\infty$ and $\mu_{x_{N+1}} = \infty$ so that the proof holds for $x = 1$ or $x = N$.

for clarity and rewrite $\frac{\sum_{Q_{i,i+n-1}\in S^n} P(q_i,\cdots,q_{i+n-1},o'|O_{i-1})}{\sum_{s\in S} P(q_i=\cdots=q_{i+n-1}=s,o'|O_{i-1})}$ as $\frac{P(S^n,o'|O_{i-1})}{\sum_{s\in S} P(S^{[s]},o'|O_{i-1})}$.

$$\frac{P(S^n,o'|O_{i-1})}{\sum_{s\in S} P(S^{[s]},o'|O_{i-1})} < \frac{P(S^n,o'|O_{i-1})}{P(S^{[s_x]},o'|O_{i-1})} = \bigcup_{j=1}^{N} \frac{P(S^{s_j},o'|O_{i-1})}{P(S^{[s_x]},o'|O_{i-1})}. \tag{4.3}$$

Now we derive an upper bound of the contribution from state paths in $S^{s_x}$. In the following equations we make use of the fact that a state path with $k$ non-self-transitions goes through at least $\frac{k}{2}$ non-$s_x$ states.

$$\begin{aligned}
\frac{P(S^{s_x},o'|O_{i-1})}{P(S^{[s_x]},o'|O_{i-1})} &= \frac{\sum_{k=0}^{n-1}\sum_{\tilde{S}\in S_k^{s_x}} P(\tilde{S},o'|O_{i-1})}{P(S^{[s_x]},o'|O_{i-1})} \\
&= \sum_{k=0}^{n-1}\sum_{\tilde{S}\in S_k^{s_x}} \frac{P(\tilde{S},o'|O_{i-1})}{P(S^{[s_x]},o'|O_{i-1})} \\
&= \sum_{k=0}^{n-1}\sum_{\substack{\tilde{S}\in S_k^{s_x}\\ \tilde{S}=Q_{i,i+n-1}}} \frac{P(q_i|O_{i-1})e^{-\left(\frac{o_i-\mu_{q_i}}{\sqrt{2}\sigma}\right)^2}}{P(s_x|O_{i-1})e^{-\left(\frac{o_i-\mu_{s_x}}{\sqrt{2}\sigma}\right)^2}} \prod_{j=i}^{i+n-2} \frac{a_{q_jq_{j+1}}e^{-\left(\frac{o_{j+1}-\mu_{q_{j+1}}}{\sqrt{2}\sigma}\right)^2}}{a_{s_xs_x}e^{-\left(\frac{o_{j+1}-\mu_{s_x}}{\sqrt{2}\sigma}\right)^2}} \\
&= \sum_{k=0}^{n-1}\sum_{\substack{\tilde{S}\in S_k^{s_x}\\ \tilde{S}=Q_{i,i+n-1}}} \frac{P(q_i|O_{i-1})}{P(s_x|O_{i-1})} \prod_{j=i}^{i+n-2} \frac{a_{q_jq_{j+1}}}{a_{s_xs_x}} \prod_{j=i}^{i+n-1} \frac{e^{-\left(\frac{o_j-\mu_{q_j}}{\sqrt{2}\sigma}\right)^2}}{e^{-\left(\frac{o_j-\mu_{s_x}}{\sqrt{2}\sigma}\right)^2}} \\
&\le \sum_{k=0}^{n-1}\sum_{\substack{\tilde{S}\in S_k^{s_x}\\ \tilde{S}=Q_{i,i+n-1}}} \alpha\left(\frac{1-t}{(N-1)t}\right)^k \prod_{j=i}^{i+n-1} \frac{e^{-\left(\frac{o_j-\mu_{q_j}}{\sqrt{2}\sigma}\right)^2}}{e^{-\left(\frac{o_j-\mu_{s_x}}{\sqrt{2}\sigma}\right)^2}} \\
&\le \sum_{k=0}^{n-1}\binom{n-1}{k}(N-1)^k\alpha\left(\frac{1-t}{(N-1)t}\right)^k\left(e^{-\frac{d\tau}{\sigma^2}}\right)^{\frac{k}{2}} \\
&= \sum_{k=0}^{n-1}\binom{n-1}{k}\alpha\left(\frac{1-t}{t}\right)^k\left(e^{-\frac{d\tau}{\sigma^2}}\right)^{\frac{k}{2}} \\
&= \sum_{k=0}^{n-1}\alpha\binom{n-1}{k}\left(\frac{1-t}{t}\right)^k\left(e^{-\frac{d\tau}{\sigma^2}}\right)^{\frac{k}{2}} \\
&= \sum_{k=0}^{n-1}\alpha\binom{n-1}{k}(rc)^k \\
&= \alpha(1+rc)^{n-1}\ .
\end{aligned} \tag{4.4}$$

Similarly, we derive an upper bound of the contribution from state paths in $S^{s_y}$, where $1 \leq y \neq x \leq N$. Now we use the fact that, because of the pigeonhole principle any state path in $S^{s_y}$ has to go through at least $\frac{n}{N}$ non-$s_x$ states.

$$\begin{aligned}
\frac{P(S^{s_y}, o'|O_{i-1})}{P(S^{[s_x]}, o'|O_{i-1})} &\leq \sum_{k=0}^{n-1} \sum_{\tilde{S}\in S_k^{s_y}} \sum_{\tilde{S}=Q_{i,i+n-1}} \alpha\left(\frac{1-t}{(N-1)t}\right)^k \prod_{j=i}^{i+n-1} \frac{e^{-\frac{1}{2}\left(\frac{o_j-\mu_{q_j}}{\sigma}\right)^2}}{e^{-\frac{1}{2}\left(\frac{o_j-\mu_{s_x}}{\sigma}\right)^2}} \\
&\leq \sum_{k=0}^{n-1} \binom{n-1}{k} (N-1)^k \alpha\left(\frac{1-t}{(N-1)t}\right)^k \left(e^{-\frac{d\tau}{\sigma^2}}\right)^{\frac{n}{N}} \\
&= \sum_{k=0}^{n-1} \binom{n-1}{k} (N-1)^k \alpha\left(\frac{1-t}{(N-1)t}\right)^k \left(e^{-\frac{d\tau}{\sigma^2}}\right)^{\frac{n}{N}} \\
&= \sum_{k=0}^{n-1} \binom{n-1}{k} \alpha\left(\frac{1-t}{t}\right)^k \left(e^{-\frac{d\tau}{\sigma^2}}\right)^{\frac{n}{N}} \\
&= \sum_{k=0}^{n-1} \alpha\binom{n-1}{k} \left(\frac{1-t}{t}\right)^k \left(e^{-\frac{d\tau}{\sigma^2}}\right)^{\frac{n}{N}} \\
&= \sum_{k=0}^{n-1} \alpha c^{\frac{2n}{N}} \binom{n-1}{k} r^k \\
&= \alpha c^{\frac{2n}{N}} (1+r)^{n-1} \ . \qquad (4.5)
\end{aligned}$$

Applying (4.4) and (4.5) in (4.3) we get,

$$\frac{\sum_{Q_{i,i+n-1}\in S^n} P(Q_{i,i+n-1}, o'|O_{i-1})}{\sum_{s\in S} P(Q_{i,i+n-1}=s, o'|O_{i-1})} \leq \alpha((1+rc)^{n-1} + (N-1)c^{\frac{2n}{N}}(1+r)^{n-1}) \ .$$

□

**Remark 1**

For realistic values of $\tau, t$, and $n$, the contribution from ignored weak state paths, which we call $\epsilon$, can be very small. If $\epsilon \ll 1$, ignoring weak state paths will not introduce large errors in the computation. For the 2-state example in Section 4.5.1, where $t = 0.9$, $d = 1$, and $\sigma^2 = 0.1$, $\epsilon$ is at most $\frac{1}{3}$ for block length $n \leq 10$ if we assume $\tau > 0.25$ and $\alpha = 1$. If $\tau$ is much larger and consequently $c^{\frac{2n}{N}}$ is much smaller, we can roughly say that $n$ can be as large as $1 + \log_{1+rc}(1+\epsilon)$ in a symmetric Gaussian HMM.

**Remark 2**

We often encounter situations where $P(q_i = s_x|O_{i-1}) \gg P(q_i \neq s_x|O_{i-1})$. Even though it is not exploited in the lemma ($\alpha$ being greater than or equal to 1), as a consequence of this, the observation sequence can be compressed into larger blocks keeping $\epsilon$ small in practice.

## 4.5 Experiments

We evaluate FBG-sampling and approximate sampling in three different settings. First, its effectiveness is verified for a simple two state model. Then, we test on simulated ArrayCGH data which is the accepted standard for method evaluation [64]. Finally, we report findings from an analysis of Mantle Cell Lymphoma (MCL) cell lines [65], Corriel cell lines [66], GBM datasets [67], and high resolution SNP arrays [54,68]. For biological data, if multiple chromosomes are present, we use pooling [50] across chromosomes, which does not allow transition between different chromosomes but assumes model parameters to be identical across chromosomes. Throughout this section we define $\sigma_D$ to be the standard deviation of all observations in the dataset. We compress the dataset with increasing values of $w = 0.25\sigma_D, 0.5\sigma_D, 0.75\sigma_D, \ldots$. For evaluation we consider the experiments as two class problems: aberrant clones belong to the positive class and normal clones belong to the negative class. When ground truth labels of a dataset are available we report F1-measure, recall, and precision for the experiment. With $tp$, $fp$, $tn$, $fn$ we denote the number of true and false positives and true and false negatives respectively. Recall is defined as $\frac{tp}{tp+fn}$, precision as $\frac{tp}{tp+fp}$, and F1-measure as $\frac{2\times recall\times precision}{recall+precision}$. Experiments were run with a Python implementation on a Linux machine with 1.6 GHz Intel Core 2 Duo processor and 2 GB memory. For Expectation Maximization (EM), we use the Baum-Welch algorithm from the GHMM package which is implemented in C and considerably faster than a Python implementation.

### 4.5.1 Synthetic Data

#### 2-State HMM

We define a HMM $\theta_{2ST} = (A, B, \pi)$ with $A = [[0.9, 0.1], [0.1, 0.9]], B = [(0, 0.1), (1, 0.1)], \pi = [\frac{1}{2}, \frac{1}{2}]$. From $\theta_{2ST}$ we sample an observation sequence $O = o_1, \ldots, o_{10,000}$, and run MCMC for $M = 100$ steps with hyperparameter values $\tilde{\mu}_{1:2} = 0, 1$ for the prior mean on $\mu$, $\tilde{\sigma}_{1:2} = 0.5, 0.5$ for the prior variance on $\mu$, $a_{1:2} = 4, 4$ for the shape of Gamma prior on $\sigma^{-2}$, $b_{1:2} = 1, 1$ for the rate of Gamma prior on $\sigma^{-2}$, $\delta^{\pi} = 1, 1$ for the Dirichlet prior on the initial distribution $\pi$, and $\delta^{A_i}_{1:2} = 1, 1$ for the Dirichlet prior on row $i$ of transition matrix $A$.

After $M$ iterations, we compare the posterior probabilities $P(q_t = i|O, \theta^M_{FBG})$ and $P(q_t = i|O, \theta^M_A)$, where $\theta^M_{FBG}$ and $\theta^M_A$ are $M$-th parameter samples of FBG-sampling and approximate sampling. Fig. 4.3 shows that the posterior probability of being in state 1 for each position can be approximated fairly well even for large values of $w$. The average posterior error $\tilde{P} = \frac{1}{2T} \sum_t \sum_i |P(q_t = i|\theta^M, O) - P(q_t = i|\theta^{true}, O)|$ reflects the same fact in Table 4.1. Similarly, we compute the Viterbi paths and report total number of mismatches between them along with the likelihoods in Table 4.1.

#### Simulation from Genetic Template

We use 500 simulated datasets published in [64]. Each dataset has 20 chromosomes and 100 clones per chromosome for a total of 2,000 clones per dataset. A four-state HMM predicts the aberrant regions—loss defined as state $S_1$ and gain defined as state $S_3$ or $S_4$. The neutral region is modeled as state $S_2$. We put an ordering constraint on the means, $\mu_1 < \mu_2 < \mu_3 < \mu_4$, to prevent label switching of the states [26]. Hyperparameter choices follow [50] and are $\tilde{\mu}_{1:4} = -0.5, 0, 0.58, 1$ for the prior mean on $\mu$, $\tilde{\sigma}_{1:4} = 0.5, 0.001, 1.0, 1.0$ for the prior variance on $\mu$, $a_{1:4} = 10, 100, 5, 5$ for the shape of gamma prior on $\sigma^{-2}$, and $b_{1:4} = \delta^{\pi} = \delta^{A_i}_{1:4} = 1, 1, 1, 1$ for the rate of gamma prior on $\sigma^{-2}$, the Dirichlet prior on initial distribution $\pi$, and the Dirichlet prior on row $i$ of transition matrix $A$, respectively.

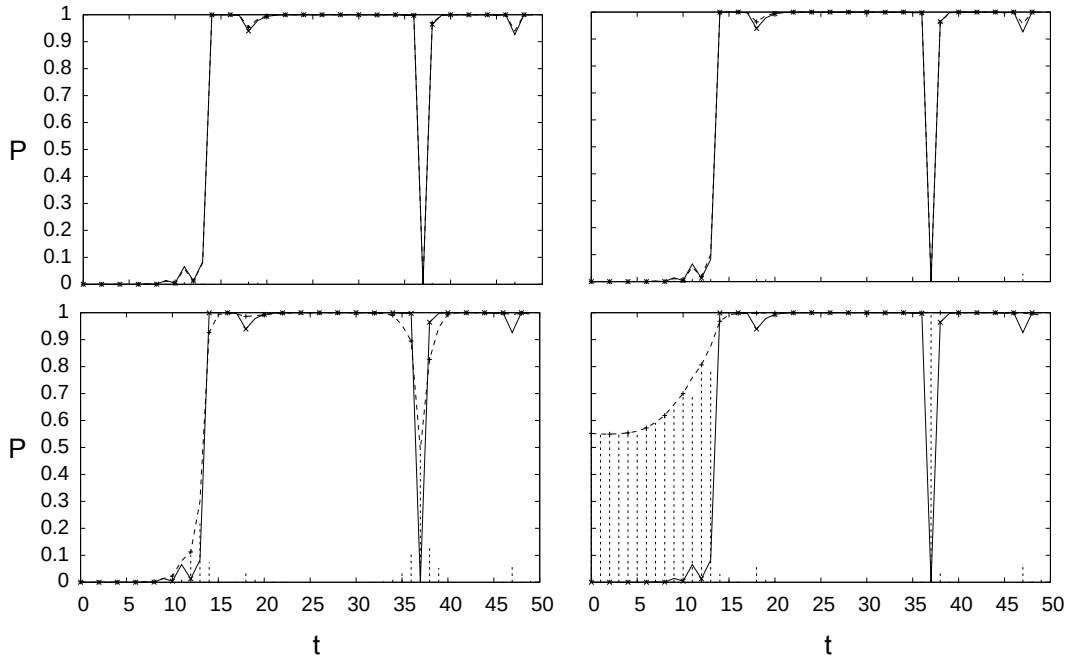



Figure 4.3: Simulated data: approximate posterior We show the posterior probability of state 1 (y-axis) for first fifty observations (x-axis) with $w = 0.5\sigma_D$ (top left), $1.0\sigma_D$ (top right), $1.5\sigma_D$ (bottom left), and $2.0\sigma_D$(bottom right). The true posterior is shown as a solid line, the approximate posterior as a dashed line, and their absolute difference is shown in dashed vertical lines.

Table 4.2 shows the mean and standard deviation of F1-measure, recall, and precision over the 500 datasets for FBG-sampling, approximate sampling, and Expectation Maximization (EM) with the ground truth provided by [64]. Even for this collection of relatively small datasets we see a 10-fold speed up. For each dataset we run FBG and approximate sampling for $M = 100$ steps (we have visually monitored the parameters and noticed convergence within 50 steps, see Fig. 4.4 for a representative example). The last 10 samples are used to compute 10 samples of the posteriors for each state and for each position in the observation sequence. Subsequently, aberrant regions are predicted based on the average of those distributions. We report the speed-up of approximate vs. FBG sampling based on the time it takes to compress the sequence and run $M$ steps of MCMC. For one individual dataset EM requires 58 seconds on average, which allows for a total of 800-1000 repetitions from randomized points sampled from the prior distributions in the time needed for FBG sampling. Each run continues until the likelihood converges and the best model based on likelihood is selected. Aberrant

regions are predicted and compared against the ground truth based on the Viterbi path. We report the mean and standard deviation of F1-measure, recall, and precision over the results of EM on 500 datasets.

### 4.5.2 Biological Data

#### Mantle Cell Lymphoma (MCL)

De Leeuw and colleagues identified recurrent variations across cell lines using ArrayCGH data of MCL cell lines [65]. Out of the eight cell lines [65] HBL-2 was fully annotated with marked gain and loss regions in the autosomes. This dataset contains about 30,000 data points (combining all the autosomes). We have used a four-state HMM for predicting aberrant regions. State 1 represents copy number loss, state 2 represents normal copy number, state 3 represents single copy gain, and state 4 multiple gain. For HBL-2 we report the F1-measure, recall, precision and speed-up. Similar to the synthetic case we put an ordering constraint on the means, $\mu_1 < \mu_2 < \mu_3 < \mu_4$. Hyperparameter choices follow [50] and are same as for the simulation from genetic template, except for $\tilde{\sigma}_{1:4} = 0.2, 0.1, 0.2, 0.2$, the prior variance on $\mu$, and $a_{1:4} = 15, 20, 10, 10$, the shape of gamma prior on $\sigma^{-2}$. Settings for FBG-sampling and approximate sampling are identical to the simulated case with one exception; for each simulated dataset sampling methods run once and we report the average and standard deviation over 500 datasets, but for HBL-2 we let them run 10 times and report the average and standard deviation of these 10 F1-measures, recalls, and precisions in Table 4.2. Each EM run starts with the initial parameter values sampled either from the prior distributions, or from uniform distributions, and continues until the likelihood value converges. We report the performance of the most likely model (which is the preferred criteria to select a model), the likelihood of the best model based on F1-measure, and the average and standard deviation of F1-measures, recalls, and precisions of all the models generated by EM. As representative examples, we also plot the segmentation of chromosome 1 and 9 computed by FBG-sampling and approximate sampling along with the ground truth labels in Fig. 4.5.

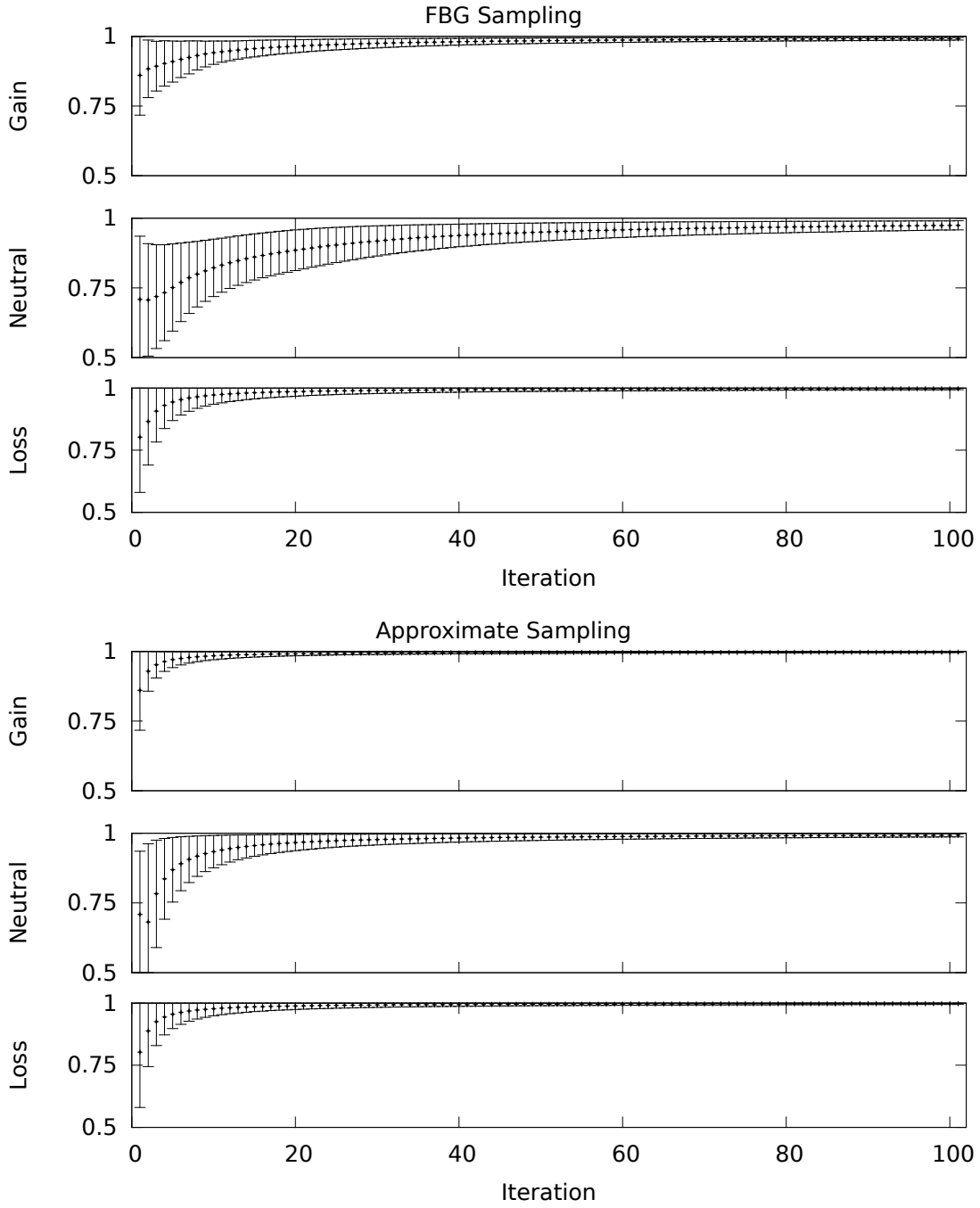



Figure 4.4: MCMC convergence The convergence of posterior probabilities for loss, neutral, and gain of three representative probes—probe 1658, probe 1512, and probe 447 respectively—from the simulated dataset 63 are shown. For each probe, at first, the posterior probability of the corresponding HMM state, given the sampled parameters from the current MCMC iteration, is computed. The time-average of these posterior probabilities, starting from the first iteration to the current iteration, approximates the posterior of the HMM state given the data. The mean of the posterior probabilities over 10 MCMC chains are shown with error bars (mean $\pm$ one standard deviation)—loss probe in the bottom row, neutral probe in the middle, and the gain probe in the top row. The top figures show the outcomes of FBG sampling and the bottom figures show the outcomes of approximate sampling. Note that the reduction in standard deviation suggests that approximate sampling converges quicker than FBG sampling for these probes.

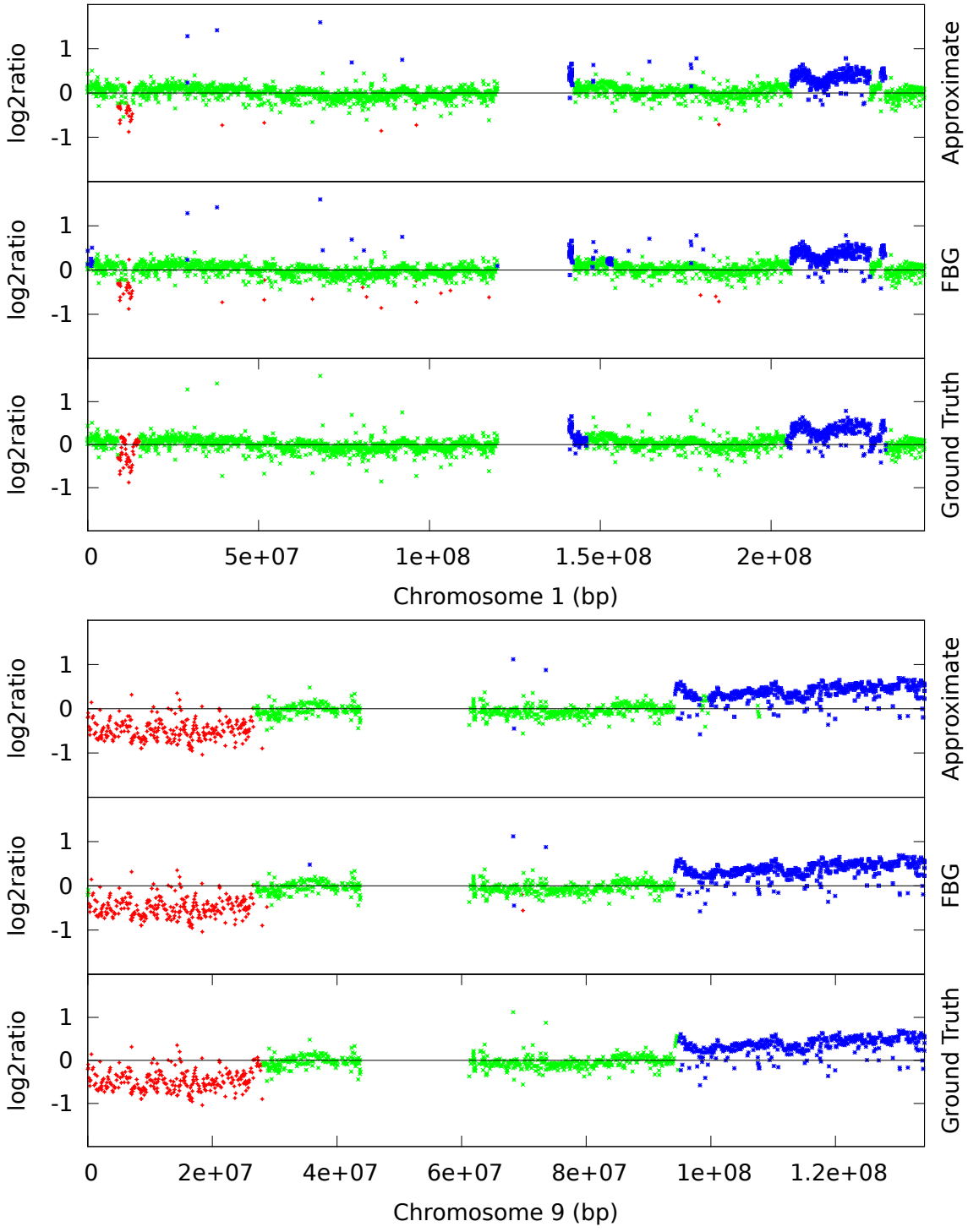



Figure 4.5: HBL-2: chromosome 1 and 9 We contrast the ground truth and the segmentations produced by FBG-sampling and approximate sampling. For approximate sampling $w$ was set to the value recommended by the L-method. Here, clones predicted as loss are shown in red, normal clones in green, and gain in blue. The figure at the top shows chromosome 1 and the bottom figure shows chromosome 9.

