INFORMATION THEORETIC AND MACHINE LEARNING TECHNIQUES FOR EMERGING GENOMIC DATA ANALYSIS

BY
MINJI KIM

DISSERTATION
Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical and Computer Engineering in the Graduate College of the University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Doctoral Committee:
Professor Olgica Milenkovic, Co-Chair
Professor Jun S. Song, Co-Chair
Professor Venugopal V. Veeravalli
Professor Saurabh Sinha
Assistant Professor Jian Peng
ABSTRACT

The completion of the Human Genome Project in 2003 opened a new era for scientists. Through advanced high-throughput sequencing technologies, we now have access to a large amount of genomic data and we can use it to answer key biological questions, such as the factors contributing to the development of cancer. Large data sets and rapidly advancing sequencing technology pose challenges for processing and storing large volumes of genomic data. Moreover, the analysis of datasets may be both computationally and theoretically challenging because statistical methods have not been developed for new emerging data. In this work, I address some of these problems using tools from information theory and machine learning.

First, I focus on the data processing and storage aspect of metagenomics, the study of microbial communities in environmental samples and human organs. In particular, I introduce MetaCRAM, the first software suite specialized for metagenomic sequencing data processing and compression, and demonstrate that MetaCRAM compresses data to 2-13 percent of the original file size.

Second, I analyze a biological dataset assaying the propensity of a DNA sequence to form a four-stranded structure called “G-quadruplex” (GQ). GQ structures have been proposed to regulate diverse key biological processes including transcription, replication, and translation. I present main factors that lead to GQ formation, and propose highly accurate linear regression and Gaussian process regression models to predict the ability of a DNA sequence to fold into GQ.

Third, I study data structures to analyze and store three-dimensional chromatin conformation data generated from high-throughput sequencing technologies. In particular, I examine statistical properties of Hi-C contact maps and propose a few suitable formats to encode pairwise interactions between genome locations.
To my academic parents.
ACKNOWLEDGMENTS

My graduate study was filled with blessings from the following individuals.

I must first acknowledge my referees, anonymous reviewers, and the selection committee of the NSF Graduate Research Fellowship application for their strong support in funding my study for three years. I also thank Professors Venugopal V. Veeravalli, Saurabh Sinha, Jian Peng, Sua Myong, Doctor Alex Kreig, and members of the Milenkovic and Song group for their valuable feedback and discussions on my work.

My parents deserve ample credit for promoting both my scholarly and musical activities, through which I made life-long friends to emotionally support me. I was also fortunate enough to study with a Hungarian Violinist János Négyesy, who was the first one