### Corriel

Corriel cell lines were used by Snijders *et al.* [66] and are widely considered a gold standard in ArrayCGH data analysis. This dataset is smaller and, in fact, fairly easy compared to the MCL cell lines. For the Corriel cell line we use a 4-state HMM and report the results for GM05296 and GM00143 in Table 4.2. Again, approximate sampling performs competitively while achieving more than a 10-fold speed-up. Hyperparameter choices follow [34].

### GBM

The glioma data from Bredel *et al.* [67] has previously been used to analyze the performance of CNV detection methods [58, 69]. According to [69], GBM datasets are noisy but contains a mixture of aberrant regions with different width and amplitude. In particular, chromosome 13 of GBM31 is reported to have low amplitude loss in p-arm and chromosome 7 of GBM29 is reported to have high amplitude gains near the EGFR locus by previous studies [58, 69]. The segmentation of these two chromosomes are shown in Fig. 4.6. Although [69] reports that EM based HMM failed to detect these aberrations we see that Bayesian HMM has successfully detected both the gain in chromosome 7 and the loss in chromosome 13. For this dataset, we use a 3-state HMM with non-informative hyperparameters, $\tilde{\mu}_{1:3} = -\frac{\sigma_D}{2}, 0, \frac{\sigma_D}{2}$ for the prior mean on $\mu$, $\tilde{\sigma}_{1:3} = 0.2, 0.1, 0.2$ for the prior variance on $\mu$, $a_{1:3} = \frac{1}{\sigma_D^2}, \frac{1}{\sigma_D^2}, \frac{1}{\sigma_D^2}$ for the shape of gamma prior on $\sigma^{-2}$, $\delta^{\pi} = 1, 9, 1$ for the Dirichlet prior on initial distribution $\pi$, and $b_{1:3} = \delta_{1:3}^{A_i} = 1, 1, 1$ for the rate of gamma prior on $\sigma^{-2}$ and the Dirichlet prior on row $i$ of transition matrix $A$, respectively, and at the recommended $w$ value we see a 10 fold speed-up.

### SNP Array

High-resolution Single Nucleotide Polymorphism (SNP) arrays are capable of detecting smaller CNVs than ArrayCGH. To demonstrate the computational advantage of approximate sampling on SNP arrays we have chosen publicly available Affymetrix 100k

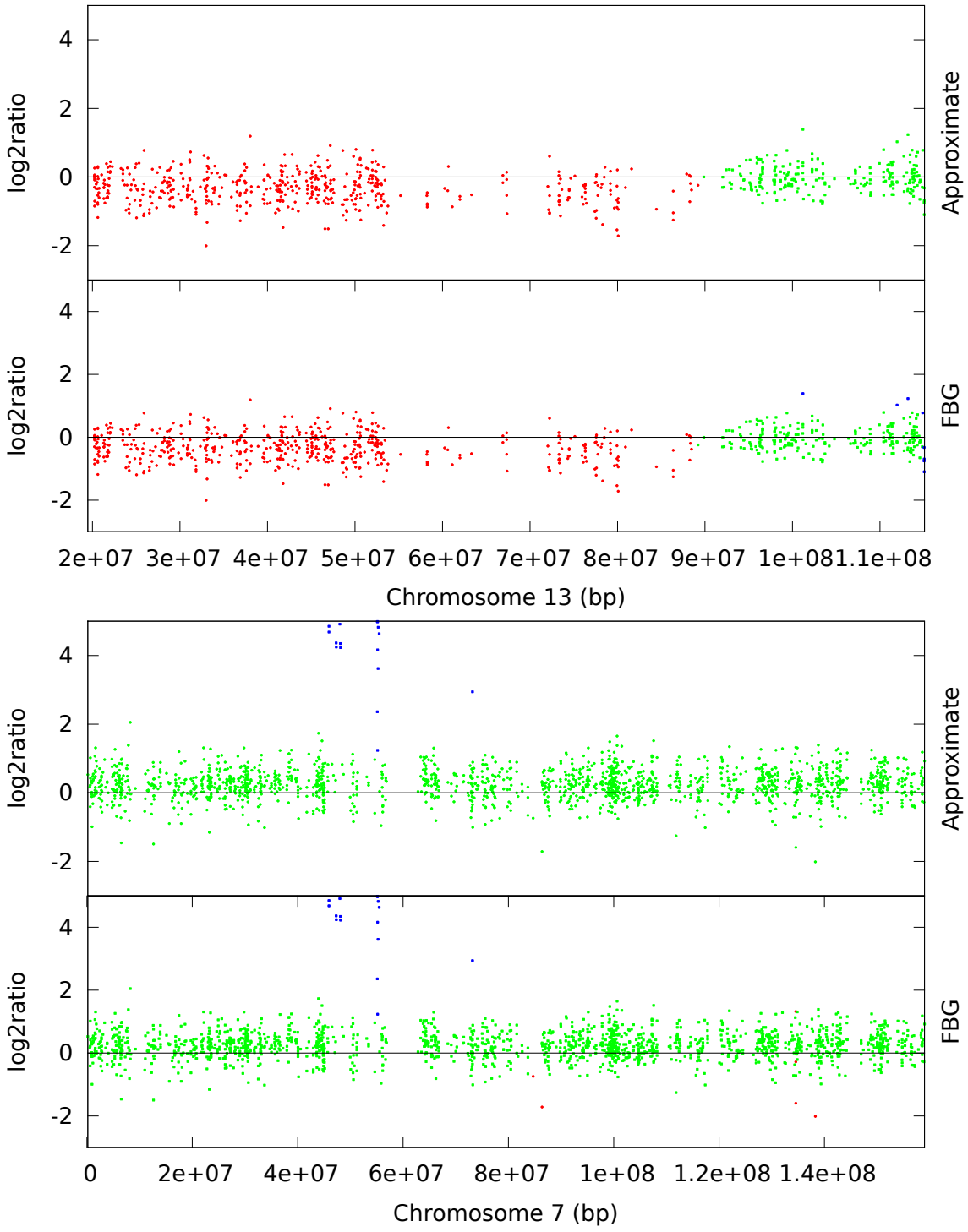



Figure 4.6: GBM: chromosome 7 (GBM29) and chromosome 13 (GBM31) Loss (red), normal (green), and gain (blue) clones are identified using FBG-sampling and approximate sampling. For approximate sampling $w = 1.5\sigma_D$ is used, which was recommended by the L-method.

pancreatic cancer datasets from [68] and Illumina HumanHap550 arrays of HapMap individuals which are provided as examples in PennCNV [54]. An Affymetrix 100k dataset consists of two different arrays each with $\approx 60,000$ SNP markers and, in total, $10^5$ data points per sample. On the other hand, the Illumina HumanHap550 array has around half a million SNP markers.

We have applied FBG-sampling and approximate sampling with $w = 1.8\sigma_D$, the recommended value by the L-method, to the sample datasets from Harada *et al.* [68] and found that the computational speed-up is 22-fold (100 runs of FBG-sampling takes 1844 seconds). Both sampling approaches mostly agree on their predictions but they, specially FBG-sampling, detect several more CNVs than previously identified [68]. For example, the high amplification in chromosome 11 (sample 33) is successfully identified by all methods but in chromosome 18 (sample 16) the sampling algorithms find a few normal regions previously predicted [68] as loss using the CNAG tool [70] (see Fig. 4.7). One possible reason for these differences is that while we use 269 HapMap samples as reference they use 12 unpublished normal samples as reference.

Similarly, we have tested our method with $2.0\sigma_D \leq w \leq 3.0\sigma_D$ against Illumina HumanHap samples and observed 70 to 90 fold speed-up in computations (100 runs of FBG-sampling takes 9693 seconds). These samples are taken from apparently healthy individuals and contain very few CNVs. As expected, both sampling algorithms' predictions are nearly identical and they seem to predict extreme outliers as aberrant markers. In contrast, PennCNV [54] does not report CNVs which are covered by less than 3 SNPs, thus suppressing the outliers as normal. We plot a typical region (from $1.4e+08$bp to $1.7e+08$bp) of chromosome 6 from sample 3 (ID 99HI0700A) in Fig. 4.8.

To set hyperparameters we follow the default parameters of the HMM used in PennCNV [54]. We have observed that HMMs for large arrays are particularly sensitive to the self-transition probabilities (which is also reflected in the default parameter values of the HMM used in PennCNV). Hence, hyperparameters were set to reflect the choice of high self-transition probability for each state —we set $\delta^{A_i}_{1:3} = \alpha_{i1}l, \alpha_{i2}l, \alpha_{i3}l$, the Dirichlet prior on row $i$ of transition matrix $A$, where $l = 5000$, $\alpha_{ii}$ is 0.99 for $i = 2$ , 0.95 for $i \neq 2$, $\alpha_{ij} = \frac{1-\alpha_{ii}}{2}$ for $i \neq j$. Other hyperparameters for the 3-state HMM

were set such that the expected values of prior distributions match the default values for PennCNV. In particular, they were $\tilde{\mu}_{1:3} = -0.66, 0, 0.54$ for the prior mean on $\mu$, $\tilde{\sigma}_{1:3} = 0.001, 0.001, 0.001$ for the prior variance on $\mu$, $a_{1:3} = 12, 30, 25$ for the shape of gamma prior on $\sigma^{-2}$, $b_{1:3} = 1, 1, 1$ for the rate of gamma prior on $\sigma^{-2}$, and $\delta^{\pi} = 1, 9, 1$ for the Dirichlet prior on initial distribution $\pi$, respectively.

### 4.5.3 Discussion

#### EM vs. MCMC

As already a 4-state Gaussian HMM has 23 free parameters applying EM is often difficult due to the existence of multiple local optima and the local convergence of EM. Consequently, the estimate has to be repeated many times with randomly initialized parameter values to find the most likely model. It should also be noted that not necessarily the model maximizing the likelihood exhibits the best performance in classifying aberrations 4.2. Additionally, applying any constraint in an EM settings, i.e. ordering of the state means, is harder than in MCMC. EM also produces inferior results on datasets exhibiting class imbalance, for example where one type of aberrations (observations for one HMM state) are rare or missing, while MCMC can overcome this problem using informative priors. In Table 4.2 we see that MCMC sampling performs better than EM on real biological data which is consistent with prior reports from Guha [34] and Shah [50] who also report difficulties with EM and better MCMC performances. In particular, for HBL-2 we observe that the best model in terms of F1-measure—which is not available in *de novo* analysis—is not the most likely model and the most likely model has very low precision and, consequently, worse F1-measure than MCMC sampling. On the simulated datasets, EM performs poorly if the dataset is imbalanced. As a result we observe slightly worse standard deviation for the precisions and F1-measures computed by EM in Table 4.2. We also notice from the confusion matrix of three classes—loss, neutral, and gain—that even though the mean F1-measure, recall, and precision (defined on two classes, neutral and aberrant) are high, due to label switching [26], EM does not distinguish gain from loss, and vice versa, very well (see

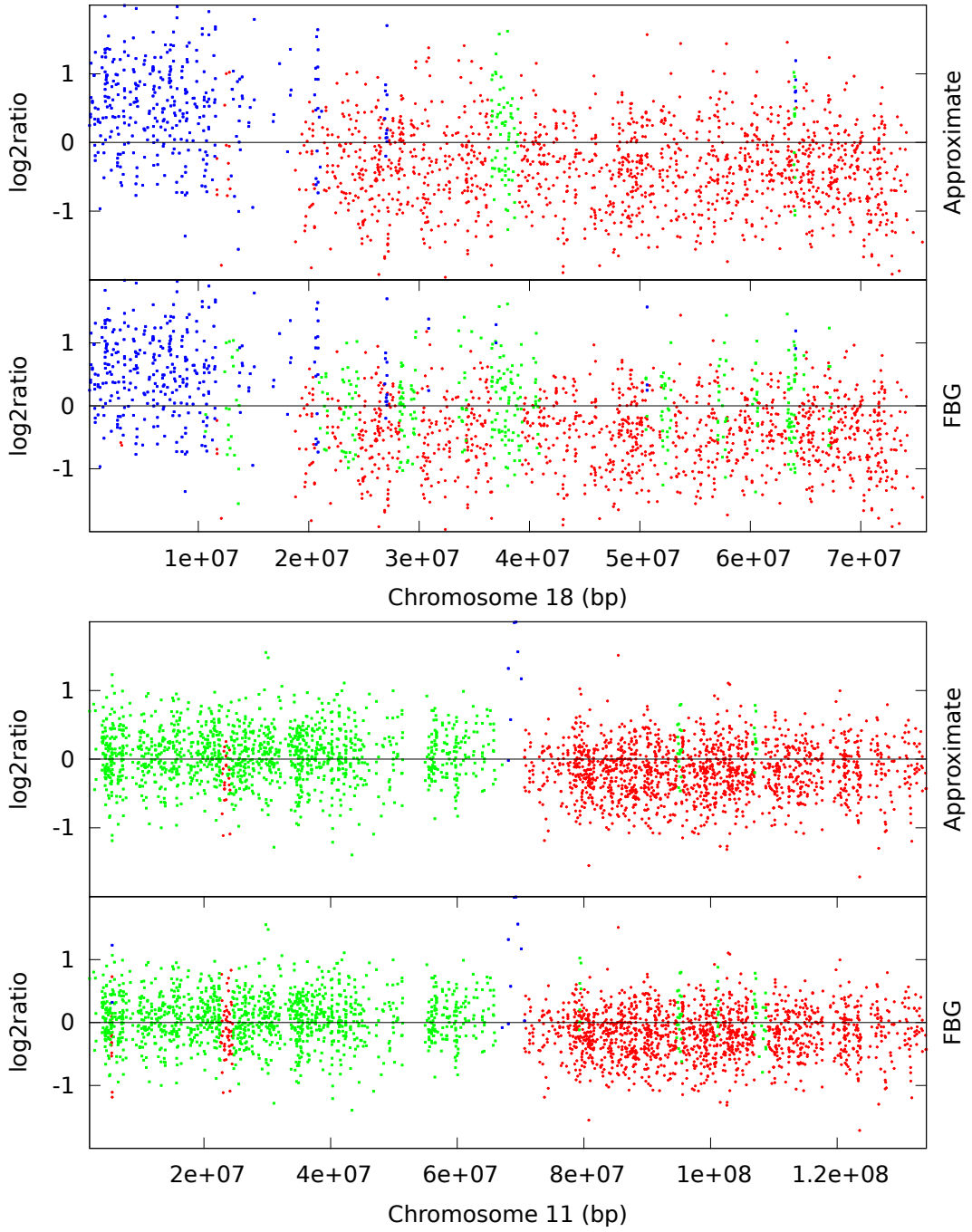



Figure 4.7: Affymetrix 100k SNP array: chromosome 18 of sample 16 and chromosome 11 of sample 33 Loss (red), normal (green), and gain (blue) clones are identified using FBG-sampling and approximate sampling. For approximate sampling $w = 1.8\sigma_D$ is used, which was recommended by the L-method.

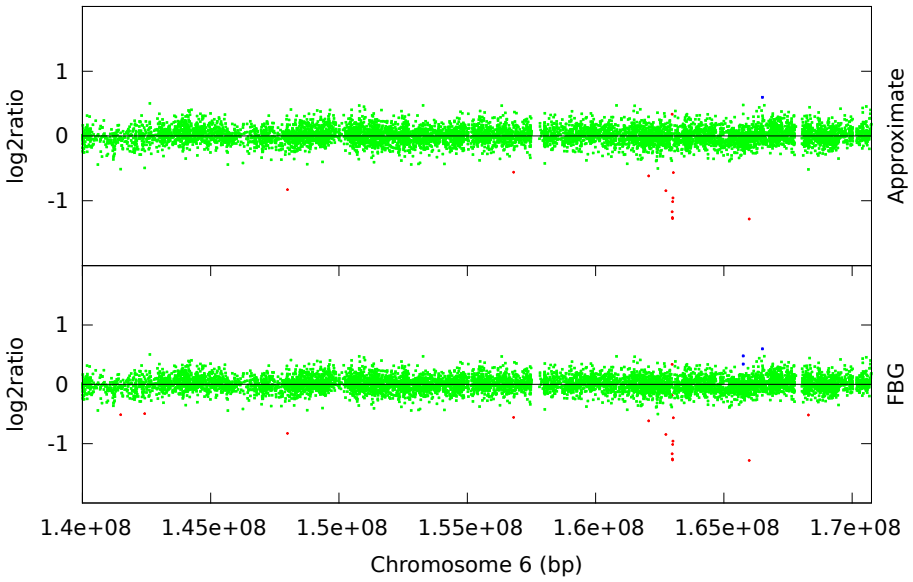



Figure 4.8: Illumina HumanMap550 array: chromosome 6 of sample 3 Loss (red), normal (green), and gain (blue) clones are identified using FBG-sampling and approximate sampling. For approximate sampling $w = 1.6\sigma_D$ is used, which was recommended by the L-method.

Table 4.3). By re-ordering the already learned state means the label switching problem can be addressed, but that increases misclassification rate due to state collapsing as the parameter values, specially means of the Gaussians, become almost identical [26]. In contrast, Bayesian methods cope with class imbalance problem by applying order constraints. Moreover, using McNemar's test [71] on the combined result of the 500 datasets we have verified that the predictions based on EM are significantly different from the predictions made by Bayesian methods with $p$-values being less than 0.001 in both cases.

### FBG vs. Approximate Sampling

In an ideal setting, like the 2-state HMM example, approximate sampling closely mimics the performance of FBG sampling up to moderate compression level. For the simulated and real dataset approximate sampling's performance is comparable to FBG's while achieving a speed-up of 10 or larger. We also observe that for higher compression levels approximate sampling reports small number of aberrant clones, which results in small $t_p$ and $f_p$ values, but large $f_n$ value. As a result, we see low recall and high precision

rate when the compression level is too high for a particular dataset (see the rows with $w \geq 4.0\sigma_D$ for HBL-2 in Table 4.2).

From Figs. 4.5, 4.6, 4.7, and 4.8 we observe that segmentations by both sampling methods are almost identical at the recommended width $w$ value. In case of HBL-2, they differ from the ground truth in some places. They predict a few extra gain regions and outliers are generally predicted as gains. We, as well as Shah *et. al.* [50], have noticed that the HBL-2 dataset has many outliers, and the variance of emission distribution of gain state 4 converges to a high value which tries to explain the outliers. In contrast, GBM has fewer outliers (see Fig. 4.6) and approximate sampling seems robust to those outliers. As the compression algorithm forces possible outliers to be included in a compressed block, it is robust to moderate frequencies of outliers.

**Width Parameter**

By varying the width parameter $w$ we can control the compression ratio $\gamma$ and the speed-up achieved by approximate sampling. But from Table 4.1 and 4.2, and Lemma 1 it is also clear that by setting a large value one can get unfavorable results. We have analyzed the effect of different $w$ values using a synthetic dataset with a controlled level of noise following [72]. Each dataset has three chromosomes with total probe counts 500, 750, and 1000. Ten aberrant regions of size 11-20 probes, randomly assigned gain or loss, are inserted in random positions of the 500 probe chromosome. Similarly, 15 aberrant regions of size 11-25 probes, randomly assigned gain or loss, are inserted into larger chromosomes. A noise component $N(0, \sigma)$ is added to the theoretical log-ratios $-1, 0, 0.58$ (loss, neutral, and gain respectively) to model the data. For a set of noise levels, $\sigma$ ranging from 0.1 to 0.5, 50 synthetic datasets are generated. We use a 3-state HMM with width parameter values in the range $0\sigma_D, \ldots, 4\sigma_D$ (where $\sigma_D$ is the standard deviation of the dataset). Signal-to-noise ratio (SNR) is defined as $\frac{0.58}{\sigma}$. In Fig. 4.9 we plot the mean compression ratio, F1-measure, recall, and precision for 50 synthetic datasets and the real biological data HBL-2. For all noise levels the compression ratio drops exponentially with increasing values of $w$. Ideally, one would like to compress as much as possible without affecting the quality of the predictions. We visually identified

a best value for width as the point after which the F1-measure drops substantially. Comparing the knee of the curve with the best value, we notice that while using the knee computed by L-method [63] is a conservative choice for width, in most cases we can still obtain a considerable speed-up.

**Outliers**

Gaussian HMMs are known to be sensitive to outliers which is evident from our results of HBL-2 and SNP arrays. Traditionally, outliers have been handled either by using a mixture distribution as the emission distribution or by preprocessing the data to remove possible outliers or impute more appropriate values. We have observed that a simple local median approach works very well to identify the outliers in a time series of $\log_2$-ratio values. Although using a mixture distribution or a distribution with fat tails, i.e. Student's-t distribution, is a better choice we lose a significant computational advantage in approximate sampling. For a block of observations $o' = o_i, \ldots, o_k$ we can compute $\prod_{j=i}^{k} P(o_j|q', \theta)$ in constant time using precomputed values $\sum_{j=i}^{k} o_j$ and $\sum_{j=i}^{k} o_j^2$ if the emission distribution is Gaussian. But it is not obvious how we can accomplish this for a more complicated distribution. Another approach, which we prefer in this context, is to use a HMM state with a very wide Gaussian and low self-transition probability to model the outliers. We have observed very good performance in this way. However, as our primary focus is to compare FBG-sampling with approximate sampling we choose to use a simple Gaussian model at the end.

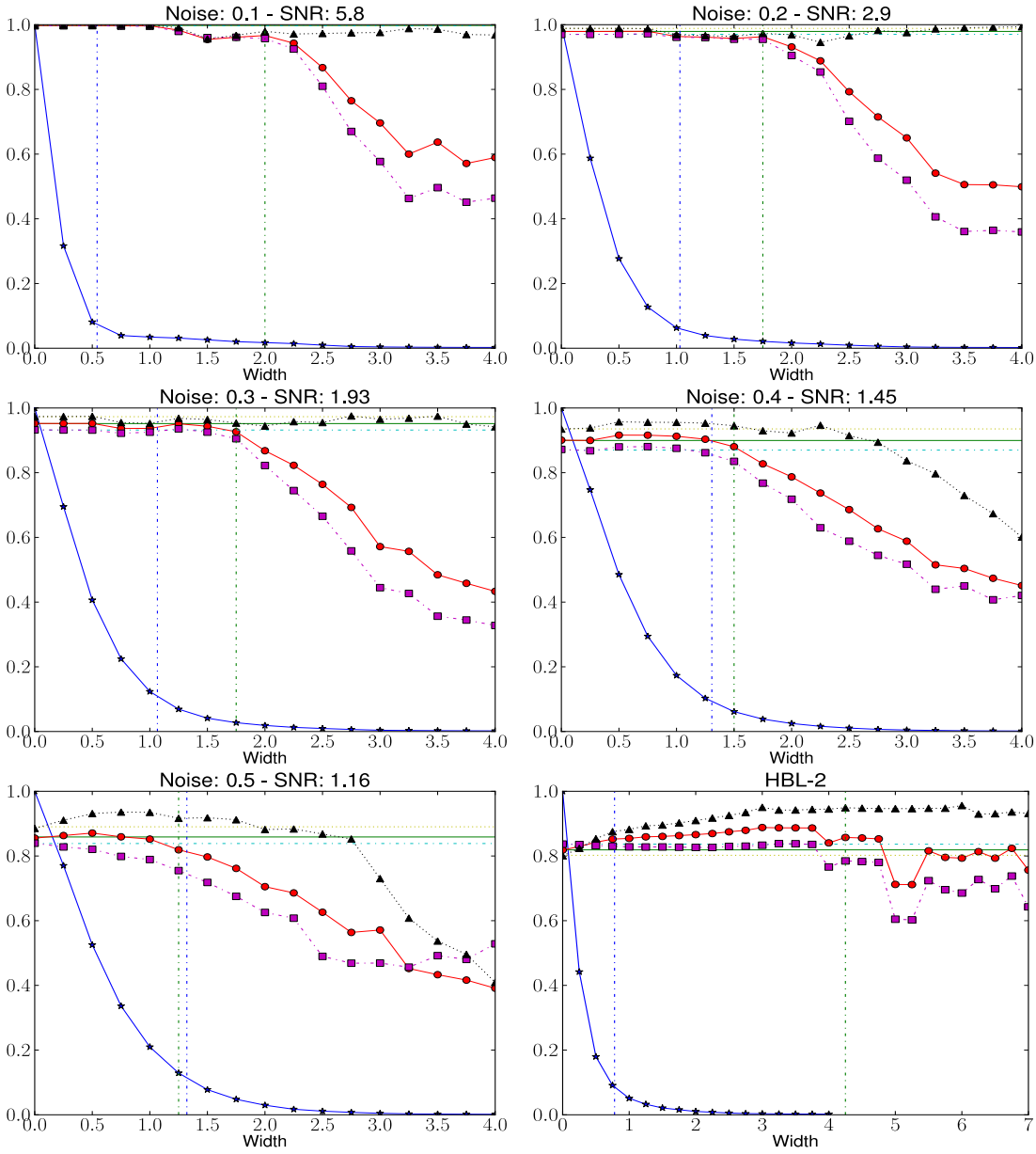



Figure 4.9: Effect of width parameter F1-measure (red, circle), recall (violet, square), and precision (black, triangle) of approximate sampling over HBL-2 and five synthetic datasets of varying noise levels are shown. For comparison, F1-measure (green, solid), recall (cyan, dashed dot), and precision (olive, dotted) of FBG-sampling are also shown as horizontal lines. For width values $0.0\sigma_D, \ldots, 4.0\sigma_D$ compression ratio $\gamma = \frac{T'}{T}$ is shown as blue line with stars. Knee is predicted using L-method and shown as a vertical line (blue, dashed dot). Vertical line (green, dashed dot) showing best width is selected by visual inspection.

Table 4.1: We show the average posterior error $\tilde{P} = \frac{1}{2T}\sum_t \sum_i |P(q_t = i|\theta^M, O) - P(q_t = i|\theta^{true}, O)|$ and total number of mismatches between the two Viterbi paths $\tilde{V}$, generated by models with parameters $\theta^{true}$ and $\theta^M$.

| Method | $w$ (in $\sigma_D$) | $\tilde{P}$ | $\tilde{V}$ | Likelihood | Time(in sec) | Speed up |
|---|---|---|---|---|---|---|
| Approx | 0.25 | 0.001 | 3 | -5470 | 74 | 1.2 |
| | 0.50 | 0.001 | 3 | -5475 | 61 | 1.4 |
| | 0.75 | 0.002 | 6 | -5469 | 35 | 2.4 |
| | 1.00 | 0.004 | 22 | -5478 | 21 | 4.2 |
| | 1.25 | 0.012 | 81 | -5588 | 13 | 6.5 |
| | 1.50 | 0.054 | 410 | -6576 | 8 | 10.4 |
| | 1.75 | 0.219 | 2345 | -8230 | 4 | 20.1 |
| | 2.00 | 0.248 | 2857 | -8492 | 3 | 34.1 |
| FBG | ... | 0.003 | 12 | -5471 | 87 | 1.0 |
| True | ... | ... | | -5470 | | ... |

Table 4.2: EM, FBG-sampling, and approximate sampling results for simulated, HBL-2, and Corriel dataset are shown here. Approximate sampling results are reported for different choices of $w$. The $w$ value which is closest to the one estimated by the L-method is shown in italic. Width $w$ is reported in $\sigma_D$ of the corresponding dataset, time is reported in seconds, and compression is defined as $\frac{T'}{T}$. For HBL-2, the initial parameter values for EM are sampled from the prior or uniform distributions, and the average (mean), most likely (ML), and best (in terms of F1-measure) performances along with likelihoods are reported.

| Dataset | Method | $w$ | F1 | Recall | Precision | Time | Compr. | Speed-up | Likelihood |
|---|---|---|---|---|---|---|---|---|---|
| Simulated | Approx | 0.50 | **0.97**±0.04 | 0.96±0.07 | **0.98**±0.02 | 27 | 0.387 | 2.2 | |
| | | 0.75 | **0.97**±0.04 | 0.96±0.06 | **0.98**±0.03 | 16 | 0.195 | 3.7 | |
| | | 1.00 | **0.97**±0.05 | 0.95±0.07 | **0.98**±0.03 | 10 | 0.097 | 5.9 | |
| | | *1.25* | 0.96±0.06 | 0.94±0.09 | **0.98**±0.03 | 7 | 0.050 | 8.8 | |
| | | 1.50 | 0.94±0.09 | 0.92±0.12 | 0.97±0.07 | 5 | 0.031 | 11.3 | |
| | | 1.75 | 0.91±0.15 | 0.89±0.18 | 0.96±0.12 | 5 | 0.023 | 12.2 | |
| | | 2.00 | 0.86±0.19 | 0.85±0.21 | 0.92±0.19 | 5 | 0.018 | 12.2 | |
| | FBG | ... | **0.97**±0.04 | 0.96±0.05 | **0.98**±0.03 | 58 | ... | 1.0 | |
| | EM prior, ML | ... | 0.96±0.09 | **0.97**±0.04 | 0.96±0.11 | 58 | ... | ... | |
| HBL-2 | Approx | *1.0* | **0.85**±0.00 | 0.83±0.00 | **0.88**±0.00 | 72 | 0.078 | 11.3 | |
| | | 2.0 | **0.87**±0.00 | 0.83±0.00 | **0.91**±0.00 | 21 | 0.018 | 39.3 | |
| | | 3.0 | **0.89**±0.00 | 0.83±0.00 | **0.95**±0.00 | 13 | 0.006 | 61.8 | |
| | | 4.0 | **0.84**±0.08 | 0.77±0.11 | **0.95**±0.01 | 13 | 0.003 | 61.9 | |
| | | 5.0 | 0.71±0.17 | 0.60±0.22 | **0.95**±0.01 | 13 | 0.002 | 64.8 | |
| | | 6.0 | 0.79±0.07 | 0.69±0.10 | **0.96**±0.01 | 14 | 0.002 | 59.3 | |
| | | 7.0 | 0.76±0.08 | 0.64±0.11 | **0.93**±0.01 | 13 | 0.001 | 61.4 | |
| | FBG | ... | 0.82±0.00 | 0.84±0.00 | 0.80±0.00 | 810 | ... | 1.0 | |
| | EM prior, ML | ... | 0.65 | **0.90** | 0.50 | 810 | ... | ... | 15158 |
| | EM prior, best | ... | 0.85 | 0.84 | 0.86 | 810 | ... | ... | 9616 |
| | EM prior, mean | ... | 0.76±0.09 | 0.86±0.03 | 0.68±0.12 | 810 | ... | ... | 13744 |
| | EM unif, ML | ... | 0.64 | **0.90** | 0.50 | 810 | ... | ... | 15159 |
| | EM unif, best | ... | 0.86 | 0.84 | 0.88 | 810 | ... | ... | 9136 |
| | EM unif, mean | ... | 0.54±0.24 | 0.77±0.21 | 0.46±0.27 | 810 | ... | ... | 13457 |
| GM05296 | Approx | 2.0 | **0.96**±0.00 | **1.00**±0.00 | **0.93**±0.01 | 3 | 0.027 | 13.0 | |
| | FBG | ... | 0.89±0.01 | **1.00**±0.00 | 0.81±0.01 | 40 | ... | 1.0 | |
| GM00143 | Approx | 2.0 | **0.98**±0.00 | **1.00**±0.00 | **0.96**±0.00 | 3 | 0.027 | 13.8 | |
| | FBG | ... | 0.89±0.24 | **1.00**±0.00 | 0.86±0.26 | 40 | ... | 1.0 | |

Table 4.3: Confusion matrices showing the proportion of accurate predictions based on EM, FBG-sampling, and approximate sampling methods on 500 simulated datasets.

| | | Truth | | |
|---|---|---|---|---|
| | | Loss | Neutral | Gain |
| EM | Loss | 0.855 | 0.071 | 0.074 |
| | Neutral | 0.001 | 0.996 | 0.003 |
| | Gain | 0.190 | 0.087 | 0.723 |
| FBG | Loss | 0.980 | 0.020 | 0.000 |
| | Neutral | 0.002 | 0.995 | 0.003 |
| | Gain | 0.002 | 0.020 | 0.973 |
| Approx. ($w = 1.25\sigma_D$) | Loss | 0.981 | 0.019 | 0.000 |
| | Neutral | 0.002 | 0.993 | 0.005 |
| | Gain | 0.009 | 0.022 | 0.969 |

# Chapter 5

# Geometric Embeddings for HTS Reads

Although microarray experiments such as arrayCGH have been effectively used for many applications including CNV detection, the resolution of such datasets are typically in the kilobase range. As a result, these datasets are not suitable for detecting mutations and small loss or gain (1-100 bp)—jointly known as insertion and deletion (indel)—in a sample genome. HTS experiments alleviates this problem by rapidly and cost-efficiently generating millions of short reads from a sample genome at single base-pair level resolution. In the last decade, HTS reads have revolutionized the field of computational biology; From discovery of SNPs to indels, from de-novo assembly to gene expression analysis, almost all area related to genomic analysis have made tremendous progress through the use of HTS datasets. Starting from this chapter, we will focus on HTS reads and discuss some of the challenges, and our contributions, related to analyzing these datasets.

In resequencing experiments, the first step is mapping HTS reads to a reference genome–the process is known as read mapping. Approximate string matching, the theoretical problem underlying read mapping, is arguably one of the most fundamental problems in bioinformatics, and a very well-studied area in data mining; for surveys see [73, 74]. Mapping reads from sequencing experiments requires solving approximate string matching problems for billions of short DNA sequences of length 50–500bp against entire genomes. Out of the variety of different approaches (see [75] for a detailed taxonomy) proposed for approximate string matching, current read mappers rely on only three different paradigms [76]: seed-and-extend (encompassing hash tables and $q$-gram filtering), prefix/suffix tries (using the Burrows-Wheeler transform [77] and FM index [78]), and one approach based on merge sort [79]. Their computational efficiency

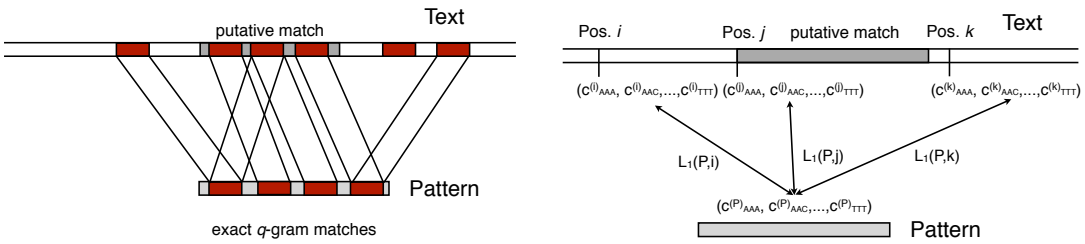



Figure 5.1: Most approaches to approximate string matching using Ukkonen's $q$-gram lemma rely on the existence of reasonably large $q$-grams which are exact matches between pattern and text. These can be found efficiently with a number of techniques and yield putative hits which are then evaluated using an alignment algorithm. For each pattern and each putative hit the number of shared $q$-grams is evaluated *de novo* (left). We map both reads and genome locations to vectors of 3-gram frequencies and identify approximate matches finding nearest neighbors (right). This is accelerated by the use of a spatial index structure, e.g. a *kd*-tree which is created by recursively partitioning the input space around the median value of a dimension.

depends on the existence of *exact* matches between the read and the genome. As one of the contributions of this thesis, we introduce a new strategy for read mapping and present a new readmapper named TreQ, following ideas first proposed for protein sequences [80] and generally referred to as *vector space frequency distance methods* [74], embedding strings as $q$-gram frequency vectors.

This chapter is organized as follows. We start with a discussion on related work. In Section 5.2, we show how $L_1$ distance serves as a lower bound for affine gap costs. In Section 5.3, we introduce our methodology and implementation details. We provide a detailed analysis on both real and simulated data to show the advantages and drawbacks of geometric embeddings in Section 5.4.

## 5.1 Related Work

Intuitively, there cannot be an approximate match of small edit distance between a read and the genome if not one or several exact matches of length $q$ exist. The relationship between the presence of such matching $q$-grams (sequence of length $q$) and the edit distance was revealed in a paper by Ukkonnen [81]: a lower bound for the edit distance between two strings is given by the $L_1$ distance between their count vectors of $q$-grams (for $q = 3$ these are the trinucleotide frequency vectors). This provides the basis for

a seed-extend strategy of using efficient algorithms for finding one initial exact $q$-gram match, exploring whether additional exact $q$-gram matches support the existence of an approximate match, see Fig. 5.1 (left), and then use an efficient alignment algorithm, such as Myers' bit-vector algorithm [82], to verify and assess the quality of the match. Existing methods either implement this idea of $q$-gram filtering [83] directly [84–86], or implicitly rely on it through suffix trie based data structures [87–89].

The running times depend on the maximal edit distance permitted: smaller maximal edit distance allows to chose larger $q$, thus there will be fewer exact $q$-gram matches and putative approximate matches to explore; indeed their probability decreases exponentially. If we think of patterns being derived from a match in the text through edit operations of technical nature (sequencing errors), or biological nature (genetic variants), the probability of hitting all $q$-grams and thus rendering $q$-gram filtering useless increases with the number of edits [90]. Spaced seeds [91] and gapped $q$-grams [92], requiring exact matches in a fixed pattern of $q$ out of $q' > q$ positions, are one way of addressing this. Most popular approaches however strictly limit the maximal edit distance and use heuristics to keep running time in check at the potential cost of missing best matches.

In particular the detection of indels suffers from the limits on edit distance of matches. Many of the existing methods have problems in mapping 100bp reads with indels to the reference genome; longer reads improve the situation for some approaches. Consequently, the state-of-the-art in the detection of structural variants is the use of *paired-end* read libraries and advanced methods for performing downstream analysis after mapping the paired-end read libraries to reference genomes [93–97]. Nevertheless, Alkan *et al.* [98] noted that, in particular, detection of small, 5–50bp indels is a largely open problem, although our analysis reveals that one recent approach, Stampy [99], provides excellent sensitivity. In the detection of such short indels the deviations from mean insert length are measured and thus the sequencing coverage required to arrive at statistical significance is inversely proportional to the indel length. Our results will show that the detection of 1–16bp indels from *single reads* is possible using $L_1$ distance.

We choose $q = 3$ and consider vectors of *all* trinucleotide frequencies, by embedding

reads of length between 100–1000bp as vectors in $\mathbb{R}^{64}$. The problem of finding a minimal edit distance approximate match now becomes the problem of finding a nearest neighbor in a data set of vectors derived from a genome by sliding a window over the genome and mapping the sequence to a frequency vector, see Fig. 5.1 (right). Finding (approximate) nearest neighbors however has been well studied and a large range of spatial index data structures have been proposed [100–103] generally leading to $O(n \log n)$ complexity for construction of the spatial index and $O(\log n)$ complexity for nearest neighbor queries, where $n$ denotes the number of points in the index. Empirical running times however vary widely based on the detailed structure of the problem instance and thus algorithm engineering is important for achieving competitive running times.

The vector space frequency distance method introduced in [80] was not further pursued except in a small scale study focusing on different ways to map strings to vectors [104, 105]. In recent years, researchers in databases, both multi-media and text, investigated indices in high-dimensional spaces [106–110], but the small alphabet size of DNA which leads to non-sparse frequency vectors preclude their use here. Boytsov *et al.* [74] implemented and evaluated a range of different approaches in approximate string matching also on DNA datasets which are of small bacterial genome size (3.2Mbp). We found that his findings do not translate when the genome size increases by a factor of 1,000. For instance, the effects of cache or page misses, which motivate cache-oblivious data structures that guarantee minimum number of cache misses irrespective of cache size and memory hierarchy, are simply not observable on small data sets. During the development of the method we used state-of-the-art *kd*-tree libraries [111, 112], but found them to be lacking in performance once the index contained more than a few million points. Indeed, on genome-size problems, the ability to effectively implement data structures in a cache-oblivious manner is more important than computational complexity.

## 5.2 The $q$-gram Lemma revisited

We use the usual notation, following [113]: A finite set of characters $\Sigma = \{a, b, c, \ldots\}$ an *alphabet* and a sequence $s$ of characters from $\Sigma$ a *string*. We denote by $|s|$ its *length*, by

$s_i$ its $i$-th character, $i > 0$, and by $s[i, j]$ the continuous *sub-string* starting at position $i$ and ending in position $j$.

We associate strings with vectors by computing the frequencies of all $q$-grams,

$$c_q(s) := \left( \left| \{ i \in \{1, \ldots, |s| - q + 1\} : s[i, i + q - 1] = w \} \right| \right)_{w \in \Sigma^q} \tag{5.1}$$

which define a map from $\Sigma^\star \to \mathbb{R}^{|\Sigma|^q}$ through $s \mapsto c_q(s)$. We assume that the $q$-words are in lexicographic order. We obtain a distance from the $q$-spectrum by considering the $L_1$ distance of the count vectors in $\mathbb{R}^{|\Sigma|^q}$, $L_1(s, t) := |c_q(s) - c_q(t)|$.

**Edit distance.** The edit distance $\mathrm{ED}(s, t)$ between two strings $s$ and $t$ is determined by the minimal number of edit operations—substitutions, insertions and deletions—necessary to transform one into another. We can notate the edit operations as rewriting rules, $a \to \epsilon$ is a deletion, $\epsilon \to a$ an insertion and $a \to b$ a substitution. Here, $a, b \in \Sigma$, $a \neq b$ and $\epsilon$ is the empty string. What is usually referred to as *the* edit distance is indeed the Levenshtein distance which assigns unit costs to the three possible operation. This of course generalizes to arbitrary costs $c_s$, $c_i$ and $c_d$ for substitutions, insertions and deletions respectively.

For $s, t \in \Sigma^q$, Ukkonen's lemma [81] states that $L_1 \leq 2q\,\mathrm{ED}(s, t)$[1], but this bound is dominated by the mismatches. It is worthwhile to consider the effects of mismatches and indels separately. Consider two strings $S_1, S_2$, where $S_2$ is derived by a single deletion of size $d$ from $S_1$, or by insertion. Then the $L_1$ distance is comprised of two components. First, the number of $q$-grams spanning the gap in $S_2$ is $q - 1$. Second, for $S_1$, the first character in the deletion accounts for $q$ deletions of $q$-grams, however, every consecutive character only accounts for one, since the other $q - 1$ are already accounted for by deleting the left neighbor. Hence, the $L_1$ distance is bounded by $L_1 \leq 2q + d - 2$. As a single mismatch can affect at most $2q$ number of different $q$-grams, it follows that

[1]Note that this bound can become arbitrarily bad, for example when $t$ is a rotation or transposition of $s$, see [81].

for $m$ mismatches and $g$ gaps of size $d_i$, $1 \leq i \leq g$,

$$L_1 \leq \sum_{i=1}^{g}(2q + d_i - 2) + 2qm. \tag{5.2}$$

Since $\sum_{i=1}^{g} d_i + m = \mathrm{ED}$, we obtain

$$L_1 \leq \mathrm{ED} + (2q-2)g + (2q-1)m \tag{5.3}$$

This shows that the number $g$ of contiguous gaps (*not* the total number of gapped positions!), provides a sharper bound than the number of mismatched positions. For example, if $\mathrm{ED} = 4$ and $q = 3$, then $L_1 \leq 8$ for a single indel of size 4, but $L_1 \leq 24$ for 4 mismatches. Any algorithm based on nearest neighbors under the $L_1$ distance is thus very well suited for mapping reads with large indels.

A preference for large indels in alignments is biologically more meaningful than alignments with many small indels, and generally addressed by using affine gap costs. The above formula naturally implies a scoring scheme for an affine edit distance $\mathrm{AED}(s,t)$. Since

$$L_1 \leq (2q-1)g + \left(\sum_{i=1}^{g} d_i - g\right) + 2qm, \tag{5.4}$$

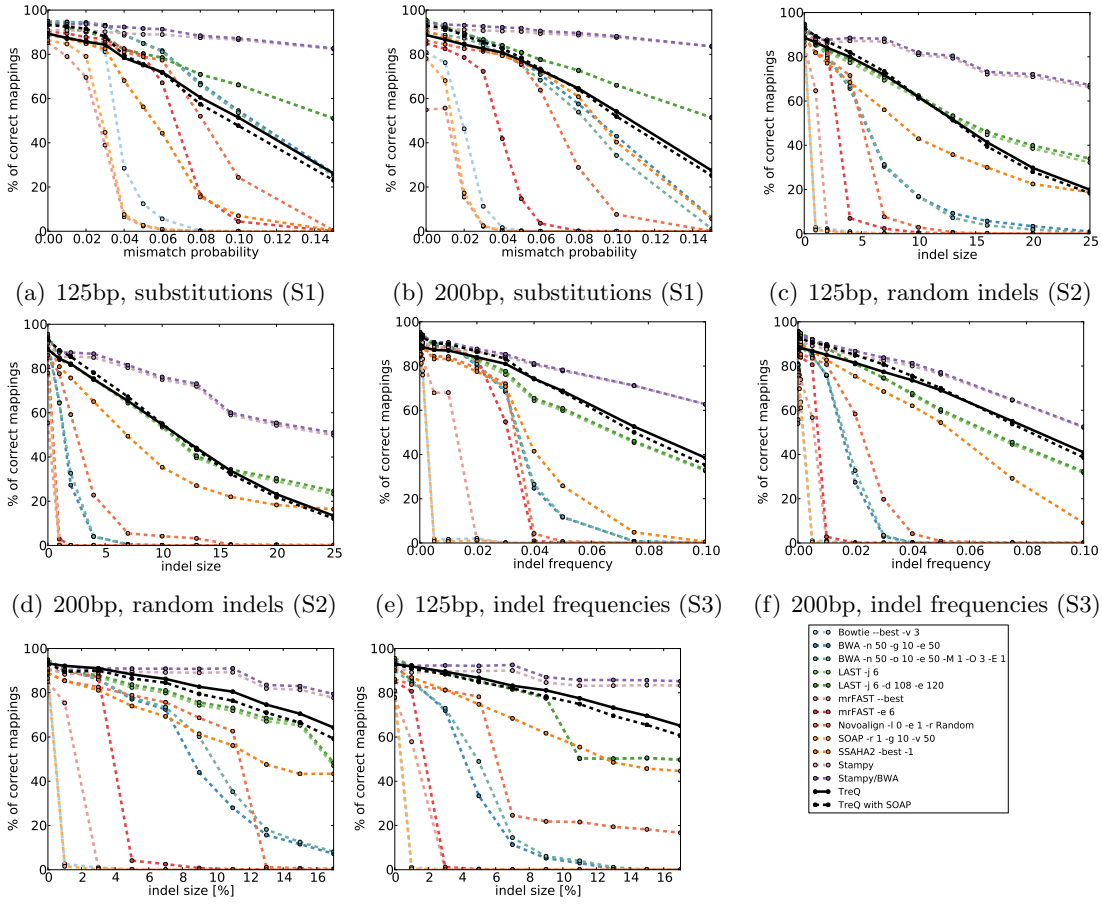
we obtain

$$L_1 \leq \mathrm{AED} := c_o g + c_e \left(\sum_{i=1}^{g} d_i - g\right) + c_s m \tag{5.5}$$

for gap opening cost $c_o := 2q - 1$, gap extension cost $c_e := 1$ and substitution cost $c_s := 2q$.

## 5.3 Read mapping with cache-oblivious *kd*-trees

Efficient searches for *exact* or *inexact* nearest neighbors in high dimensions generally involve creating a tree-like index structure that recursively partitions the space. Their efficiency depends on the quality of the index and the geometric distributions of points. A comparative study [114] showed that in a wide range of practical instances *kd*-trees outperform more advanced methods [100–103].




(a) 125bp, substitutions (S1) (b) 200bp, substitutions (S1) (c) 125bp, random indels (S2)

(d) 200bp, random indels (S2) (e) 125bp, indel frequencies (S3) (f) 200bp, indel frequencies (S3)

(g) 125bp, single indel (S4) (h) 200bp, single indel (S4)

Figure 5.2: **Comparison of popular read mappers with TreQ.** Accuracy is defined as the percentage of single best reads that are mapped to the exact genomic location they were drawn from in the simulation. Notice that TreQ outperforms most popular read mappers except Stampy, and is mostly on par with LAST.

In the index generation step, for each sub-tree of the *kd*-tree, a dimension—usually the one with the highest variance—is chosen. Then the set of points under the sub-tree is partitioned using the median value of the chosen dimension as pivot. This process continues recursively and eventually completes in $O(dn \log n)$ time for $n$ $d$-dimensional points. During the search step, a query point's corresponding coordinate is compared to the pivot and a decision to search the left or right sub-tree is made. If there is no exact match to be found, the search procedure backtracks.

As the index size gets larger the effect of cache misses becomes very prominent and the running time increases substantially. As a result, over the last decade many important data structures including *kd*-trees were made cache-oblivious [115, 116]. Our cache-oblivious *kd*-tree implementation stores the tree in sequential memory using the van Emde Boas layout [117, 118] which guarantees an optimal number of cache misses.

We have implemented the following modification to the usual *kd*-tree construction. During index generation we use pre-selected dimensions based on the entropy of the dimensions over the full dataset, which makes the index building step $O(n \log n)$ instead of finding the dimension with highest variance. The minimal axis-parallel hyper-rectangle containing all the points in the subtree defines a bounding box per subtree. These bounding boxes help to reduce the search space at the cost of increasing the index generation to $O(dn \log n)$. In the search step we compute lower bounds for the $L_1$ distance between the query point and *all* the points inside the bounding box of the left and right sub-tree in $O(d)$ time at every subtree. We proceed with the sub-tree giving the best lower bound and store a pointer to the other sub-tree in a min-heap for future consideration (note that, since there is small finite number of possible lower bounds, we implement min-heap using a simple array indexed by the lower bounds). If no exact match is found the search process becomes expensive. We bound the number of alternate paths searched per query with the parameter $\beta$, which bounds the running time per query to $O(\beta d \log n)$.

In the bottom levels, the running time overhead to find lower bounds using the bounding boxes is comparable to directly computing the $L_1$ distances. Thus, for the last $\tau$ levels we do not create bounding boxes, and in the search step we simply compute

the $L_1$ distances between the query point and the points in the sub-tree (we use fast hardware accelerated $L_1$ distance computation). This also decreases memory requirement to store the tree by $2^\tau$ times.

**Verification by Myers' Bitvector Algorithm**

Using Myers' bit-vector algorithm [82] we compute the edit distance and the exact location for each read based on putative locations identified as nearest neighbors, adding a slack of 14bp on each boundary. The best position is reported, breaking ties arbitrarily.

**Parameter Choices**

Parameter $\beta$, the maximum number of different paths explored in the search, controls the running time and sensitivity of TreQ. However, this does not restrict the maximal edit distance of matches in contrast to trie-based methods which avoid exponential blow-up with such restrictions. A second parameter, $\tau$ influences the memory footprint and running times, as the lowest $\tau$ levels of the cache-oblivious $kd$-tree are not stored and rather direct $L_1$ distance computations are performed. To further reduce memory requirements, the genomic window is shifted by $g$ base pairs to create $d$-dimensional ($d = 4^q$) frequency vectors (final match positions are based on Myers' alignment). A third parameter $\alpha$ determines number of vectors for which we only store the changes from a nearby point as they can be constructed with minimal overhead from their differences in few dimensions from the $(2\alpha+1)$-$th$ point which is actually stored. In our experiments we have found $g = 3$, $\alpha = 2$, $\beta = 3000$ and $\tau = 3$ to be a good choice for the human genome, and unless otherwise stated these are the default parameter values for TreQ. Note that for a wide range of parameter choices TreQ's accuracy remains effectively the same (cf. Fig. 5.3).

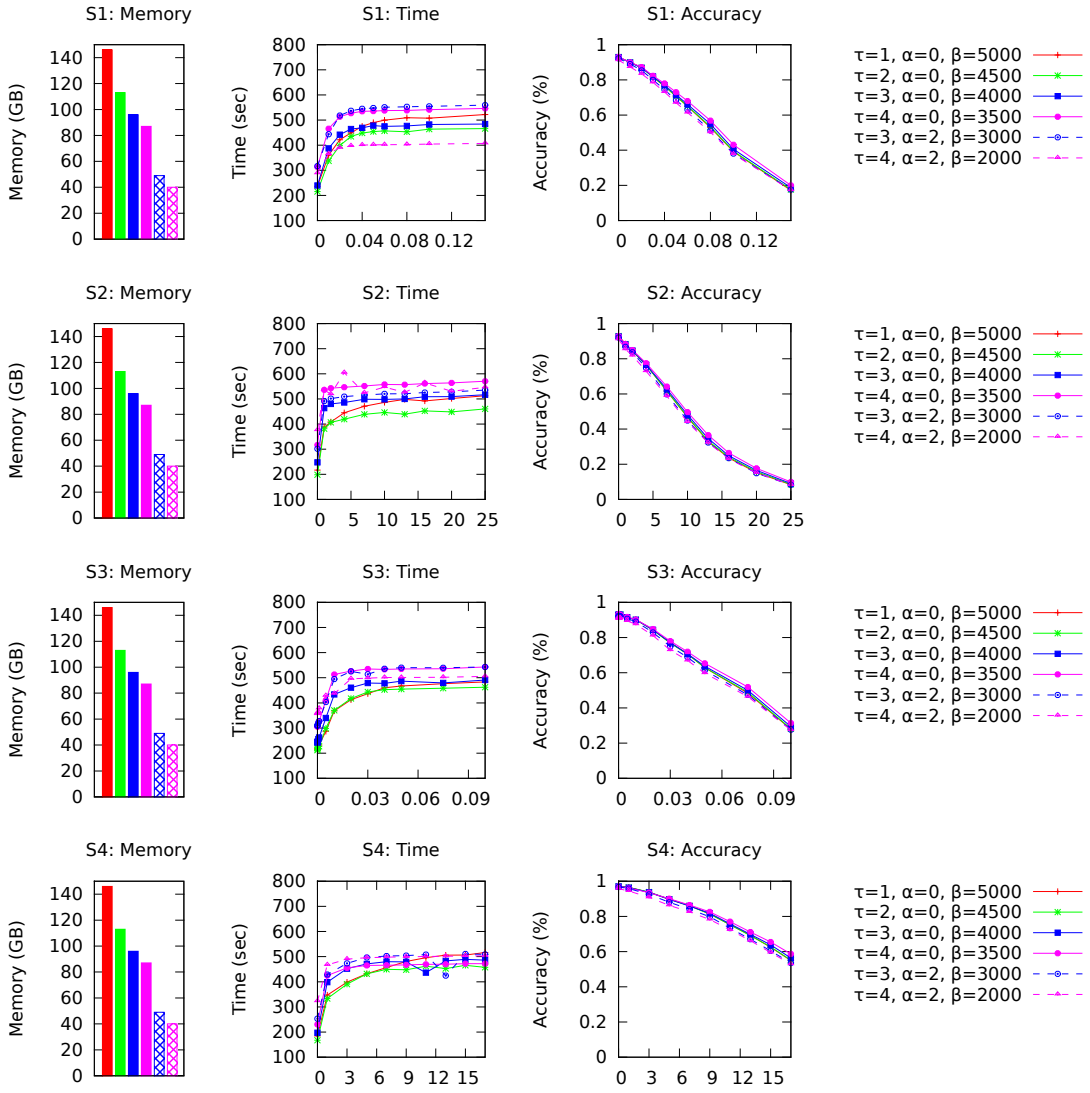



Figure 5.3: Effect of different parameters on TreQ. Simulated datasets S1, S2, S3, and S4 (for details see Section: Discussion) of read length 100bp are tested with different parameter choices for TreQ. The memory requirement for TreQ comes down from 150GB ($\tau = 1$, $\alpha = 0$) to 40GB ($\tau = 4$, $\alpha = 2$) while a careful selection of $\beta$ achieves equivalent mapping accuracy in similar amount of time.

Table 5.1: Running times for read mappers used for evaluating simulated data.

| Read mapper | S1 | | S2 | | S3 | | S4 | |
|---|---|---|---|---|---|---|---|---|
| | 125 | 200 | 125 | 200 | 125 | 200 | 125 | 200 |
| Bowtie (-v 3) | 0:05 | 0:03 | 0:03 | 0:03 | 0:04 | 0:04 | 0:03 | 0:06 |
| BWA | 0:07 | 0:04 | 0:11 | 0:07 | 0:08 | 0:10 | 0:13 | 0:29 |
| BWA (-n 50 -o 10 -e 50 -M 1 -O 3 -E 1) | 1:30 | 2:34 | 4:30 | 5:55 | 3:14 | 4:42 | 3:44 | 4:57 |
| SOAP2 (-r 1 -g 10 -v 50) | 0:06 | 0:03 | 0:05 | 0:05 | 0:06 | 0:05 | 0:03 | 0:08 |
| mrFAST (–best) | 0:52 | 0:46 | 0:56 | 0:48 | 1:19 | 1:20 | 0:48 | 1:08 |
| mrFAST (-e 6) | 1:54 | 1:28 | 1:41 | 1:37 | 2:47 | 2:28 | 1:49 | 2:10 |
| Novoalign (-l 0 -e 1 -r Random) | 0:08 | 0:12 | 0:14 | 0:23 | 0:10 | 0:17 | 0:16 | 0:23 |
| SSAHA2 (–best -1) | 4:30 | 8:42 | 5:16 | 9:23 | 8:21 | 16:43 | 6:50 | 14:38 |
| TreQ ($\tau = 1$, $\beta = 10000$, $\alpha = 0$) | 2:29 | 2:55 | 2:37 | 3:01 | 2:33 | 3:03 | 2:40 | 2:31 |
| LAST w/ LAMA | 0:43 | 1:51 | 0:33 | 1:18 | 0:57 | 2:32 | 0:40 | 1:46 |
| LAST w/ LAMA (-d108 -e120) | 1:07 | 2:29 | 0:52 | 1:52 | 1:27 | 3:25 | 0:59 | 2:25 |
| Stampy | 0:18 | 0:31 | 0:37 | 1:13 | 0:40 | 1:16 | 0:36 | 1:05 |
| Stampy w/ BWA | 0:10 | 0:19 | 0:41 | 1:18 | 0:30 | 1:00 | 0:40 | 1:12 |

## 5.4 Experiments

To evaluate TreQ and compare its performance, we ran a number of read mappers–Bowtie [119] (version 0.12.7), BWA [87] (version 0.5.9), SOAP2 [120] (version 2.21), mrFAST [121] (version 2.1.0.0), Novoalign (http://www.novocraft.com) (version 2.07.13), SSAHA2 [122] (version 2.5.5), LAST [123, 124], Stampy [99] (version 1.0.19) and RazerS [85]–on simulated and real read datasets (see Table 5.1 for version numbers). These read mappers were evaluated with their default and, in some cases, customized parameters for allowing maximal permissible edit distance. We also forced them to report one single best hit. For TreQ, parameters $q = 3$, $d = 64\,(= 4^q)$, $g = 3$, and $k = 200$ were fixed throughout the experiments. Currently, quality scores are ignored in the match evaluation phase of TreQ. For simulated data we define accuracy as the percentage of reads mapped to the actual genomic locations from where they were sampled (Fig. 5.2). We use human genome HG18 build 36 as the reference for all the experiments.

### 5.4.1 Simulated Data

We simulated four different read datasets, comprising a set of different model parameters, each by sampling 10,000 reads from the human reference genome. Given read length $\ell$, we have:

- **S1** 0 to 15$\ell$% single nucleotide substitutions at random positions

- **S2** Indels of size 0 to 25 at $2\ell\%$ random locations in the read
- **S3** Indels of size 2 at 0 to $10\ell\%$ random locations in the read
- **S4** A single indel of size 0 to $17\ell\%$ at a random location

On top of that, we simulated sequencing error by estimating and interpolating an Illumina error profile. We estimated a first-order Markov chain of Phred score transitions from 1 million reads of length 101 from a Yoruba African individual (NA18507) for each of those 101 positions. In order to remove noise and simulate reads of lengths other than 101bp, we used univariate spline interpolation to estimate the evolution of each entry of the matrix over the read sequence. These functions were then stretched or skewed for other read lengths, and new transition probabilities were derived by evaluating the spline function at the appropriate positions and rescale the rows so that the matrix becomes stochastic. The resulting Phred score distributions of the reads simulated by iterating the new Markov chains were then verified to correspond to those of real data. Since Phred scores correspond to actual error rates [125], we used them to simulate position-dependent sequencing errors.

Our evaluation shows that Stampy outperforms all other methods in terms of accuracy (Fig. 5.2). Its running time however depends on the type of read. For instance, mapping all S1 reads of 200bp length takes 0h31m, whereas mapping S2 takes 1h13m (Table 5.1). TreQ's running time ranges consistently at about 3 hours. Stampy's running time also increases with read length, although it is mostly better than TreQ's. LAST performs in about the same accuracy range as TreQ, but is not always competitive in running time.

In terms of accuracy, geometric embedding outperforms all suffix trie and seed-extend-based read mappers other than Stampy and LAST on almost all instances. The authors of both these programs argue that their accuracy is mainly due to the elaborate downstream analysis they perform after finding candidates. Our results are thus preliminary, since in essence we are comparing statistical alignment models of Stampy and LAST to a simple filter based on Levenshtein distance in our case. BWA can be customized to be competitive on S1 for shorter reads, at the cost of higher

running times, but this improvement does not translate to S2-S4. Similarly, LAST outperforms TreQ on S1, but its performance is similar or worse than TreQ's on other conditions. Trie-based read mappers (Bowtie, BWA, and SOAP2) are very fast but do not perform well in general. Using a hybrid approach–in which SOAP's unmapped reads are mapped by TreQ–increases the accuracy for lower distances at the expense of slightly lower accuracy for higher distances, while drastically reducing the running time.

### 5.4.2 Biological Data

Following the evaluation in [126], we compared TreQ to popular read mappers on a set of 1 million randomly selected 101bp reads from Yoruba African individual (NA18507) [127]. The running time and the percentage of reads mapped to the reference human genome HG18 build 36 within 3, 6, 12, and 18 edit distances are reported in Table 5.2. Times are for a single thread on a single core of a 2.2 GHZ AMD Opteron processor.

We have found suffix-trie based read mappers, implemented using Burrows-Wheeler transform, to be very fast on the real dataset, but, unsurprisingly, limited in their ability to map reads with higher edit distances. Seed-extend-based techniques in contrast usually map more reads at large edit distances but require more CPU time. Except LAST, Stampy, and TreQ, none of the other read mappers that we have evaluated successfully maps reads at high edit distances, possibly containing indels, in a reasonable amount of time. TreQ does so at competitive running time compared to mrFAST, RazerS, SSAHA2 and BWA with customized parameters (BWA's default parameters are not competitive with respect to accuracy). The hybrid SOAP/TreQ approach, taking advantage of suffix-trie based read mappers' efficiency on low edit distances and TreQ's sensitivity at higher edit distances, uses less time and maps more reads within all edit distances considered.

Although TreQ outperforms other read mappers on many mismatches or with large indels, it's performance start to degrade gradually. As a result, TreQ's specificity should drop at large edit distances. We have indirectly tested TreQ and other read mappers'

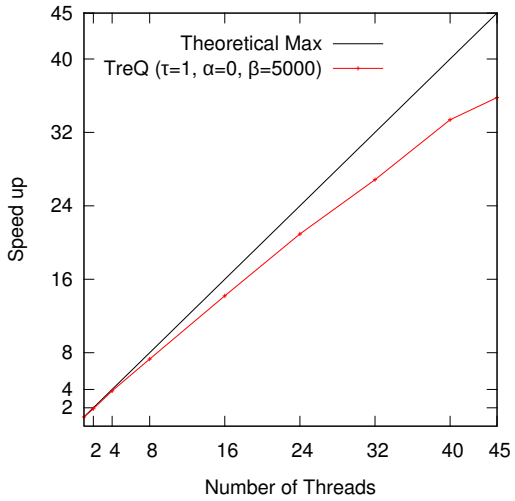



Figure 5.4: Speed up achieved by the multi-threaded version of TreQ on the task of mapping a randomly selected 0.1 million 101bp single end reads from Yoruba African individual (NA18507).

specificity by combining the genome of human and chicken (a distant organism from human) and mapping the same one million real reads to this combined genome. We have found that BWA, BWA (customized), SOAP2, Novoalign, mrFAST, and TreQ (within edit distance 18) map 87, 191, 65, 149, 189, and 187 reads respectively to the chicken genome. This experiment shows that TreQ's specificity is comparable to the other read mappers. For greater control over specificity, an optional maximum edit distance threshold $m$ can be set in TreQ to discard any alignment with edit distance greater than $m$ (default value $\frac{l}{4}$, for read length $l$).

### 5.4.3 Discussion

#### Multi-threading

We have developed a multi-threaded version of TreQ and tested its performance by mapping 0.1 million randomly selected Illumina single end reads (Yoruba African individual, NA18507) on a 48-core AMD Opteron 2.2 GHz server with 256GB memory. The performance of TreQ scales very well in the number of threads, within 84% of the achievable maximum up to 40 cores (see Figure 5.4).

Table 5.2: One million randomly selected Illumina single end reads mapped to HG18 build 36. The percentages of reads mapped within a fixed edit distance (ED) by various read mappers are reported. As expected, trie-based read mappers are very fast, but mostly fail to map reads with higher errors. BWA with customized parameters performs well, but with significantly increased running time. Seed-extend based methods have varied outcomes; mrFAST, RazerS, and SSAHA2 take significantly more running time than others, Novoalign is comparably fast but fails to map reads with higher edit distances, while LAST (without LAMA option) and Stampy map almost similar amount of reads as TreQ. In contrast to most read mappers, TreQ is not restricted to few mismatches, small indels or few number of indels, and maps either an almost similar percentage of reads or more with various different parameter settings. TreQ's running time is significantly lower than mrFAST, RazerS and SSAHA2, and comparable to customized BWA; we stopped SSAHA2 after it did not finish running in 45 hours. Additionally, Hybrid TreQ/SOAP outperforms most read mappers while significantly reducing the required running time. Note that Bowtie only allows mismatches and is restricted to at most 3. All running times are based on running the read mappers single-threaded on a single core of a 2.2 GHZ AMD Opteron processor.

| Technique | Algorithm | Parameters | Time (h:m) | Mapped percentage $\leq$ ED | | | |
|---|---|---|---|---|---|---|---|
| | | | | $\leq 3$ | $\leq 6$ | $\leq 12$ | $\leq 18$ |
| Suffix trie | Bowtie* | `--best` | 0:04 | 85.22 | – | – | – |
| | | `--best -v 3` | 0:04 | 86.85 | – | – | – |
| | BWA | default | 0:14 | 87.31 | 89.35 | – | – |
| | | `-n 50 -o 10 -e 50 -M 1 -O 3 -E 1` | 5:53 | 87.35 | 90.08 | 92.39 | 93.03 |
| | SOAP2 | `-v 50 -g 10 -r 1` | **0:03** | 84.87 | – | – | – |
| Seed-extend | mrFAST | `--best -e 6` | 19:50 | 87.54 | 90.59 | – | – |
| | Novoalign | `-l 0 -e 1 -r Random` | 0:27 | 83.68 | 84.80 | 85.18 | 85.19 |
| | SSAHA2 | `--best -1` | 45:36+ | – | – | – | – |
| | RazerS | `--unique` | 14:45 | 66.67 | 79.41 | – | – |
| | Stampy | default | 1:57 | 85.73 | 88.32 | 90.90 | 92.15 |
| | | `--bwa-options` | 0:38 | **90.37** | **92.05** | **93.81** | **94.84** |
| | LAST | default | 1:32 | 84.76 | 87.66 | 90.23 | 90.78 |
| | | `-d108 -e120` | 1:35 | 84.85 | 87.77 | 90.69 | 91.69 |
| | | default, LAMA | 4:36 | 68.74 | 71.12 | 73.26 | 73.72 |
| | | `-d108 -e120`, LAMA | 4:39 | 39.26 | 40.60 | 41.95 | 42.42 |
| Geometric embedding | TreQ | $\tau = 1,\ \beta = 5000,\ \alpha = 0$ | 7:00 | 87.34 | 90.12 | 93.06 | 94.67 |
| | | $\tau = 2,\ \beta = 4500,\ \alpha = 0$ | 6:28 | 87.27 | 90.06 | 93.01 | 94.61 |
| | | $\tau = 3,\ \beta = 4000,\ \alpha = 0$ | 6:36 | 87.22 | 90.04 | 93.02 | 94.62 |
| | | $\tau = 1,\ \beta = 5000,\ \alpha = 2$ | 8:15 | 87.32 | 90.11 | 93.04 | 94.66 |
| | | $\tau = 2,\ \beta = 4000,\ \alpha = 2$ | 7:44 | 87.16 | 89.93 | 92.87 | 94.50 |
| | | $\tau = 3,\ \beta = 3000,\ \alpha = 2$ | 8:50 | 86.90 | 89.69 | 92.66 | 94.30 |
| Hybrid | SOAP2 + TreQ | $\tau = 3,\ \beta = 3000,\ \alpha = 2$ | 2:06 | 87.89 | 90.50 | 93.26 | 94.83 |

**Memory Requirements**

The memory requirement for the cache-oblivious $kd$-tree and the $d$-dimensional vectors are $d\,2^{\lceil \log \frac{G}{g} \rceil - \tau}$ and $\frac{G}{g(2\alpha+1)}(d + 2g\alpha(\alpha+1))$, respectively. As a result, TreQ requires about $O\big(\frac{dG}{g}\big(\frac{1}{2^\tau} + \frac{1}{\alpha}\big)\big)$ bytes of memory, given that $2g\alpha(\alpha+1) \leq d$. Here $G$ is the genome size, $g$ is the offset by which the genomic windows are shifted, see Fig. 5.1 (right), while creating $d$-dimensional $q$-gram vectors, $2\alpha$ is the number of vectors for which we only store the changes from a nearby point, and $\tau$ is the number of ignored lowest levels in the $kd$-tree. If we set $\tau = 4$, $\alpha = 2$ and $g = 3$, for $d = 64$ TreQ's memory requirement is around 40GB (which is equivalent to using less than 1GB per core in a 48-core machine with the multi-threaded TreQ). Interestingly, these parameters have minor effects on accuracy and running time; for a detailed analysis see Figure 5.3. In addition, memory requirement can be further reduced by creating separate $kd$-trees for each chromosome and loading one $kd$-tree at a time in the memory.

**Memory reduction:**

We create $d$-dimensional $q$-gram frequency vectors by shifting a window of size $l$ (read length) over a genome of size $G$. This naive approach uses $O(Gd)$ amount of memory. To reduce this large memory requirement for storing count vectors, we apply two techniques. First, we only consider every $g$-th genomic window for computing frequency vector, which brings down the memory requirement to $O(\frac{Gd}{g})$. Second, we observe that consecutive $g$-th windows can have at most $2g$ differences in their frequency vectors. We exploit this observation by not storing the left $\alpha$ vectors and right $\alpha$ vectors of a particular vector $V$. Instead we define these $2\alpha$ vectors by their differences from $V$. When we need these vectors in later stages we compute them on-the-fly. After applying these two ideas, the total memory requirement for storing count vectors is $\frac{Gd}{g(2\alpha+1)} + \frac{2G\alpha(\alpha+1)}{2\alpha+1}$.

# Chapter 6

# Reduced Representation through Clustering

The sheer amount of data produced by current high-throughput DNA sequencing machines, over 100Gb/day for a Illumina HiSeq 2500 for example, demands enormous computational power for primary analysis tasks such as read mapping. Although a vast, varied body of work is concerned with read mapping [3, 87, 119, 120, 128–134], from computational point of view, readmappers can be roughly categorized into two groups. Fast mappers such as BWA which severely restrict the maximal edit distance between read and reference genome, and more sensitive tools such as TreQ and Stampy. Stampy is an order of magnitude slower than BWA but provides a much higher sensitivity in the presence of a high degree of variation. Although there is a clear incentive to use highly sensitive readmappers their use have been rather limited so far due to the enormous time requirement for mapping large HTS datasets.

As one of the major contributions to this thesis, here we propose a method for computing clusters from HTS reads and, instead of individually mapping one read at a time, map reads based on their cluster assignment. Our idea originates from one crucial observation about a computational aspect of the readmappers: to the best of our knowledge, most readmappers map one read at a time. In high coverage libraries reads originating from the same genomic locus share many bases among them, which, if processed together, can significantly reduce computational demand. The first step in exploiting the redundancy among reads originating from the same genomic locus is to identify them through clustering. To compute clusters, we use an incremental greedy approach ensuring stringent criteria for cluster membership in terms of overlap length and similarity to the anchor read representing the cluster (see Figure 6.1). Prior works on clustering HTS reads were concerned with detection of clusters in which cluster

members either completely or almost fully overlap with a high degree of similarity between them [135–137]. Relaxing this criteria makes clustering computationally feasible, and it is not detrimental to further analysis as the clusters are not per se focus of the investigation. Assuming that the clusters contain reads sequenced from the same genomic locus, we only map clusters, that is, we map one anchor read per cluster and align other cluster members using a local alignment algorithm. Mapping clusters, instead of individual reads, speeds up computation irrespective of the readmapper used.

This chapter is organized as follows. We start with a discussion on related work. In Section 6.2, we explain the clustering process in detail. In Section 6.3, we show how information about cluster membership can be exploited for mapping reads. We perform experiments on both simulated and biological datasets, and report results on the performance of clustering and the readmapping process in Section 6.4.

## 6.1 Related Work

Since clustering large datasets, as a pre-processing step, is a fundamental statistical technique it has been extensively used for many HTS related problems. For example, in read compression, some tools exploit redundancy among reads to achieve higher compression rate without using a reference genome; i.e., Coil [138] and ReCoil [139] use an approach similar to us for grouping similar reads together before compression, and a recent tool SCALCE [140] gains impressive compression rate by finding reads that share a large substring among them. Similar ideas have also been applied to error correction problem. Multiple sequence alignment (MSA) based error correction tools such as Coral [141] and ECHO [142] creates MSAs using highly similar overlaps among reads, which can be considered as clustering reads. Redundancy removal is another application based on clustering; i.e., SEED [135] shows that removing redundant reads through clustering can improve running time, reduce memory requirement and improve de-novo assembly quality. Similarly, for RAD-Seq data, Rainbow [143] shows improved de-novo assembly after clustering.

Although many HTS tools exploit sequence redundancy through clustering in various settings there are very few tools that use clustering as a pre-processing step before read mapping. One such tool, FreeClu [144], creates an interesting parent-child tree structure based on read frequency and hamming distance between reads. They report increased read alignment rate by utilizing the relationship represented as a tree. Unfortunately, their algorithm is not scalable to large datasets, and, in their definition, overlap between two reads must start from the same position and the hamming distance can be at most one, which severely restricts the full potential of a clustering approach. Another tool, Oculus [145], combines redundancy check and read mapping together as a package. In a streaming settings, they process one read at a time, align a previously unseen read using Bowtie and store results in a hash table. This allows them to skip expensive alignment step for redundant reads. Like FreClu, they need full overlap (same starting position) and, unlike FreClu, require complete match between reads to consider them redundant. This approach can achieve moderate speed-up for very high coverage data, for example RNAseq, but unlikely to be useful for low to moderate coverage whole genome sequencing data in presence of sequencing errors. In contrast, we process up to a billion read in one batch, allow reads to partially overlap and contain mismatches in the overlap (in spirit, similar to MSA based error correction approaches), which leads to significant speed-up in read mapping.

## 6.2 Clustering

By R= $\{r_1, \ldots, r_N\}$ we denote the read library. For all reads we assume a fixed length $L$. We also assume $R$ is produced by an Illumina-like sequencing experiment where substitution errors are dominant compared to insertion and deletion (indel) errors. In the following we discuss how we efficiently cluster $R$ and use these clusters to accelerate read mapping.

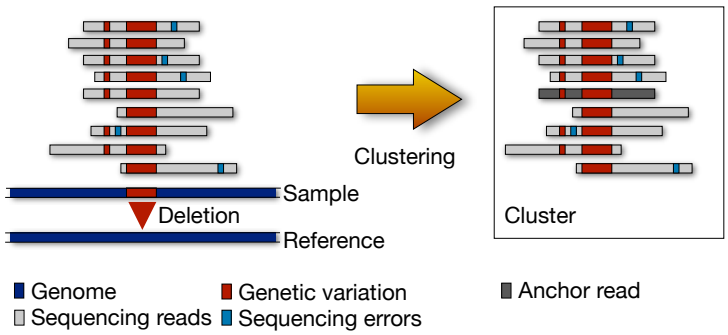



Figure 6.1: **Schematic view of clustering.** Genetic variations do not contribute to the edit distance between reads from the same genomic location. Thus, the only source of difference in an overlap is sequencing error. Since the sequencing error rate is small for Illumina-like datasets, in most cases, the clustering process can overcome the differences in the overlap between reads and group them together in a cluster. Moreover, to reduce the probability of assigning reads from different genomic locations to the same cluster, the overlap length has to be sufficiently large.

### 6.2.1 Clustering billions of reads for mapping

Our two main design goals for the clustering are computational efficiency and stringency with respect to only assigning reads from the same genomic locus to one cluster. Stringency and the size of the data imply large number of clusters, which precludes the use of iterative clustering methods. Note that the clusters themselves are not the focus of investigation, but rather a computational aid. Consequently, we allow reads from the same locus to be assigned to distinct clusters. We propose the following greedy approach based on shared $k$-mers for clustering reads.

#### Clustering single-end reads

Given two reads $r_i$ and $r_j$ with a prefix-suffix overlap of length $l$, where $l'$ number of bases match, we say that they have an overlap of length $l$ with similarity $s = \frac{l'}{l}$. If $l \geq \alpha L$ and $s \geq \beta$ ($\alpha$ and $\beta$ are constants), we assume $r_i$ and $r_j$ are *sufficiently similar* to be member of the same cluster. Clusters are represented by their anchor reads (in contrast to centroids in $k$-means clusters' anchors remain fixed) and reads, which are sufficiently similar to the anchors, are greedily assigned to the clusters (Figure 6.1). If a read of sufficient quality—i.e., not a *bad* read as defined in the following—fails to find a sufficiently similar anchor it becomes an anchor read itself and thus forms a new

cluster.

Specifically, we create a set of anchor reads AR and an array of $4^k$ lists AL = $\{\text{AL}_0, \ldots, A_m, \ldots, \text{AL}_{4^k-1}\}$, where $A_m$ stores pointers to the anchor reads that contain the $k$-mer $m$ (note that there is a one-to-one relation between $k$-mers and $2k$ bit numbers), and perform the following steps for each read $r_i$ in the library.

1. A base $r_{i,j}$ is defined as a bad base if it is ambiguous (represented by 'N') or it has Phred score below the threshold $f$. If there are too many bad bases uniformly distributed in $r_i$ we call $r_i$ a bad read and discard it from clustering. In particular, we discard $r_i$ if
   - $r_{i,s}$ and $r_{i,e}$ are the first and last non-bad bases respectively, and $e-s+1 < 2k$, or
   - the maximum length of a contiguous sub-sequence containing no bad base is less than $\frac{k}{2}$.
2. For each $k$-mer in $r_i$, we
   (a) compute the corresponding $2k$-bit integer representation $m$ and get the list of anchor reads that share the same $k$-mer from $\text{AL}_m$.
   (b) For each anchor read $r_j$ in $\text{AL}_m$, whose content is already stored in AR, if $r_i$ and $r_j$ has a prefix-suffix overlap of at least $l \geq \alpha L$ bases respecting the position of the shared $k$-mer, we compute similarity between $r_i$ and $r_j$ in the overlapped region.
   (c) If at least $\beta l$ number of bases match in the overlapped region, we declare $r_i$ as a member of the cluster formed around the anchor read $r_j$ and halt further computation for $r_i$.
   (d) Otherwise, we proceed with the next $k$-mer.
3. If $r_i$ fails to be assigned to any cluster, it qualifies to be an anchor read and forms it's own cluster. We store $r_i$ in the set of anchor reads AR and for every $k$-mer $m$ present in $r_i$'s $\alpha L$-length prefix and suffix we insert a record in $\text{AL}_m$.

### Clustering paired-end reads

Although clustering a paired-end read library in single-end mode is acceptable for many applications, for paired-end read mapping it is necessary to take pairing information into account. Unlike single-end clustering, where we consider one read at a time, in paired-end clustering, we process a pair of reads simultaneously and do not allow one end of a pair to be anchor read and the other end to be non-anchor read. This constraint lets us map anchor reads in paired-end mode using a traditional readmapper. If the constraint is violated for a pair $r_{i_1}$ and $r_{i_2}$, where, lets say, $r_{i_1}$ is an anchor read but $r_{i_2}$ is not, we resolve it in the following order.

1. If $r_{i_2}$ is a bad read, we force it to be an anchor read and form a new cluster.
2. If $r_{i_2}$ is a member of the cluster formed by anchor read $r_{j_2}$, whose other end is another anchor read $r_{j_1}$, we try to find a prefix-suffix overlap of length $l \geq \frac{\alpha L}{2}$ between $r_{i_1}$ and $r_{j_1}$ where $\beta l$ bases match (since $r_{i_2}$ already provides some evidence for this choice we relax the overlap length restriction from $\leq \alpha L$ to $\leq \frac{\alpha L}{2}$). If such an overlap is found we force $r_{i_1}$ to be a member of $r_{j_1}$.
3. Otherwise, $r_{i_2}$ is forced to be an anchor read and form a new cluster.

### Optimal cluster assignment

The above procedure partitions the read library into three sets; bad reads, anchor reads and member reads. Since not all anchor reads were known before the completion of the first phase, we defer final cluster assignment to the second phase of the algorithm. In particular, for each member read $r_i$ with an overlap of length $l$ with it's current cluster's anchor read, we try to change the assigned cluster by finding another anchor read with an overlap $l' > l$. We perform this by running only step 2 from the single-end algorithm and modifying 2.c to find the best possible cluster assignment.

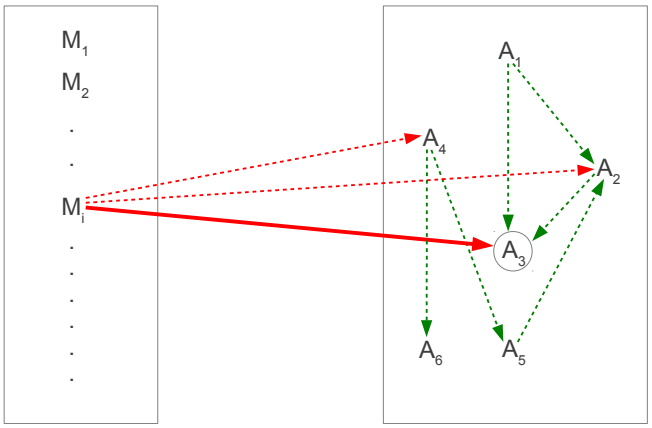



Figure 6.2: **Sub-optimal cluster choices.** Here $M_i$ is a member read of the cluster formed by anchor read $A_3$ (relationship shown with a solid red arrow). In addition to $A_3$, $M_i$ also has at least $\alpha L$ length overlap with $A_2$ and $A_4$, where at least ($\beta' \times$ overlap length) number of bases match (shown with dotted red arrow). Similarly, some of the anchor reads also have significant overlap among them (shown with dotted green arrow). These sub-optimal choices are considered in the read mapping step to increase the probability of accurately mapping non-unique, repetitive reads.

### Sub-optimal cluster assignment for repeat resolution

Another crucial aspect to take into consideration is the complexity of the underlying genome from which sequencing reads originate. Because of the non-uniformity and repetitiveness present in the genome, sub-optimal cluster assignments are sometimes preferred. Towards this goal, in the second phase, along with the cluster with highest overlap, we also store at most $S_M$ number of sub-optimal (allowing $\beta' < \beta$) cluster assignments for each member read (red dotted arrows in Figure 6.2). We take this idea further and, for an anchor read, store at most $S_A$ number of similar anchor reads by modifying the first phase of the algorithm (green dotted arrows in Figure 6.2). Storing these two kinds of sub-optimal choices can be very useful in resolving ambiguity and repeats in read mapping.

### Practical Considerations

Even our greedy two stage clustering strategy can become infeasible already for one billion reads. We have carefully taken some decisions and made some parameter choices to keep the clustering process feasible. In particular, smaller values of $k$ increase sensitivity

but require more computational resources due to more frequent $k$-mer hits. Since our cluster definition is comparatively relaxed, a large $k$, in this case 15, is suitable for read libraries with few sequencing errors such as Illumina's. Additionally, we have made the following choices: One, we perform a Hamming distance computation on the overlap, thus ignoring indels introduced by sequencing, between two reads only if they share at least two $k$-mers, which decreases the probability of spurious hits due to the choice of $k$ and the presence of sequencing errors; two, since overlapping $k$-mers in a read are not independent and, hence, considering all $k$-mers increases computational demand without significant improvement in optimal cluster assignment, in the second phase, we abandon the search for best cluster assignment after considering $\frac{L-k+1}{4}$ equally spaced $k$-mers from the read; three, since frequent $k$-mers are less informative the size of a list $\mathrm{AL}_m$, storing the anchor reads containing $m$, is restricted to 256; four, we restrict the number of sub-optimal choices $S_M$ to 16 and $S_A$ to 256 and store them in external memory; finally, we set $\beta = 0.95$, $\beta' = 0.8$ and $\alpha = max\{0.5, \frac{31}{L}\}$, which restricts the allowed read length to be at least 31 and guarantees that the clustering algorithm can overcome at least one error $(2 * 15 + 1 = 31)$ in the overlap.

### 6.2.2 Running time and memory usage

For each of the $N$ reads of length $L$ in the library, the clustering process computes Hamming distance between the read and potential anchor reads found using shared $k$-mers. Consequently, the running time complexity of the algorithm is $O(NL^2)$. Due to the practical choices from the previous section, in practice, the total running time is not quadratic in $L$, and the average running time is approximately $O(NL)$. For space complexity, lets assume the clustering algorithm finds $\tau N$ clusters, where $0 \leq \tau \leq 1$. In that case, the set of 2-bit encoded anchor reads AR needs $O(\tau NL)$ bytes, the array of lists AL needs $O(\tau NL + N)$ bytes and cluster membership information takes $O(N)$ bytes. Thus, the total amount of space required by these three objects is $O(\tau NL)$. Including other overhead, specially for multi-threading support, a 100-bp 1.5 billion human read dataset, where $\tau = 0.08$ ($\tau = 0.11$) for single-end (paired-end), requires 76GB (91GB) memory. If the read sequences were error free, we would need

significantly less amount of memory since $\tau$ would be $O(\frac{G}{L})$. But, unfortunately, HTS libraries contain frequent sequencing errors, and hence, this large space requirement is unavoidable for de-novo clustering.

### 6.2.3 Additional points

#### Coverage and read length

Statistically significant findings in the downstream analysis require high coverage sequencing. For example, Illumina's cancer pipeline recommends 40x coverage for normal and 80x coverage for cancer tissues. The higher the coverage of the library the more advantageous the clustering becomes, as we expect the average cluster size to increase. We tested read libraries with reads of 36–100bp and we expect results to improve for even longer reads. For significantly shorter reads read mapping based on clusters will likely fail to achieve any significant speed-up or the loss of sensitivity will be non-negligible.

#### Bias of 15-mer

An overlap of length at least $15 * (E + 1) + E$ is guaranteed to overcome $E$ errors in our algorithm. Since we use a fixed value for $\alpha$, some reads will not be detected as members and possibly create singleton clusters. Smaller $k$ will lessen this problem, but to keep the algorithm efficient for large datasets we do not want to lower $k$. Instead, if a read library is suspected to contain many reads with large number of errors $\alpha$ can be increased to deal with the issue.

#### Parameter choices

If the sequencing error rate $\epsilon$ (typically, less than 0.02 for Illumina experiments) can be determined a priori, we suggest using a similarity cutoff $\beta \leq (1 - 2\epsilon)$. Too large a value for $\beta$ makes it difficult to assign a read to clusters, thus creating too many clusters, while a value too low might assign reads to wrong clusters.

For $\alpha$, if we choose a smaller value, we will get fewer clusters, which will in turn decrease running time, but unfortunately read mapping sensitivity will drop. On the

other hand, a value too large leads to more clusters. This increases running time of both clustering and mapping, but it also increases mapping sensitivity. The values we used, $\alpha = \max\{0.5, \frac{31}{L}\}$ and $\beta = 0.95$, worked well for three genomes of very different sizes and we expect that the choice will work well for experiments with similar coverage, read length and genome size and structure. Should the parameters of the experiment change drastically the choices might have to be re-evaluated in a preliminary study.

**Automatic choice of parameters**

It is possible to design a scheme to automatically select the necessary parameters $\alpha$ and $\beta$. A small, randomly selected set of reads can be mapped to the reference genome to compute the error rate $\epsilon$ from the uniquely mapped reads. Given $\epsilon$, we can perform a grid search over the possible values of $\alpha$ and $\beta$. A chosen set of parameters should fulfill two criteria: One, a valid overlap, defined by $\alpha$ and $\beta$, between two reads from different genomic locus should only occur with very low probability, and two, the amount of required memory, which is primarily determined by the number of clusters, should be less than the available system memory. Next, we show an analysis of the expected number of clusters produced by a set of parameters.

**Number of clusters:**

We will assume that the reads are numbered in the order they are processed. Let $r'_i \neq r'_j$ means that read $r_i$ has an overlap with read $r_j$ of length at least $\alpha L$ and their overlapped portion $r'_i$ and $r'_j$ has similarity less than $\beta$. Let the number of sequencing errors in $r'_i$ be $s(r'_i)$. Assuming $s(r'_i)$ is binomially distributed with rate $\epsilon$, $Pr[s(r'_i) = k] = \binom{|r'_i|}{k}\epsilon^k(1-\epsilon)^{|r'_i|-k}$. Let $C_i \subseteq \{r_1, r_2, \cdots, r_{i-1}\}$ be the set of anchor reads with at least $\alpha L$ overlap with $r_i$. Given $C_i$, the probabity of read $r_i$ becoming an anchor read and forming it's own cluster is $P_i = Pr\big[\bigcap_{r_j \in C_i} r'_i \neq r'_j\big]$. We simplify the analysis by assuming that all the reads in $C_i$ has overlap of length $l' = \frac{(1+\alpha)}{2}L$, which is the expected length of a valid overlap. The updated probability is $P^*_i \approx P_i$. Since the

number of mismatches allowed is at most $m = (1-\beta)l'$, $P_i^*$ is upper bounded by

$$\sum_{x=0}^{l'} \left( Pr[s(r'_i) = x] \prod_{r_j \in C_i} Pr[s(r'_j) \geq m - x] \right)$$
$$= \sum_{x=0}^{l'} Pr[s(r'_i) = x] Pr[s(r'_j) \geq m - x]^{|C_i|}$$
$$= \sum_{x=0}^{m-1} Pr[s(r'_i) = x] Pr[s(r'_j) \geq m - x]^{|C_i|} + Pr[s(r'_i) \geq m].$$

Let $T_{i-1}$ be the number of clusters formed after processing $i-1$ reads. Assuming the reads were sequenced uniform randomly from the genome, $E[|C_i|] = \sum_{j=0}^{i-1} \frac{2(1-\alpha)L}{G} P_j = \frac{2(1-\alpha)L}{G} T_{i-1}$. Now, if we replace $|C_i|$ with $E[|C_i|]$, $P_i^*$ is approximately equal to

$$\approx \sum_{x=0}^{m-1} Pr[s(r'_i) = x] Pr[s(r'_j) \geq m - x]^{\frac{2(1-\alpha)L}{G} T_{i-1}} + Pr[s(r'_i) \geq m]$$
$$= \sum_{x=0}^{m-1} \binom{l'}{x} \epsilon^x (1-\epsilon)^{l'-x} \left( \sum_{y=m-x}^{l'} \binom{l'}{y} \epsilon^y (1-\epsilon)^{l'-y} \right)^{\frac{2(1-\alpha)L}{G} T_{i-1}} + \sum_{y=m}^{l'} \binom{l'}{y} \epsilon^y (1-\epsilon)^{l'-y}.$$

We numerically evaluate the approximate upper bound for the number of clusters $T_N \approx T_{N-1} + P_N^*$. See Figure 6.3 for an application of this analysis.

## 6.3 Read Mapping

In resequencing experiments the first step in the analysis typically is mapping reads to a reference genome. The choice of a tool depends on the biological question under investigation, the expected genetic variation between sample and reference genome and the computational resources available. The tools roughly fall into two categories: fast read-mappers which trade losses in mapping reads for speed such as SOAP2 [120], Bowtie 2 [146] or BWA [87] or highly-sensitive, but much slower tools such as mrFAST [121], RazerS [85], LAST [123], Novoalign (`http://www.novocraft.com`) or Stampy [131]. We will show how we accelerate read mapping based on the clustering irrespective of the readmapper used.

Our clustering algorithm works under the assumption that members of a cluster

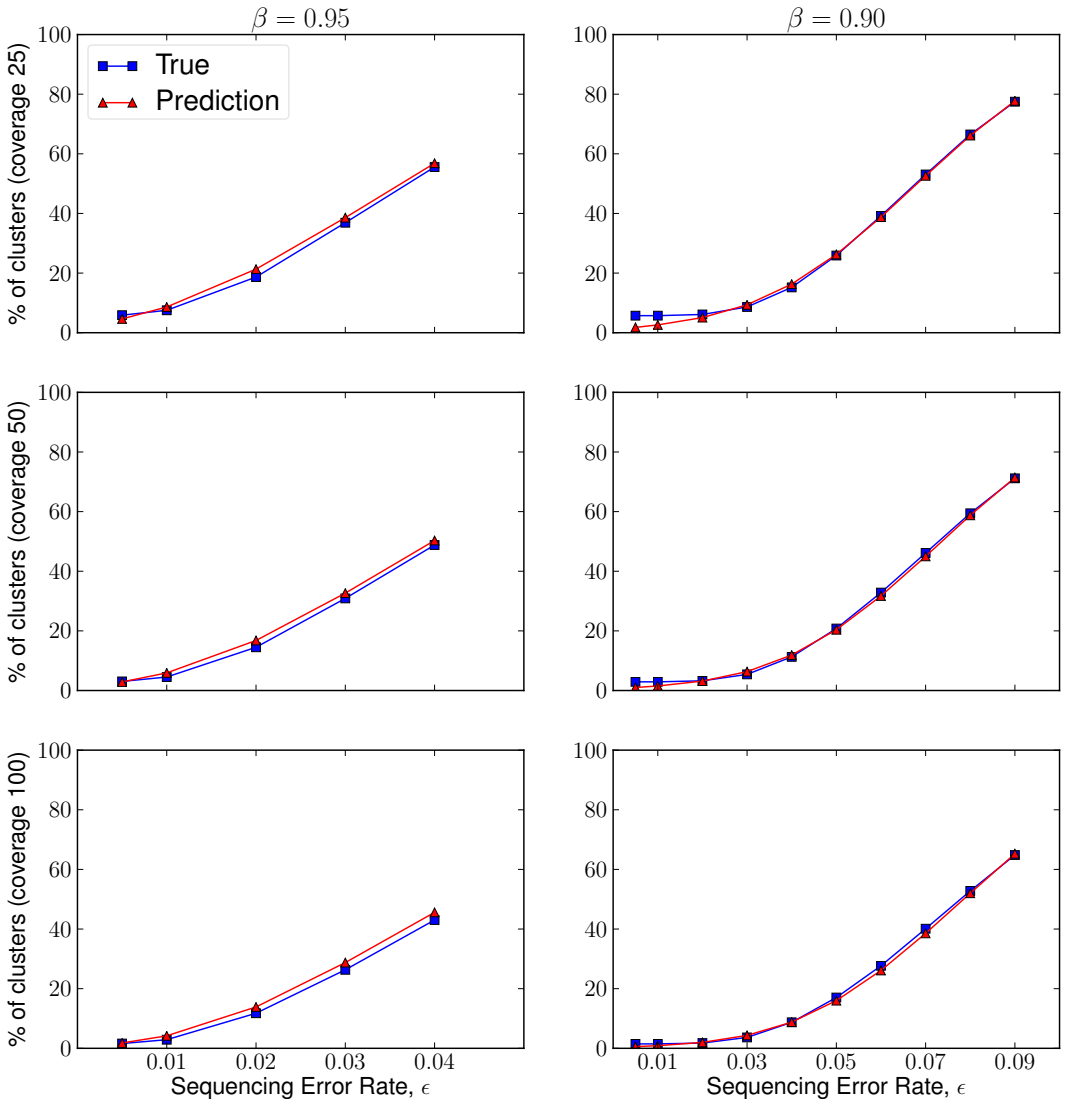



Figure 6.3: **Expected number of clusters.** From an artificially generated sample genome, an i.i.d. sequence of 1 million bases, we sequence 100bp single-end reads with coverage 25, 50, and 100, and insert sequencing error at a rate ranging from 0.005 to 0.09. We compare the true number of clusters computed using the clustering algorithm to the numerically evaluated ones for $\beta = 0.95$ and $\beta = 0.9$. The total number of cluster is computed very accurately except for small values of $\epsilon$. It is evident that $\beta \leq (1 - 2\epsilon)$ keeps the total number of clusters small.

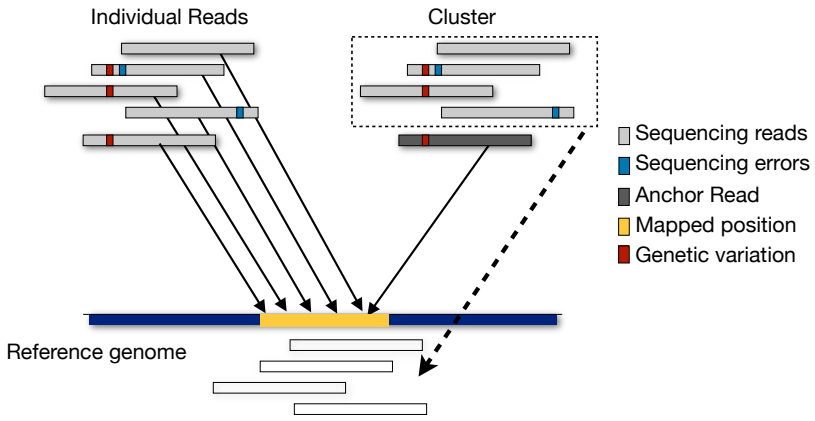



Figure 6.4: **Schematic view of clustered read mapping.** Read mappers map all reads in input individually to the reference genome (ignoring savings achieved using index structures). In contrast, in clustered read mapping only the anchor read is mapped to the reference genome. The mapping positions of individual cluster members are recovered w.r.t. to the position of the anchor read.

originate from the same genomic locus in the sample genome. We exploit this assumption by mapping the anchors in single-end or paired-end mode to a reference genome and subsequently use the information about the mapped position of the anchor read to find good mappings for member reads. As a result the runtime complexity of read mapping reduces to the proportion of clusters found; in practice we need more time as the cost of mapping a single read varies considerably. Although we can find more than one location for non-unique, repetitive reads using multiple hits for the anchor reads and utilizing sub-optimal cluster choices, in this study, for a suitable comparison between clustered vs. individual read mapping, we only report single best hits for reads. Specifically, given the read library $R$ we perform the following steps to compute the read alignments assuming fixed costs 3, $-3$, $-40$, and $-3$ for nucleotide match, mismatch, gap open, and gap extension.

1. Identify the anchor reads and map them in single best mode using a readmapper. Report the alignments in a SAM file $S$.
2. Extract mapping information of the anchor reads from $S$.
3. For each read $r$ in $R$,
   (a) if $r$ is an anchor read, do nothing.

(b) If $r$ is a bad read, report the read as unmapped in $S$.

(c) If $r$ is a member read,

- we use the overlap information between anchor read and member read to identify the genomic position from where the member read may have originated, and compute hamming distance.
- If the total cost of mismatches from previous step is greater than a single gap opening cost (for the tested datasets it happens for approximately 30% of the reads) we run the Smith-Waterman algorithm [147] to find the best local alignment between the read and the mapped position of the anchor in the genome ($\pm L$ bases). We use a modified version of a SSE based library form [148] for Smith-Waterman computation.
- We repeat the above two steps for the sub-optimal cluster choices (Figure 6.2). In particular, we increase the search space by following a path, through the dotted red and green links, of length at most two, from the member read to the anchor reads.
- If the best alignment has score less than `min_score` (default $\frac{L}{3}$) we report the read as unmapped. Otherwise, we append the alignment information at the end of file $S$.

For paired-end reads, we modify the above process in the following way. As in paired-end clustering anchor reads are stored in pairs, we use a traditional readmapper in paired-end mode. After extracting the mapping information of anchor reads from the SAM file, we align member reads one pair at a time. If one or both reads in a pair are bad reads, we do not do anything different from the above process. For the remaining member read pairs, we do the following.

- For each end of a pair, we compute a list of alignment positions in the genome where alignment score is $\geq$ `min_score`. We use optimal and sub-optimal cluster information to generate the list following the single-end process.
- From the two lists of potential positions, if there is a concordant location for the pair we report that location.

- Otherwise, for each potential position of one end in the list, we compute alignment score between the other end and the potential genomic location where the other end could be present (based on insert length and standard deviation). If the alignment score is above `min_score` we report this location.

- If no concordant location is found using the above steps, we report the best location for each end separately.

Since readmappers often use different kind of alignments — choice varies among local, global, and semi-global alignment, score values, heuristics, and strategies to improve paired-end mapping, it is difficult to select an universal set of criteria which is good for all. The differences in choices and heuristics used give them distinguishing characteristics which is often desirable by downstream analysis tools. We believe that the choices we have made here are not necessarily the best for every readmapper, and it makes sense to have different clustered mapper for different readmappers. However, we do not pursue that direction here.

## 6.4 Experiments

We have selected four datasets to analyze our method.

- `SIM`: 2 million simulated 100bp paired-end reads (mean insert length 300 and standard deviation 30) simulated with a coverage of 50x obtained with the popular read simulator ART [149] from human chromosome 17 (from the 10Mbp to 14Mbp region) after artificially introducing SNPs (`SIM-SNP`) and inserting indels (`SIM-INDEL`).
- `ECOL`: 21 million 36bp paired-end E. coli reads (SRX000429) with a coverage of 160x at a genome size of 5Mb. We estimated the insert length to be 215bp with a standard deviation of 10bp.
- `DROS`: 48 million 76bp paired-end Drosophila reads (SRR097732) [150] with a coverage of 29x at a genome size of 120Mb. We estimated the insert length to be 320bp with a standard deviation of 18bp.

- `YOR`: 1.46 billion 100bp paired-end Illumina reads of a Yoruba individual (ERA015743) (HapMap: NA18507) with a coverage of 46x at a genome size of 3.2Gb. We estimated the insert length to be 310bp with a standard deviation of 20bp.

For all datasets we use $\alpha = \max\{0.5, \frac{31}{L}\}$ and $\beta = 0.95$. All the experiments are performed on a Linux machine with 48 AMD Opteron cores clocked at 2.2 GHz and 256GB memory. Whenever possible, in particular for multi-threaded tools, 22-24 cores were assigned, and for both multithreaded and non-multi-threaded programs total system time is reported.

In the following, we will show the effectiveness of our clustering algorithm and the benefits of using clusters in read mapping. We will also discuss how the characteristics of the read library affect performance and parameter choices.

### 6.4.1 Clustering performance

Since clustering is a fundamental technique for large data analysis, many tools have been developed in the past for biological data; i.e. for protein clustering [151–153], metagenomics [154–157], and expressed sequence tags (ESTs) [158–161]. Recently, there is a trend of developing alignment based (for alignment-free clustering see [162]) fast clustering tools for very large HTS data sets; among these, CD-HIT [163], DNACLUST [164], UCLUST [137] and SEED [135] are prominent. These four tools use greedy incremental approaches and apply heuristics to accelerate clustering (our clustering algorithm also belongs to this group). Except DNACLUST, which uses a suffix array, these tools also use some kind of hash/seed based data structure. Among them, SEED [135] has been identified as the state-of-the-art in read clustering. Although it can cluster up to tens of millions of reads given enough time and memory, SEED, and other tools, are still not very attractive as a pre-processing step before read mapping. There are two reasons behind this: First, they require highly overlapping reads, usually $\alpha > 0.90$, and, in some cases, high similarity in the overlap, which results in a large number of clusters; Second, their algorithms and data structures are not designed to process billions of reads. While large overlap is absolutely necessary for very short

reads (say $< 50$bp), for moderate to large sized reads (say $> 50$bp), $\alpha = 0.5$ is sufficient for declaring two reads similar for read mapping purpose since per base error rate for Illumina reads are small. To overcome these difficulties, and to support particular requirements of read mapping, such as paired-end support and sub-optimal cluster choices, we have designed our own clustering algorithm TreQ-CG. As a representative of the fast clustering tools (see [165] for a set of examples) we have selected UCLUST and SEED, and compared their performance with TreQ-CG (Table 6.1). We indirectly measure the quality of the produced clusters by using them for read mapping in a later step. Among the two tools in consideration, only SEED allows non-overlapping bases (at most 6 bases can be excluded), and it allows at most 3 mismatches in an overlap which corresponds to parameters $(\alpha = 0.83, \beta = 0.90)$, $(\alpha = 0.92, \beta = 0.96)$ and $(\alpha = 0.94, \beta = 0.97)$ for `ECOL`, `DROS` and `YOR` respectively. On the other hand, UCLUST allows a flexible $\beta$ but it does not allow users to select $\alpha$. Compared to them, TreQ-CG allows both parameters to be modified. Taking these differences into consideration we have selected three set of parameters for comparison: $C_1$ allows UCLUST to run with smallest possible $\beta$ allowed by SEED, $C_2$ allows smallest possible $\alpha$ and $\beta$ allowed by SEED, and $C_3$ is the parameter set used by TreQ-CG for subsequent analysis.

Along with memory requirement and running time, our primary criteria for evaluating clustering tools is the number of clusters produced, which is normalized as $\%\text{ of clusters} = \frac{\#\text{ of anchor reads}}{\#\text{ of reads}}$. Since SEED depends on large overlap between reads, it produces 37.79% clusters for `DROS` in 7.56 hours and does not complete on the `YOR` dataset. On the other hand, UCLUST's use of full overlap and gapped alignment makes it very slow on large datasets. It takes 52 hours on the `ECOL` dataset but on the larger datasets it did not complete in one week. In comparison, TreQ-CG works with a wide range of parameter choices in both single-end and paired-end mode, produces fewer clusters, and runs faster than SEED. Additionally, for parameter set $C_3$, TreQ-CG discards 0.3%, 1.6%, and 5.2% reads as bad reads in single-end mode, and 0.2%, 1.3%, and 4.8% reads in paired-end mode. From the remaining reads, ones with large number of sequencing errors usually form clusters with very few member reads, which is evident in the singleton clusters generated from the datasets. The clusters produced by

SEED and other clustering tools on smaller datasets such as ECOL are useful for other applications, for example in meta-genomics. But unless they support smaller overlap length, which is difficult for their particular choice of data structures and algorithms, they will not be attractive choices for pre-processing reads before read mapping. We do not consider them for subsequent analysis in the following section.

### 6.4.2 Read mapping performance

On biological datasets there is no ground truth available for deciding whether a read is mapped correctly. There are also differences—in particular for low-quality reads and in presence of genetic variations—in mapping performance between individual readmappers, and likewise we expect some differences between clustered mapping and individual mapping even with the same readmapper.

We compare clustered and individual mapping for Bowtie 2 [146] version 2.1.10, BWA [87] version 0.5.9, Novoalign (http://www.novocraft.com/) version 2.07.13 and Stampy [131] version 1.0.19. All mappers, chosen because they are widely used and implement different algorithmic approaches, were run with default parameters except forcing reporting of a single best hit. We expect other readmappers, specially highly sensitive computationally intensive ones, to benefit from clustered approach and their overall performance to follow the performance of the readmappers chosen for comparison.

Including clustering time, clustered read mapping achieves a speed-up compared to the time of mapping individual reads for all readmappers tested, see Table 6.2 for single-end reads and supplementary Table 6.3 for paired-end reads. The speed-up ranges from 8.9 using Stampy on YOR to 1.4 using Novoalign on ECOL. One would expect that the time needed for mapping anchor reads should be proportional to the number of anchors. But in practice, reads with high sequencing errors form singleton clusters (that is a cluster without any member except the anchor read) and read mappers take more time to map these reads. As a result the speed-up is not exactly proportional to the number of clusters. Still, due to the achieved speed-up, the use of sensitive readmappers like Stampy becomes computationally as feasible as running BWA.

Table 6.1: **Comparison of clustering tools.** The performance of three clustering tools, UCLUST, SEED, and TreQ-CG, on three biological datasets `ECOL`, `DROS`, and `YOR` is shown. Three different parameter configurations are used for comparison. SEED was run with parameters `--shift X --mismatch Y --fast --reverse`, where `X` and `Y` corresponds to $\alpha$ and $\beta$ respectively in TreQ-CG. Additionally, $C_1, C_2$ and $C_3$ are respectively $(1.00, 0.90), (0.83, 0.90), (0.86, 0.95)$, for `ECOL`, $(1.00, 0.96), (0.92, 0.96), (0.50, 0.95)$, for `DROS` and $(1.00, 0.97), (0.94, 0.97), (0.50, 0.95)$ for `YOR`.

| Dataset | Param. ($\alpha$, $\beta$) | Program | Single-end | | | Paired-end | | |
|---|---|---|---|---|---|---|---|---|
| | | | Time (h:mm) | Memory (GB) | Clusters (%) | Time (h:mm) | Memory (GB) | Clusters (%) |
| `ECOL` `21 mil.` `36 bp` | $C_1$ | UCLUST | 52:07 | **2** | 17.47 | — | — | — |
| | | SEED | 0:32 | 12 | 38.97 | — | — | — |
| | | TreQ-CG | 0:24 | 17 | 23.86 | 0:30 | 17 | 31.21 |
| | $C_2$ | UCLUST | — | — | — | — | — | — |
| | | SEED | 0:27 | 11 | 5.71 | — | — | — |
| | | TreQ-CG | **0:12** | 16 | **4.07** | **0:13** | **16** | **5.14** |
| | $C_3$ | UCLUST | — | — | — | — | — | — |
| | | SEED | 0:30 | 11 | 7:30 | — | — | — |
| | | TreQ-CG | 0:14 | 16 | 7.30 | 0:15 | 16 | 8.57 |
| `DROS` `48 mil.` `76 bp` | $C_1$ | UCLUST | — | — | — | — | — | — |
| | | SEED | 3:22 | 25 | 75.97 | — | — | — |
| | | TreQ-CG | 3:36 | 34 | 70.34 | 4:02 | 35 | 76.73 |
| | $C_2$ | UCLUST | — | — | — | — | — | — |
| | | SEED | 7:56 | 25 | 37.79 | — | — | — |
| | | TreQ-CG | 2:00 | 23 | 24.50 | 2:30 | 24 | 30.09 |
| | $C_3$ | UCLUST | — | — | — | — | — | — |
| | | SEED | — | — | — | — | — | — |
| | | TreQ-CG | **1:09** | **20** | **7.29** | **1:18** | **20** | **9.18** |
| `YOR` `1.5 bil.` `100 bp` | $C_1$ | UCLUST | — | — | — | — | — | — |
| | | SEED | — | — | — | — | — | — |
| | | TreQ-CG | — | — | — | — | — | — |
| | $C_2$ | UCLUST | — | — | — | — | — | — |
| | | SEED | — | — | — | — | — | — |
| | | TreQ-CG | 197:35 | 196 | 29.45 | — | — | — |
| | $C_3$ | UCLUST | — | — | — | — | — | — |
| | | SEED | — | — | — | — | — | — |
| | | TreQ-CG | **107:52** | **76** | **7.48** | **133:22** | **91** | **10.90** |

For qualitative assessment of clustered read mapping we use the following three metrics.

- **Accuracy.** Since the true originating position of a read is known for simulated data, we define accuracy as the proportion of reads that map within $\pm L$ bases of the true position [132]. We do not require a strict match because there can be ambiguity in the starting position of the optimal alignment due to the presence of gaps and exact alignment parameters.
- **Alternate mapping rate.** Since the ground truth is not known for biological data, we compute alternate mapping rate between individual mapping and clustered mapping. Alternate mapping rate is defined as the percentage of reads for which either both approaches report mapping positions not within $\pm L$ bases of each other, or one approach maps the read while the other fails. This essentially measures how well the clustered read mapping recreates the *exact* output of the individual read mapping.
- **Concordance.** For biological data, another important measure is concordance of read pairs. We define it as the proportion of read pairs for which the estimated insert size is within the mean insert length of the library, allowing for $\pm 5$ standard deviation. It has been argued in the literature [131] that the concordance of paired-end reads mapped in single-end mode provides an indirect measure of mapping accuracy.

### Simulated Data

We have tested the effect of SNPs and indels on single-end and paired-end clustered mapping; the results are reported in Figure 6.5. For `SIM-SNP` we have modified the sample genome with substitution rate ranging from 0.001 to 0.15 while allowing no indels. For `SIM-Indel` we have inserted indels of a particular size $I \in [1, 20]$ with an indel rate of 0.01 per base. For each SNP and indel size configuration we have sequenced 2 million reads using the read simulator ART, which applies Illumina-specific sequencing errors on top of the genetic variations present in the artificial sample genome. Since

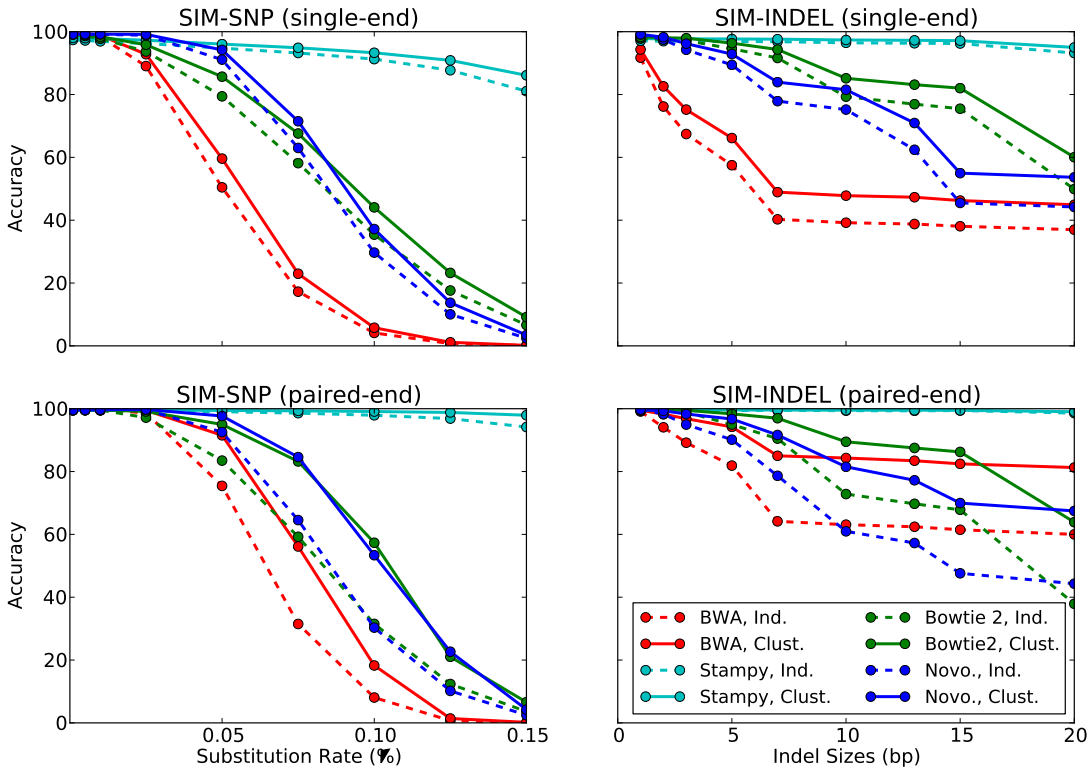



Figure 6.5: **Quality improvements in detecting SNPs and structural variants.** Accuracy of mapping single-end simulated reads between individual and clustered approach is shown. As expected, BWA and Bowtie 2 do not perform very well in presence of large variations. Clustered read mapping performs significantly better than individual BWA and Bowtie 2, and generally agrees with the individual version of the sensitive readmappers.

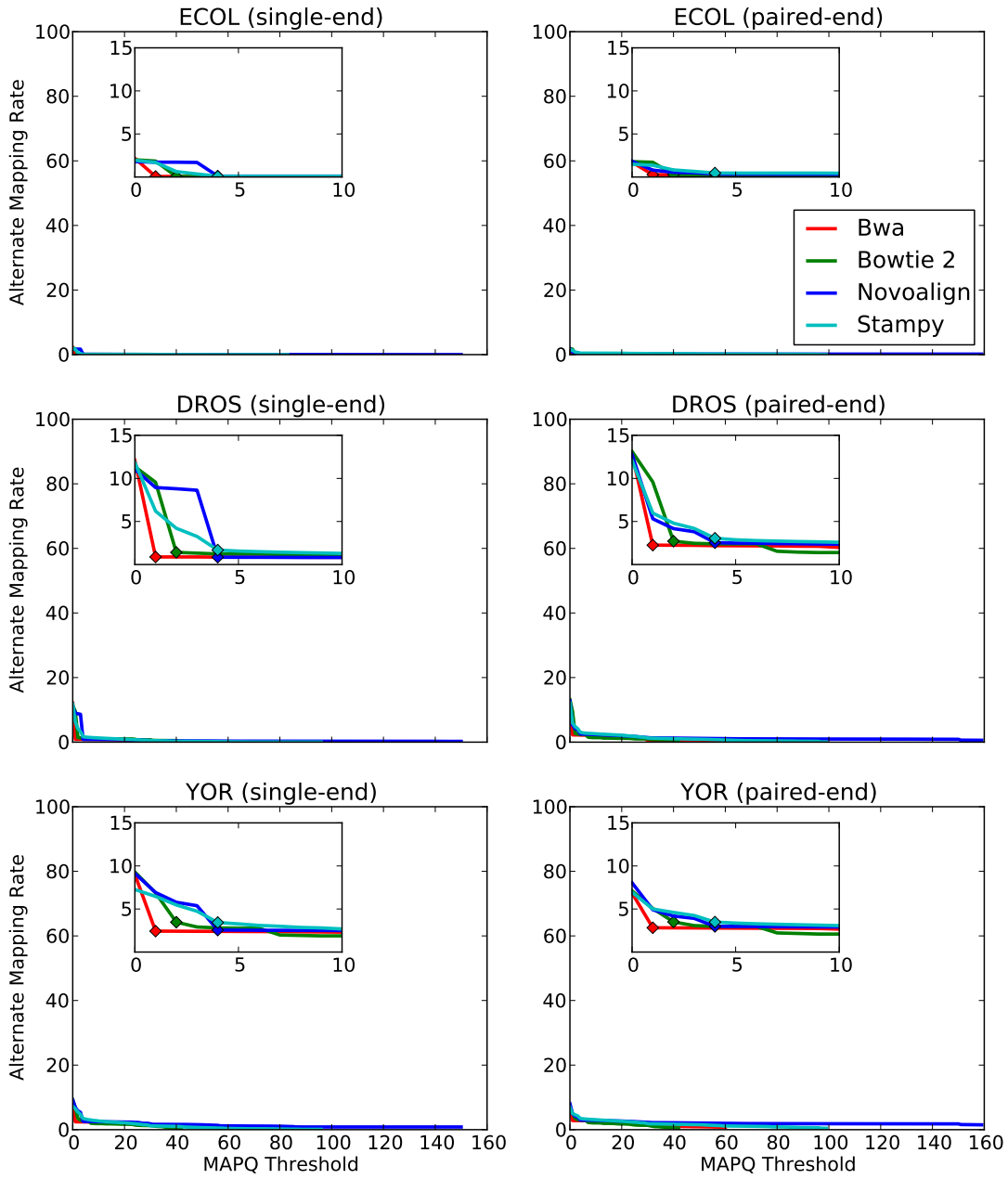



Figure 6.6: **Alternate mapping rate between clustered and individual single-end read mapping.** Alternate mapping rate is shown as a function of MAPQ threshold. Maximum reported MAPQ value varies between readmappers; for BWA, Bowtie 2, Novoalign and Stampy maximum reported MAPQ for single-end reads are respectively 37, 42, 150 and 96. The cutoffs used to report alternate mapping rate in Table 6.2 are shown with diamond signs in the inset figures.

indels are randomly placed in the sample genome before sequencing, approximately 40% reads in the `SIM-Indel` dataset do not contain any indel. As expected, Stampy's performance stays excellent for the range of variations tested but others do not perform well in presence of high genetic variation. In case of Stampy, clustered mapping closely follows individual mapping. On the other hand, clustered mapping performs better than individual BWA, Bowtie 2, and Novoalign. To gain efficiency these readmappers limit their mapping process through limited edit distance or score cutoff. In contrast, clustered mapping is more permissive due to low score cutoff (default $\frac{L}{3}$).

**Biological Data**

It is reasonable to expect that the reads mapped with low alignment quality are those which should cause the most differences between clustered and individual mapping, and consequently influence the accuracy measures. We use the MAPQ value reported in the SAM file, where higher value indicates higher confidence in a particular alignment, as thresholds to limit the read alignments considered to compute alternate mapping rate. In practice, along with unmapped reads (reported with default MAPQ value 0), low quality alignments are frequently excluded from downstream analysis. Following [87], we select MAPQ value of 1 and 2 for BWA and Bowtie 2 respectively, and 4 for Novoalign and Stampy, as a cutoff to report alternate mapping rate in Table 6.2 and Table 6.3. Alternate mapping rates computed using different values of MAPQ threshold are reported in Fig. 6.6. We also report concordance of reads mapped in Table 6.2 and Table 6.3.

From Figure 6.6 we observe that the alternate mapping rate among all reads above a MAPQ threshold strictly decreases as a function of the threshold and eventually drops below 1%. At the selected MAPQ thresholds, depending on the readmapper used, clustered mapping reports alternate mapping for about 0–1%, 1–3%, and 2–4% reads for `ECOL`, `DROS` and `YOR` datasets respectively. Compared to single-end reads, there is an 1-2% increase in alternate mapping rate for paired-end mode (see Table 6.3). As for concordance of single-end reads in clustered approach, there is a small gain in concordance for `ECOL` and `DROS`, and a small loss in concordance for `YOR` (Table 6.2).

Table 6.2: Running time and memory requirement of individual and clustered read mapping on single-end datasets are reported. As expected, clustered read mapping achieves significant speed-up for all datasets. Although running Stampy on YOR dataset in individual mapping mode is prohibitively expensive, clustered approach makes running Stampy as feasible as running BWA.

| Dataset (single-end) | Mapper | Memory (GB) | | | Time (hh:mm) | | | Speed-up | Mapping quality | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Individual mapping | TreQ-CG | | Individual mapping | TreQ-CG | | | Alternate map. rate | Concordance | |
| | | | Clust. | Map. | | Clust. | Map. | | | Ind. | Clust. |
| ECOL | BWA | **0.1** | 16 | 0.9 | 0:49 | 0:14 | **0:12** | 1.9 | 0.06 | 92.93 | **93.44** |
| | Bowtie 2 | **0.3** | | 0.9 | 0:43 | | **0:13** | 1.6 | 0.06 | 93.41 | **93.64** |
| | Novoalign | **0.0** | | 1.0 | 0:38 | | **0:14** | 1.4 | 0.10 | 96.59 | **96.80** |
| | Stampy | **0.0** | | 1.0 | 4:17 | | **0:28** | 6.1 | 0.08 | **96.23** | 96.19 |
| DROS | BWA | **0.3** | 20 | 1.4 | 3:29 | 1:09 | **0:49** | 1.8 | 0.88 | 71.02 | **72.07** |
| | Bowtie 2 | **0.4** | | 1.4 | 2:53 | | **0:39** | 1.6 | 1.41 | **74.82** | 73.87 |
| | Novoalign | **0.6** | | 1.4 | 6:19 | | **1:09** | 2.8 | 0.85 | 71.20 | **73.87** |
| | Stampy | **0.3** | | 1.4 | 28:08 | | **2:59** | 6.8 | 1.68 | 74.06 | **74.31** |
| YOR | BWA | **5.0** | 76 | 8.3 | 477:30 | 107:52 | **157:29** | 1.8 | 2.43 | **78.43** | 77.54 |
| | Bowtie 2 | **3.5** | | 8.2 | 307:45 | | **75:04** | 1.7 | 3.47 | **82.38** | 79.75 |
| | Novoalign | **7.8** | | 8.3 | 714:07 | | **230:47** | 2.1 | 2.55 | **79.40** | 78.24 |
| | Stampy | **2.7** | | 8.2 | 3203:39 | | **251:40** | 8.9 | 3.43 | **80.51** | 78.87 |

On the other hand, for paired-end reads, except for ECOL dataset, there is always a gain in concordance, which is mostly due to altering mapping locations based on pairing information (see Table 6.2). Increase in alternate mapping rate and gain in concordance for paired-end reads indicates that our paired-end mapping might be too permissive compared to what is allowed by an individual readmapper.

Table 6.3: Running time and memory requirement of paired-end clustered read mapping.

| Dataset (paired-end) | Mapper | Memory (GB) | | | Time (hh:mm) | | | Speed-up | Mapping quality | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Individual mapping | TreQ-CG | | Individual mapping | TreQ-CG | | | Alternate map. rate | Concordance | |
| | | | Clust. | Map. | | Clust. | Map. | | | Ind. | Clust. |
| ECOL | BWA | **0.2** | 16 | 0.9 | 1:01 | 0:15 | **0:13** | 2.2 | 0.27 | **98.77** | 98.62 |
| | Bowtie 2 | **0.3** | | 0.9 | 0:49 | | **0:12** | 1.8 | 0.28 | **98.39** | 98.36 |
| | Novoalign | **0.0** | | 1.0 | 0:57 | | **0:15** | 1.9 | 0.33 | **99.57** | 99.28 |
| | Stampy | **0.0** | | 1.0 | 4:08 | | **0:31** | 5.4 | 0.33 | **99.41** | 99.18 |
| DROS | BWA | **0.4** | 20 | 1.4 | 3:57 | 1:18 | **1:09** | 1.6 | 2.25 | 84.17 | **85.62** |
| | Bowtie 2 | **0.5** | | 1.4 | 5:33 | | **1:05** | 2.3 | 2.71 | 84.56 | **85.51** |
| | Novoalign | **0.6** | | 1.4 | 11:25 | | **1:35** | 4.0 | 2.53 | 82.82 | **85.24** |
| | Stampy | **0.3** | | 1.4 | 38:51 | | **4:17** | 7.0 | 3.05 | 82.52 | **86.08** |
| YOR | BWA | **5.0** | 91 | 11.0 | 533:38 | 133:22 | **171:31** | 1.8 | 2.82 | 89.24 | **90.56** |
| | Bowtie 2 | **3.7** | | 10.9 | 352:49 | | **89:11** | 1.6 | 3.50 | 90.48 | **91.46** |
| | Novoalign | **8.0** | | 10.9 | 969:19 | | **368:39** | 1.9 | 2.98 | 87.52 | **90.41** |
| | Stampy | **2.7** | | 10.9 | 3397:02 | | **453:59** | 5.8 | 3.48 | 88.38 | **91.18** |

# Chapter 7

# Discussion

The scale and characteristics of the dataset are crucial aspects of genomic data analysis that lead to different algorithmic and statistical approaches for different problems. For finding homogeneous segments in DNA sequences and CNV detection from array-CGH data, we rely on HMMs, which are natural statistical models for segmentation problems. We argue in favor of superior Bayesian HMMs and show that, by utilizing short repeated sequences and, in case of continuous observations, grouping together similar consecutive observations, MCMC sampling can be made faster, thereby making Bayesian HMMs much more practical for these problems than before. For HTS read mapping, we present a $q$-gram frequency based readmapper, which shows competitive results in detecting genetic variants and brings new ideas to the well studied read mapping problem. However, high-throughput nature of these datasets prohibits the use of computationally demanding sensitive read mappers. We show that these datasets can be effectively reduced through clustering, and applications, such as read mapping, can directly work on those reduced representations to gain significant advantage in overall running time.

In the following, we present the final remarks and future directions for each contribution of this thesis.

## Exploiting Repetition in Discrete Sequences

In chapter 3, we present a modified version of the forward-backward Gibbs sampling algorithm for Bayesian analysis with a logarithmic improvement in running time. We use the four Russians method to pre-compute all possible quantities of interest and show that exact sampling can work with fewer forward variables by using the pre-computed

quantities. We demonstrate the advantage of our method on the DNA segmentation problem. As biological sequences are often long and the alphabet size is small, our approach can be adopted to make Bayesian computations faster in other biological applications. A natural extension to our approach would be applying other compression schemes. In some cases, when observations in a sequence are less uniform in nature, other schemes may outperform the four Russians method.

### Compressed Gaussian Observations

In chapter 4, we propose a method to accelerate MCMC for Bayesian HMMs modeling CNVs in arrayCGH and SNParray data. Our method constitutes of ideas from spatial index structures for several consecutive observations and approximate computations based on geometric arguments for HMM. We are able to achieve significant speed-ups on these biological datasets, which measure chromosomal aberrations and copy number variations, while maintaining competitive prediction accuracy compared to the state-of-the-art. As datasets with even higher resolution, both from higher density DNA microarrays and next generation sequencing, become available, we believe that the need for precise *and* efficient MCMC techniques will increase.

Applying approximate sampling to multi-dimensional observations—to jointly analyze data sets for recurrent CNVs [166] instead of analyzing individuals and post-processing results—and considering more complicated HMM topologies and observation densities are directions for our future work.

### Geometric Embeddings for HTS Reads

In chapter 5, we address the problem of mapping HTS reads in an indel-tolerant manner by establishing geometric embedding as a promising paradigm, allowing identification of structural variants: We map reads and genomic locations to trinucleotide frequency vectors, embedding them in $\mathbb{R}^{64}$. The $L_1$ distance between $q$-gram frequency vectors provides a lower bound for an edit distance with affine gap costs in which even long indels have small distance, improving the sensitivity of their detection. The problem

of approximate matching is thus transformed into one of computing nearest neighbors using a spatial index.

An obvious area of future investigation is a parallel distributed index which will allow our readmapper to run on clusters. Generally, we expect further improvements in running times, and consequently in accuracy, from an spatial index tailored specifically to the high-dimensional, integer coordinate problem setting, e.g. an adaptation of X-trees [102] or through the use of locality-sensitive hashing [167]. Additional improvements in terms of both memory and running time can be made by using batch processing for queries. The simplistic evaluation of putative matches using Levensthein distance can be replaced by a statistical, quality-score aware analysis following the lead of Stampy [99] and LAST [123, 124], which attribute their success to a large degree to the quality of their putative hit filtering. In our geometric embedding, quality scores can be used while searching for putative hits by using floating-point or fixed-point arithmetic and fractional count contributions for low-quality nucleotides.

### Reduced Representation through Clustering

In chapter 6, we propose to cluster HTS reads as a first step in the analysis and show that our greedy clustering algorithm accelerates read mapping. We observe speed-ups for all readmappers tested ranging from 1.4 to 8.9 with very little alternate mapping between individual and clustered mapping for high-quality reads, and low level of alternate mapping rate for all reads with a small loss in concordance for some datasets. We do expect even more favorable results for read mapping if there is a lot of genetic variation in the sample genome, such as in cancer datasets, or the evolutionary distance between sample and reference genome is large.

There are several areas in which the method can be improved, the most important ones being cluster quality and size, as running time is inversely proportional to the total number of clusters. It seems reasonable that the singleton clusters (that is a cluster with cardinality one, which arises either due to high sequencing error or very low coverage) can be reduced by additional passes, possibly with a smaller value of $k$ for k-mers.

The additional effort in clustering is very likely to be made up in mapping, by net savings in running time. So far read mapping is performed with anchor reads without any post-processing. An obvious improvement can be made by considering clusters as an instance of multiple sequence alignment problem, and correcting errors using the alignment information. Moreover, computing an elongated consensus sequence and using it instead of the anchor read is also desirable. However, one has to judiciously choose, based on coverage and within cluster sequence variation, the actual fragment of the consensus to use for mapping to see further improvements.

# Appendices

# Appendix A

# Notations

## All chapters

$A$ The transition probability matrix.

$B$ The observation probability matrix.

$O$ The observation sequence.

$O_{i,j}$ The consecutive observation sequence.

$Q$ The state sequence.

$Q_{ML}$ Viterbi path, the most likely state path of a HMM given an observation sequence.

$Q_{i,j}$ The consecutive state sequence.

$S$ The set of states.

$T$ The length of the observation sequence.

$\alpha_t(j)$ The forward variable for state $j$ at time $t$.

$\pi$ The initial probability over states.

$\theta$ The HMM with parameters $A, B$ and $\pi$.

$\theta^{A_i}$ The hyperparameter for the multinomial distribution over transitions from state $i$.

$\theta^{B_i}$ The hyperparameter for the multinomial distribution over discrete observations in state $i$.

$\theta^{\pi}$ The hyperparameter for the multinomial distribution over initial probability over states.

$\theta_{ML}$ The most likely parameters of a HMM.

$\tilde{\mu_i}$ One of the hyperparameters for the mean of the Gaussian distribution for state $i$.

$\tilde{\sigma_i}$ One of the hyperparameters for the mean of the Gaussian distribution for state $i$.

$a_i$ One of the hyperparameters for the variance of the Gaussian distribution for state $i$.

$a_{i,j}$ The probability of making a transition from state $i$ ot state $j$.

$b_i$ One of the hyperparameters for the variance of the Gaussian distribution for state $i$.

$b_{i,o}$ The probability of observing $o$ in state $i$.

## Chapter 3

$\gamma$ The order of the observation process.

## Chapter 4

$T'$ The total number of blocks after compression.

## Chapter 5

$L_1$ The $L_1$ distance between two vectors.

$\alpha$ The number of consecutive frequency vectors which are represented by their differences from a reference vector.

$\beta$ Maximum number of alternate path searched in a kd-tree.

$\tau$ Lowest number of levels ignored in a kd-tree.

$d$ The length of a frequency vector.

$n$ Total number of reads in the dataset in consideration.

$q$ The length of individual $q$-grams.

### Chapter 6

$L$ Read length.

$R$ The read library.

$\alpha$ Ratio between required minimum overlap length and the read length.

$\beta$ Minimum required similarity between reads in an overlap.

$\epsilon$ The sequencing error rate.

$l$ Overlap length between two reads.

$r_i$ $i$-th read in the library.

# Appendix B

# Abbreviations

**arrayCGH** Array comparative genomic hybridization.

**CNV** Copy number variation.

**DNA** Deoxyribonucleic acid.

**HMM** Statistical model.

**HTS** High throughput sequencing.

**MAP** Maximum a posterior.

**MCMC** Markov Chain Monte Carlo.

**ML** Maximum likelihood.

**RNA** Ribonucleic acid.

**SNP** Single neucleotide polymorphism.

**SNParray** Single neucleotide polymorphism array.

# References


[1] Md Pavel Mahmud and Alexander Schliep. Speeding up bayesian HMM by the four russians method. In TeresaM. Przytycka and Marie-France Sagot, editors, *Algorithms in Bioinformatics*, volume 6833 of *Lecture Notes in Computer Science*, pages 188–200. Springer Berlin Heidelberg, 2011.

[2] Md Pavel Mahmud and Alexander Schliep. Fast MCMC sampling for hidden markov models to determine copy number variations. *BMC Bioinformatics*, 12:428, 2011.

[3] Md Pavel Mahmud, John Wiedenhoeft, and Alexander Schliep. Indel-tolerant read mapping with trinucleotide frequencies using cache-oblivious kd-trees. *Bioinformatics*, 28(18):i325–i332, September 2012.

[4] Md Pavel Mahmud and Alexander Schliep. TreQ-CG: Clustering accelerates high-throughput sequencing read mapping. *arXiv preprint arXiv:1404.2872*, 2014.

[5] A M Maxam and W Gilbert. A new method for sequencing dna. *Proceedings of the National Academy of Sciences*, 74(2):560–564, 1977.

[6] F. Sanger and A.R. Coulson. A rapid method for determining sequences in {DNA} by primed synthesis with {DNA} polymerase. *Journal of Molecular Biology*, 94(3):441 – 448, 1975.

[7] F. Sanger, S. Nicklen, and A. R. Coulson. Dna sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74(12):5463–5467, 1977.

[8] E S Lander, L M Linton, B Birren, C Nusbaum, M C Zody, J Baldwin, K Devon, K Dewar, M Doyle, W FitzHugh, R Funke, D Gage, K Harris, A Heaford, J Howland, L Kann, J Lehoczky, R LeVine, P McEwan, K McKernan, J Meldrim, J P Mesirov, C Miranda, W Morris, J Naylor, M Rosetti, R Santos, A Sheridan, C Sougnez, N Stange-Thomann, N Stojanovic, A Subramanian, D Wyman, J Rogers, J Sulston, R Ainscough, S Beck, D Bentley, J Burton, C Clee, N Carter, A Coulson, R Deadman, P Deloukas, A Dunham, I Dunham, R Durbin, L French, D Grafham, S Gregory, T Hubbard, S Humphray, A Hunt, M Jones, C Lloyd, A McMurray, L Matthews, S Mercer, S Milne, J C Mullikin, A Mungall, R Plumb, M Ross, R Shownkeen, S Sims, R H Waterston, R K Wilson, L W Hillier, J D McPherson, M A Marra, E R Mardis, L A Fulton, A T Chinwalla, K H Pepin, W R Gish, S L Chissoe, M C Wendl, K D Delehaunty, T L Miner, A Delehaunty, J B Kramer, L L Cook, R S Fulton, D L Johnson, P J Minx, S W Clifton, T Hawkins, E Branscomb, P Predki, P Richardson, S Wenning, T Slezak, N Doggett, J F Cheng, A Olsen, S Lucas, C Elkin, E Uberbacher, M Frazier, R A Gibbs, D M Muzny, S E Scherer, J B Bouck, E J Sodergren, K C Worley, C M Rives, J H

Gorrell, M L Metzker, S L Naylor, R S Kucherlapati, D L Nelson, G M Weinstock, Y Sakaki, A Fujiyama, M Hattori, T Yada, A Toyoda, T Itoh, C Kawagoe, H Watanabe, Y Totoki, T Taylor, J Weissenbach, R Heilig, W Saurin, F Artiguenave, P Brottier, T Bruls, E Pelletier, C Robert, P Wincker, D R Smith, L Doucette-Stamm, M Rubenfield, K Weinstock, H M Lee, J Dubois, A Rosenthal, M Platzer, G Nyakatura, S Taudien, A Rump, H Yang, J Yu, J Wang, G Huang, J Gu, L Hood, L Rowen, A Madan, S Qin, R W Davis, N A Federspiel, A P Abola, M J Proctor, R M Myers, J Schmutz, M Dickson, J Grimwood, D R Cox, M V Olson, R Kaul, C Raymond, N Shimizu, K Kawasaki, S Minoshima, G A Evans, M Athanasiou, R Schultz, B A Roe, F Chen, H Pan, J Ramser, H Lehrach, R Reinhardt, W R McCombie, M de la Bastide, N Dedhia, H Blöcker, K Hornischer, G Nordsiek, R Agarwala, L Aravind, J A Bailey, A Bateman, S Batzoglou, E Birney, P Bork, D G Brown, C B Burge, L Cerutti, H C Chen, D Church, M Clamp, R R Copley, T Doerks, S R Eddy, E E Eichler, T S Furey, J Galagan, J G Gilbert, C Harmon, Y Hayashizaki, D Haussler, H Hermjakob, K Hokamp, W Jang, L S Johnson, T A Jones, S Kasif, A Kaspryzk, S Kennedy, W J Kent, P Kitts, E V Koonin, I Korf, D Kulp, D Lancet, T M Lowe, A McLysaght, T Mikkelsen, J V Moran, N Mulder, V J Pollara, C P Ponting, G Schuler, J Schultz, G Slater, A F Smit, E Stupka, J Szustakowski, D Thierry-Mieg, J Thierry-Mieg, L Wagner, J Wallis, R Wheeler, A Williams, Y I Wolf, K H Wolfe, S P Yang, R F Yeh, F Collins, M S Guyer, J Peterson, A Felsenfeld, K A Wetterstrand, A Patrinos, M J Morgan, P de Jong, J J Catanese, K Osoegawa, H Shizuya, S Choi, Y J Chen, J Szustakowki, and International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, Feb 2001.

[9] Eleanor Raffan and Robert K. Semple. Next generation sequencingimplications for clinical practice. *British Medical Bulletin*, 99(1):53–71, 2011.

[10] Yury Lifshits, Shay Mozes, Oren Weimann, and Michal Ziv-Ukelson. Speeding up HMM decoding and training by exploiting sequence repetitions. *Algorithmica*, 54(3):379–399, 2009.

[11] Shay Mozes, Oren Weimann, and Michal Ziv-ukelson. Speeding up HMM decoding and training by exploiting sequence repetitions. In *In Proc. 18th Annual Symposium On Combinatorial Pattern Matching (CPM), LNCS 4580*, pages 4–15. Springer-Verlag, 2007.

[12] Adam L. Buchsbaum and Raffaele Giancarlo. Algorithmic aspects in speech recognition: an introduction. *J. Exp. Algorithmics*, 2, January 1997.

[13] Christopher D. Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, Cambridge, MA, USA, 1999.

[14] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme J. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[15] Chris Burge and Samuel Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268(1):78 – 94, 1997.

[16] Gary A. Churchill. Hidden Markov chains and the analysis of genome structure. *Computers and Chemistry*, 16(2):107 – 115, 1992.

[17] Gary Churchill. Stochastic models for heterogeneous DNA sequences. *Bulletin of Mathematical Biology*, 51:79–94, 1989. 10.1007/BF02458837.

[18] J S Liu and C E Lawrence. Bayesian inference on biopolymer models. *Bioinformatics*, 15(1):38–52, 1999.

[19] Pierre Nicolas, Laurent Bize, Florence Muri, Mark Hoebeke, Franois Rodolphe, S. Dusko Ehrlich, Bernard Prum, and Philippe Bessires. Mining bacillus subtilis chromosome heterogeneities using Hidden Markov Models. *Nucleic Acids Research*, 30(6):1418–1426, 2002.

[20] Uwe Ohler, Heinrich Niemann, Guo-chun Liao, and Gerald M. Rubin. Joint modeling of DNA sequence and physical properties to improve eukaryotic promoter recognition. *Bioinformatics*, 17(suppl 1):S199–S206, 2001.

[21] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series)*. The MIT Press, August 2012.

[22] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.

[23] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[24] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[25] W.R. Gilks, W.R. Gilks, S. Richardson, and D.J. Spiegelhalter. *Markov chain Monte Carlo in practice*. Interdisciplinary statistics. Chapman & Hall, 1996.

[26] Steven Scott. Bayesian methods for hidden markov models: Recursive computing in the 21st century. *Journal of the American Statistical Association*, pages 337–351, Mar 2002.

[27] Siddhartha Chib. Calculating posterior distributions and modal estimates in markov mixture models. *Journal of Econometrics*, 75(1):79–97, November 1996.

[28] Lin Liu, Yinhu Li, Siliang Li, Ni Hu, Yimin He, Ray Pong, Danni Lin, Lihua Lu, and Maggie Law. Comparison of next-generation sequencing systems. *J Biomed Biotechnol*, 2012:251364, 2012.

[29] Michael A Quail, Miriam Smith, Paul Coupland, Thomas D Otto, Simon R Harris, Thomas R Connor, Anna Bertoni, Harold P Swerdlow, and Yong Gu. A tale of three next generation sequencing platforms: comparison of ion torrent, pacific biosciences and illumina miseq sequencers. *BMC Genomics*, 13:341, 2012.

[30] Richard J. Boys and Daniel A. Henderson. A Bayesian approach to DNA sequence segmentation. *Biometrics*, 60(3):573–581, 2004.

[31] Richard J. Boys, Daniel A. Henderson, and Darren J. Wilkinson. Detecting homogeneous segments in DNA sequences by using Hidden Markov Models. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 49(2):pp. 269–285, 2000.

[32] Jerome V. Braun and Hans-Georg Muller. Statistical methods for DNA sequence segmentation. *Statistical Science*, 13(2):pp. 142–162, 1998.

[33] Michael Andrec, Ronald M. Levy, and David S. Talaga. Direct determination of kinetic rates from single-molecule photon arrival trajectories using Hidden Markov Models. *The Journal of Physical Chemistry A*, 107(38):7454–7464, 2003. PMID: 19626138.

[34] Subharup Guha, Yi Li, and Donna Neuberg. Bayesian hidden markov modeling of array cgh data. *Journal of the American Statistical Association*, 103:485–497, June 2008.

[35] Toby A. Patterson, Len Thomas, Chris Wilcox, Otso Ovaskainen, and Jason Matthiopoulos. State-space models of individual animal movement. *Trends in Ecology and Evolution*, 23(2):87 – 94, 2008.

[36] Benjamin D. Redelings and Marc A. Suchard. Joint Bayesian estimation of alignment and phylogeny. *Systematic Biology*, 54(3):401–418, 2005.

[37] Steven L. Scott. A Bayesian paradigm for designing intrusion detection systems. *Computational Statistics and Data Analysis*, 45(1):69 – 83, 2004. Computer Security and Statistics.

[38] Christopher A. Sims and Tao Zha. Were there regime switches in U.S. monetary policy? *The American Economic Review*, 96(1):pp. 54–81, 2006.

[39] Yelena Frid and Dan Gusfield. A simple, practical and complete O(n$\hat{3}$/log n)-time algorithm for RNA folding using the Four-Russians speedup. In *Proceedings of the 9th international conference on Algorithms in bioinformatics*, WABI'09, pages 97–107, Berlin, Heidelberg, 2009. Springer-Verlag.

[40] William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18 – 31, 1980.

[41] Eugene Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986. 10.1007/BF01840446.

[42] Gene Myers. A Four Russians algorithm for regular expression pattern matching. *J. ACM*, 39:432–448, April 1992.

[43] Anders Krogh. Two methods for improving performance of a HMM and their application for gene finding. In *Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology*, pages 179–186. AAAI Press, 1997.

[44] Jonathan Sebat, B Lakshmi, Jennifer Troge, Joan Alexander, Janet Young, Pär Lundin, Susanne Månér, Hillary Massa, Megan Walker, Maoyen Chi, Nicholas Navin, Robert Lucito, John Healy, James Hicks, Kenny Ye, Andrew Reiner,

T Conrad Gilliam, Barbara Trask, Nick Patterson, Anders Zetterberg, and Michael Wigler. Large-scale copy number polymorphism in the human genome. *Science*, 305(5683):525–8, Jul 2004.

[45] A. John Iafrate, Lars Feuk, Miguel N Rivera, Marc L Listewnik, Patricia K Donahoe, Ying Qi, Stephen W Scherer, and Charles Lee. Detection of large-scale variation in the human genome. *Nat Genet*, 36(9):949–951, Sep 2004.

[46] Pawe Stankiewicz and James R Lupski. Structural variation in the human genome and its role in disease. *Annu Rev Med*, 61:437–455, 2010.

[47] Edwin H Cook and Stephen W Scherer. Copy-number variations associated with neuropsychiatric conditions. *Nature*, 455(7215):919–923, Oct 2008.

[48] Dalila Pinto, Alistair T Pagnamenta, Lambertus Klei, Richard Anney, Daniele Merico, Regina Regan, Judith Conroy, Tiago R Magalhaes, Catarina Correia, Brett S Abrahams, Joana Almeida, Elena Bacchelli, Gary D Bader, Anthony J Bailey, Gillian Baird, Agatino Battaglia, Tom Berney, Nadia Bolshakova, Sven Blte, Patrick F Bolton, Thomas Bourgeron, Sean Brennan, Jessica Brian, Susan E Bryson, Andrew R Carson, Guillermo Casallo, Jillian Casey, Brian H Y Chung, Lynne Cochrane, Christina Corsello, Emily L Crawford, Andrew Crossett, Cheryl Cytrynbaum, Geraldine Dawson, Maretha de Jonge, Richard Delorme, Irene Drmic, Eftichia Duketis, Frederico Duque, Annette Estes, Penny Farrar, Bridget A Fernandez, Susan E Folstein, Eric Fombonne, Christine M Freitag, John Gilbert, Christopher Gillberg, Joseph T Glessner, Jeremy Goldberg, Andrew Green, Jonathan Green, Stephen J Guter, Hakon Hakonarson, Elizabeth A Heron, Matthew Hill, Richard Holt, Jennifer L Howe, Gillian Hughes, Vanessa Hus, Roberta Igliozzi, Cecilia Kim, Sabine M Klauck, Alexander Kolevzon, Olena Korvatska, Vlad Kustanovich, Clara M Lajonchere, Janine A Lamb, Magdalena Laskawiec, Marion Leboyer, Ann Le Couteur, Bennett L Leventhal, Anath C Lionel, Xiao-Qing Liu, Catherine Lord, Linda Lotspeich, Sabata C Lund, Elena Maestrini, William Mahoney, Carine Mantoulan, Christian R Marshall, Helen McConachie, Christopher J McDougle, Jane McGrath, William M McMahon, Alison Merikangas, Ohsuke Migita, Nancy J Minshew, Ghazala K Mirza, Jeff Munson, Stanley F Nelson, Carolyn Noakes, Abdul Noor, Gudrun Nygren, Guiomar Oliveira, Katerina Papanikolaou, Jeremy R Parr, Barbara Parrini, Tara Paton, Andrew Pickles, Marion Pilorge, Joseph Piven, Chris P Ponting, David J Posey, Annemarie Poustka, Fritz Poustka, Aparna Prasad, Jiannis Ragoussis, Katy Renshaw, Jessica Rickaby, Wendy Roberts, Kathryn Roeder, Bernadette Roge, Michael L Rutter, Laura J Bierut, John P Rice, Jeff Salt, Katherine Sansom, Daisuke Sato, Ricardo Segurado, Ana F Sequeira, Lili Senman, Naisha Shah, Val C Sheffield, Latha Soorya, Ins Sousa, Olaf Stein, Nuala Sykes, Vera Stoppioni, Christina Strawbridge, Raffaella Tancredi, Katherine Tansey, Bhooma Thiruvahindrapduram, Ann P Thompson, Susanne Thomson, Ana Tryfon, John Tsiantis, Herman Van Engeland, John B Vincent, Fred Volkmar, Simon Wallace, Kai Wang, Zhouzhi Wang, Thomas H Wassink, Caleb Webber, Rosanna Weksberg, Kirsty Wing, Kerstin Wittemeyer, Shawn Wood, Jing Wu, Brian L Yaspan, Danielle Zurawiecki, Lonnie Zwaigenbaum, Joseph D Buxbaum, Rita M Cantor, Edwin H Cook, Hilary Coon, Michael L Cuccaro, Bernie Devlin, Sean Ennis,

Louise Gallagher, Daniel H Geschwind, Michael Gill, Jonathan L Haines, Joachim Hallmayer, Judith Miller, Anthony P Monaco, John I Nurnberger, Andrew D Paterson, Margaret A Pericak-Vance, Gerard D Schellenberg, Peter Szatmari, Astrid M Vicente, Veronica J Vieland, Ellen M Wijsman, Stephen W Scherer, James S Sutcliffe, and Catalina Betancur. Functional impact of global rare copy number variation in autism spectrum disorders. *Nature*, 466(7304):368–372, Jul 2010.

[49] David St Clair. Copy number variation and schizophrenia. *Schizophr Bull*, 35(1):9–12, Jan 2009.

[50] Sohrab P Shah, Xiang Xuan, Ron J DeLeeuw, Mehrnoush Khojasteh, Wan L Lam, Raymond Ng, and Kevin P Murphy. Integrating copy number polymorphisms into array cgh analysis using a robust hmm. *Bioinformatics*, 22(14):e431–e439, Jul 2006.

[51] Subharup Guha, Yi Li, and Donna Neuberg. Bayesian hidden markov modeling of array cgh data. *Journal of the American Statistical Association*, 103(482):485–497, Dec 2008.

[52] Robin Andersson, Carl E. G. Bruder, Arkadiusz Piotrowski, Uwe Menzel, Helena Nord, Johanna Sandgren, Torgeir R. Hvidsten, Teresita Diaz de Sthl, Jan P. Dumanski, and Jan Komorowski. A segmental maximum a posteriori approach to genome-wide copy number profiling. *Bioinformatics*, 24(6):751–758, 2008.

[53] J Fridlyand, AM Snijders, D Pinkel, DG Albertson, and AN Jain. Hidden markov models approach to the analysis of array cgh data. *J Multivariate Anal*, 90(1):132–153, Dec 2004.

[54] Kai Wang, Mingyao Li, Dexter Hadley, Rui Liu, Joseph Glessner, Struan F A Grant, Hakon Hakonarson, and Maja Bucan. Penncnv: an integrated hidden markov model designed for high-resolution copy number variation detection in whole-genome snp genotyping data. *Genome Research*, 17(11):1665–74, Nov 2007.

[55] Adam B Olshen, E S Venkatraman, Robert Lucito, and Michael Wigler. Circular binary segmentation for the analysis of array-based DNA copy number data. *Biostatistics (Oxford, England)*, 5(4):557–72, Sep 2004.

[56] F Picard, S Robin, E Lebarbier, and J-J Daudin. A segmentation/clustering model for the analysis of array cgh data. *Biometrics*, 63(3):758–66, Aug 2007.

[57] Paul H C Eilers and Renée X de Menezes. Quantile smoothing of array CGH data. *Bioinformatics*, 21(7):1146–53, Mar 2005.

[58] Robert Tibshirani and Pei Wang. Spatial smoothing and hot spot detection for cgh data using the fused lasso. *Biostatistics*, 9(1):18–29, 2008.

[59] Pei Wang, Young Kim, Jonathan Pollack, Balasubramanian Narasimhan, and Robert Tibshirani. A method for calling gains and losses in array cgh data. *Biostatistics (Oxford, England)*, 6(1):45–58, Dec 2005.

[60] Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 277–281, New York, NY, USA, 1999. ACM.

[61] J. Fritsch and I. Rogina. The bucket box intersection (bbi) algorithm for fast approximative evaluation of diagonal mixture gaussians. In *In Proc. ICASSP*, pages 837–840, 1996.

[62] S. Srivastava. Fast gaussian evaluations in large vocabulary continuous speech recognition. *M.S. Thesis, Department of Electrical and Computer Engineering, Mississippi State University*, Oct. 2002.

[63] Stan Salvador and Philip Chan. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '04, pages 576–584, Washington, DC, USA, 2004. IEEE Computer Society.

[64] Hanni Willenbrock and Jane Fridlyand. A comparison study: applying segmentation to array cgh data for downstream analyses. *Bioinformatics*, 21(22):4084–4091, Nov 2005.

[65] Ronald J. De Leeuw, Jonathan J. Davies, Andreas Rosenwald, Gwyn Bebb, Y D. Gascoyne, Martin J. S. Dyer, Louis M. Staudt, Jose A. Martinez-climent, and Wan L. Lam. Comprehensive whole genome array cgh profiling of mantle cell lymphoma model genomes. *Hum. Mol. Genet*, 13:1827–1837, 2004.

[66] Antoine M. Snijders, Norma Nowak, Richard Segraves, Stephanie Blackwood, Nils Brown, Jeffrey Conroy, Greg Hamilton, Anna Katherine Hindle, Bing Huey, Karen Kimura, Sindy Law, Ken Myambo, Joel Palmer, Bauke Ylstra, Jingzhu Pearl Yue, Joe W. Gray, Ajay N. Jain, Daniel Pinkel, and Donna G. Albertson. Assembly of microarrays for genome-wide measurement of dna copy number. *Nat Genet*, 29(3):263–264, Nov 2001.

[67] Markus Bredel, Claudia Bredel, Dejan Juric, Griffith R. Harsh, Hannes Vogel, Lawrence D. Recht, and Branimir I. Sikic. High-resolution genome-wide mapping of genetic alterations in human glial brain tumors. *Cancer Research*, 65(10):4088–4096, 2005.

[68] T. Harada, C. Chelala, V. Bhakta, T. Chaplin, K. Caulee, P. Baril, B. D. Young, and N. R. Lemoine. Genome-wide dna copy number analysis in pancreatic cancer using high-density single nucleotide polymorphism arrays. *Oncogene*, 27(13):1951–1960, Oct 2007.

[69] Weil R Lai, Mark D Johnson, Raju Kucherlapati, and Peter J Park. Comparative analysis of algorithms for identifying amplifications and deletions in array cgh data. *Bioinformatics*, 21(19):3763–70, Sep 2005.

[70] Yasuhito Nannya, Masashi Sanada, Kumi Nakazaki, Noriko Hosoya, Lili Wang, Akira Hangaishi, Mineo Kurokawa, Shigeru Chiba, Dione K Bailey, Giulia C Kennedy, and Seishi Ogawa. A robust algorithm for copy number detection using

high-density oligonucleotide single nucleotide polymorphism genotyping arrays. *Cancer Res*, 65(14):6071–6079, Jul 2005.

[71] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12:153–157, 1947. 10.1007/BF02295996.

[72] Sandro Morganella, Luigi Cerulo, Giuseppe Viglietto, and Michele Ceccarelli. VEGA: variational segmentation for copy number detection. *Bioinformatics*, 26(24):3020–3027, 2010.

[73] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[74] Leonid Boytsov. Indexing methods for approximate dictionary searching: Comparative analysis. *J. Exp. Algorithmics*, 16:1.1:1.1–1.1:1.91, May 2011.

[75] Leonid Boytsov. Indexing methods for approximate dictionary searching. *Journal of Experimental Algorithmics*, 16(1):1, May 2011.

[76] Heng Li and Nils Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, 11(5):473–83, September 2010.

[77] M. Burrows, D. J. Wheeler, M. Burrows, and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, 1994.

[78] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.

[79] Nawar Malhis and Steven J M Jones. High quality SNP calling using Illumina data at shallow coverage. *Bioinformatics (Oxford, England)*, 26(8):1029–35, April 2010.

[80] E. Bugnion, T. Roos, F. Shi, P. Widmayer, and F. Widmer. A spatial index for approximate multiple string matching. *Journal of the Brazilian Chemical Society*, 1(3):28–35, 1995.

[81] Esko Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–211, 1992.

[82] Gene Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM*, 46(3):395–415, 1999.

[83] Gonzalo Navarro, Erkki Sutinen, and Jorma Tarhio. Indexing text with approximate $q$-grams. *J. Discrete Algorithms*, 3(2-4):157–175, 2005.

[84] H Li, J Ruan, and R Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research*, 18(11):1851–1858, November 2008.

[85] David Weese, Anne-Katrin Emde, Tobias Rausch, Andreas Dring, and Knut Reinert. Razers fast read mapping with sensitivity control. *Genome Research*, 19(9):1646–1654, 2009.

[86] Stephen M Rumble, Phil Lacroute, Adrian V Dalca, Marc Fiume, Arend Sidow, and Michael Brudno. Shrimp: accurate mapping of short color-space reads. *PLoS Computational Biology*, 5(5):e1000386, May 2009.

[87] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrowswheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[88] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3):R25, Jan 2009.

[89] Ruiqiang Li, Yingrui Li, Karsten Kristiansen, and Jun Wang. Soap: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–4, Feb 2008.

[90] Erkki Sutinen and Wojciech Szpankowski. On the Collapse of q-Gram Filtration. In *FUN with Algorithms*, pages 178–193, 1998.

[91] Bin Ma, John Tromp, and Ming Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, Mar 2002.

[92] Stefan Burkhardt and Juha Kärkkäinen. Better filtering with gapped q-grams. *Fundam. Inf.*, 56(1,2):51–70, 2002.

[93] Ken Chen, John W Wallis, Michael D McLellan, David E Larson, Joelle M Kalicki, Craig S Pohl, Sean D McGrath, Michael C Wendl, Qunyuan Zhang, Devin P Locke, and et al. Breakdancer: an algorithm for high-resolution mapping of genomic structural variation. *Nature Methods*, 6(9):677–681, 2009.

[94] Fereydoun Hormozdiari, Can Alkan, Evan E. Eichler, and S. Cenk Sahinalp. Combinatorial algorithms for structural variation detection in high-throughput sequenced genomes. *Genome Research*, 19(7):1270–1278, 2009.

[95] Fereydoun Hormozdiari, Iman Hajirasouliha, Phuong Dao, Faraz Hach, Deniz Yorukoglu, Can Alkan, Evan E Eichler, and S Cenk Sahinalp. Next-generation variationhunter: combinatorial algorithms for transposon insertion discovery. *Bioinformatics*, 26(12):i350–i357, 2010.

[96] Jan O Korbel, Alexej Abyzov, Xinmeng Jasmine Mu, Nicholas Carriero, Philip Cayting, Zhengdong Zhang, Michael Snyder, and Mark B Gerstein. Pemer: a computational framework with simulation-based error models for inferring genomic structural variants from massive paired-end sequencing data. *Genome Biology*, 10(2):R23, 2009.

[97] Seunghak Lee, Fereydoun Hormozdiari, Can Alkan, and Michael Brudno. Modil: detecting small indels from clone-end sequencing with mixtures of distributions. *Nature Methods*, 6(7):473–474, 2009.

[98] Can Alkan, Bradley P. Coe, and Evan E. Eichler. Genome structural variation discovery and genotyping. *Nat Rev Genet*, 12(5):363–376, May 2011.

[99] Gerton Lunter and Martin Goodson. Stampy: A statistical algorithm for sensitive and fast mapping of illumina sequence reads. *Genome Research*, 21(6):936–939, 2011.

[100] Marshall Bern. Approximate closest-point queries in high dimensions. *Inf. Process. Lett.*, 45:95–99, February 1993.

[101] Norio Katayama and Shin'ichi Satoh. Sr-tree: An index structure for nearest-neighbor searching of high-dimensional point data. *Systems and Computers in Japan*, 29(6):59–73, 1998.

[102] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The x-tree : An index structure for high-dimensional data. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 28–39. Morgan Kaufmann, 1996.

[103] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, VLDB '87, pages 507–518, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[104] Ozgur Ozturk and Hakan Ferhatosmanoglu. Effective indexing and filtering for similarity search in large biosequence databases. In *BIBE*, pages 359–366. IEEE Computer Society, 2003.

[105] Ozgur Ozturk and Hakan Ferhatosmanoglu. Vector space indexing for biosequence similarity searches. *International Journal on Artificial Intelligence Tools*, 14(5):811–826, 2005.

[106] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373, 2001.

[107] Michael E. Houle and Jun Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In *ICDE*, pages 619–630. IEEE Computer Society, 2005.

[108] Bin Yao, Feifei Li, Marios Hadjieleftheriou, and Kun Hou. Approximate string search in spatial databases. In Feifei Li, Mirella M. Moro, Shahram Ghandeharizadeh, Jayant R. Haritsa, Gerhard Weikum, Michael J. Carey, Fabio Casati, Edward Y. Chang, Ioana Manolescu, Sharad Mehrotra, Umeshwar Dayal, and Vassilis J. Tsotras, editors, *ICDE*, pages 545–556. IEEE, 2010.

[109] Benjamin Bustos and Gonzalo Navarro. Improving the space cost of $k$-nn search in metric spaces by using distance estimators. *Multimedia Tools Appl.*, 41(2):215–233, 2009.

[110] Gonzalo Navarro and Edgar Chávez. A metric index for approximate string matching. *Theor. Comput. Sci.*, 352(1-3):266–279, 2006.

[111] David M. Mount and Sunil Arya. Ann: A library for approximate nearest neighbor searching, 2010. `http://www.ncbi.nlm.nih.gov/pubmed/19668202`.

[112] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[113] Dan Gusfield. *Algorithms on strings, trees, and sequences.* Cambridge University Press, Cambridge, 1997. Computer science and computational biology.

[114] Ashraf Kibriya and Eibe Frank. An empirical comparison of exact nearest neighbour algorithms. In Joost Kok, Jacek Koronacki, Ramon Lopez de Mantaras, Stan Matwin, Dunja Mladenic, and Andrzej Skowron, editors, *Knowledge Discovery in Databases: PKDD 2007*, volume 4702 of *Lecture Notes in Computer Science*, pages 140–151. Springer Berlin / Heidelberg, 2007.

[115] Pankaj K. Agarwal, Lars Arge, Andrew Danner, and Bryan Holland-Minkley. Cache-oblivious data structures for orthogonal range searching. In *Proceedings of the nineteenth annual symposium on Computational geometry*, SCG '03, pages 237–245, New York, NY, USA, 2003. ACM.

[116] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 285–, Washington, DC, USA, 1999. IEEE Computer Society.

[117] P. van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, pages 75–84, Washington, DC, USA, 1975. IEEE Computer Society.

[118] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Theory of Computing Systems*, 10:99–127, 1976. 10.1007/BF01683268.

[119] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[120] Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[121] Can Alkan, Jeffrey M. Kidd, Tomas Marques-Bonet, Gozde Aksay, Francesca Antonacci, Fereydoun Hormozdiari, Jacob O. Kitzman, Carl Baker, Maika Malig, Onur Mutlu, S. Cenk Sahinalp, Richard A. Gibbs, and Evan E. Eichler. Personalized copy number and segmental duplication maps using next-generation sequencing. *Nat Genet*, 41(10):1061–1067, October 2009.

[122] Zemin Ning, Anthony J. Cox, and James C. Mullikin. Ssaha: A fast search method for large dna databases. *Genome Research*, 11(10):1725–1729, 2001.

[123] Martin C. Frith, Raymond Wan, and Paul Horton. Incorporating sequence quality data into alignment improves dna read mapping. *Nucleic Acids Research*, 38(7):e100, 2010.

[124] Michiaki Hamada, Edward Wijaya, Martin C. Frith, and Kiyoshi Asai. Probabilistic alignments with quality scores: an application to short-read mapping toward accurate snp/indel detection. *Bioinformatics*, 27(22):3085–3092, 2011.

[125] Brent Ewing and Phil Green. Base-Calling of Automated Sequencer Traces Using Phred. II. Error Probabilities. *Genome Res.*, 8(3):186–194, 1998.

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

[127] The 1000 Genomes Project Consortium. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061–1073, October 2010.

[128] Stephen M Rumble, Phil Lacroute, Adrian V Dalca, Marc Fiume, Arend Sidow, and Michael Brudno. SHRiMP: accurate mapping of short color-space reads. *PLoS Computational Biology*, 5(5):e1000386, May 2009.

[129] Steve Hoffmann, Christian Otto, Stefan Kurtz, Cynthia M Sharma, Philipp Khaitovich, Jörg Vogel, Peter F Stadler, and Jörg Hackermüller. Fast mapping of short sequences with mismatches, insertions and deletions using index structures. *PLoS Computational Biology*, 5(9):e1000502, September 2009.

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

[131] Gerton Lunter and Martin Goodson. Stampy: a statistical algorithm for sensitive and fast mapping of Illumina sequence reads. *Genome Res*, 21(6):936–939, Jun 2011.

[132] Michiaki Hamada, Edward Wijaya, Martin C Frith, and Kiyoshi Asai. Probabilistic alignments with quality scores: an application to short-read mapping toward accurate SNP/indel detection. *Bioinformatics*, 27(22):3085–3092, November 2011.

[133] David Weese, Manuel Holtgrewe, and Knut Reinert. RazerS 3: faster, fully sensitive read mapping. *Bioinformatics*, 28(20):2592–2599, October 2012.

[134] Athena Ahmadi, Alexander Behm, Nagesh Honnalli, Chen Li, Lingjie Weng, and Xiaohui Xie. Hobbes: optimized gram-based methods for efficient read alignment. *Nucleic Acids Research*, 40(6):e41, 2012.

[135] Ergude Bao, Tao Jiang, Isgouhi Kaloshian, and Thomas Girke. SEED: efficient clustering of next-generation sequences. *Bioinformatics*, 27(18):2502–2509, September 2011.

[136] Kana Shimizu and Koji Tsuda. SlideSort: all pairs similarity search for short reads. *Bioinformatics*, 27(4):464–470, February 2011.

[137] Robert C. Edgar. Search and clustering orders of magnitude faster than blast. *Bioinformatics*, 26(19):2460–2461, 2010.

[138] W Timothy White and Michael Hendy. Compressing dna sequence databases with coil. *BMC Bioinformatics*, 9(1):242, 2008.

[139] Vladimir Yanovsky. Recoil - an algorithm for compression of extremely large datasets of dna data. *Algorithms for Molecular Biology*, 6(1):23, 2011.

[140] Faraz Hach, Ibrahim Numanagic, Can Alkan, and S Cenk Sahinalp. SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics*, 28(23):3051–3057, December 2012.

[141] Leena Salmela and Jan Schröder. Correcting errors in short reads by multiple alignments. *Bioinformatics*, 27(11):1455–1461, June 2011.

[142] Wei-Chun Kao, Andrew H. Chan, and Yun S. Song. ECHO: A reference-free short-read error correction algorithm. *Genome Research*, 21(7):1181–1192, July 2011.

[143] Zechen Chong, Jue Ruan, and Chung-I. Wu. Rainbow: an integrated tool for efficient clustering and assembling rad-seq reads. *Bioinformatics*, 28(21):2732–2737, 2012.

[144] Wei Qu, Shin-ichi Hashimoto, and Shinichi Morishita. Efficient frequency-based de novo short-read clustering for error trimming in next-generation sequencing. *Genome Research*, 19(7):1309–1315, 2009.

[145] Brendan Veeneman, Matthew Iyer, and Arul Chinnaiyan. Oculus: faster sequence alignment by streaming read compression. *BMC Bioinformatics*, 13(1):297, 2012.

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

[147] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

[148] Mengyao Zhao, Wan-Ping Lee, Erik Garrison, and Gabor T. Marth. SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications. *CoRR*, abs/1208.6350v2, 2013.

[149] Weichun Huang, Leping Li, Jason R. Myers, and Gabor T. Marth. Art: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2012.

[150] Charles H. Langley, Marc Crepeau, Charis Cardeno, Russell Corbett-Detig, and Kristian Stevens. Circumventing heterozygosity: Sequencing the amplified genome of a single haploid drosophila melanogaster embryo. *Genetics*, 188(2):239–246, 2011.

[151] P. Pipenbacher, A. Schliep, S. Schneckener, A. Schnhuth, D. Schomburg, and R. Schrader. Proclust: improved clustering of protein sequences with an extended graph-based approach. *Bioinformatics*, 18(suppl 2):S182–S191, 2002.

[152] Yaniv Loewenstein, Elon Portugaly, Menachem Fromer, and Michal Linial. Efficient algorithms for accurate hierarchical clustering of huge datasets: tackling the entire protein space. *Bioinformatics*, 24(13):i41–i49, 2008.

[153] Weizhong Li and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006.

[154] Beifang Niu, Limin Fu, Shulei Sun, and Weizhong Li. Artificial and natural duplicates in pyrosequencing reads of metagenomic data. *BMC Bioinformatics*, 11(1):187, 2010.

[155] Jack A. Gilbert, Dawn Field, Ying Huang, Rob Edwards, Weizhong Li, Paul Gilna, and Ian Joint. Detection of large numbers of novel sequences in the metatranscriptomes of complex marine microbial communities. *PLoS ONE*, 3(8):e3042, 08 2008.

[156] Shibu Yooseph, Granger Sutton, Douglas B Rusch, Aaron L Halpern, Shannon J Williamson, Karin Remington, Jonathan A Eisen, Karla B Heidelberg, Gerard Manning, Weizhong Li, Lukasz Jaroszewski, Piotr Cieplak, Christopher S Miller, Huiying Li, Susan T Mashiyama, Marcin P Joachimiak, Christopher van Belle, John-Marc Chandonia, David A Soergel, Yufeng Zhai, Kannan Natarajan, Shaun Lee, Benjamin J Raphael, Vineet Bafna, Robert Friedman, Steven E Brenner, Adam Godzik, David Eisenberg, Jack E Dixon, Susan S Taylor, Robert L Strausberg, Marvin Frazier, and J. Craig Venter. The ¡italic¿sorcerer ii¡/italic¿ global ocean sampling expedition: Expanding the universe of protein families. *PLoS Biol*, 5(3):e16, 03 2007.

[157] Weizhong Li, John C. Wooley, and Adam Godzik. Probing metagenomics by rapid cluster analysis of very large datasets. *PLoS ONE*, 3(10):e3375, 10 2008.

[158] Geo Pertea, Xiaoqiu Huang, Feng Liang, Valentin Antonescu, Razvan Sultana, Svetlana Karamycheva, Yuandan Lee, Joseph White, Foo Cheung, Babak Parvizi, Jennifer Tsai, and John Quackenbush. Tigr gene indices clustering tools (tgicl): a software system for fast clustering of large est datasets. *Bioinformatics*, 19(5):651–652, 2003.

[159] John Burke, Dan Davison, and Winston Hide. d2 cluster: A validated method for clustering est and full-length cdna sequences. *Genome Research*, 9(11):1135–1142, 1999.

[160] Andrey Ptitsyn and Winston Hide. Clu: A new algorithm for est clustering. *BMC Bioinformatics*, 6(Suppl 2):S3, 2005.

[161] Scott Hazelhurst and Zsuzsanna Liptk. Kaboom! a new suffix array based algorithm for clustering expression data. *Bioinformatics*, 27(24):3348–3355, 2011.

[162] Alexander Solovyov and W Lipkin. Centroid based clustering of high throughput sequencing reads based on n-mer counts. *BMC Bioinformatics*, 14(1):268, 2013.

[163] Limin Fu, Beifang Niu, Zhengwei Zhu, Sitao Wu, and Weizhong Li. Cd-hit: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, 28(23):3150–3152, 2012.

[164] Mohammadreza Ghodsi, Bo Liu, and Mihai Pop. Dnaclust: accurate and efficient clustering of phylogenetic marker genes. *BMC Bioinformatics*, 12(1):271, 2011.

[165] Weizhong Li, Limin Fu, Beifang Niu, Sitao Wu, and John Wooley. Ultrafast clustering algorithms for metagenomic sequence analysis. *Briefings in Bioinformatics*, 13(6):656–668, November 2012.

[166] Sohrab P Shah, Wan L Lam, Raymond T Ng, and Kevin P Murphy. Modeling recurrent dna copy number alterations in array cgh data. *Bioinformatics*, 23(13):i450–i458, Jul 2007.

[167] Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg. Locality sensitive hashing: a comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358, August 2010. QUAERO.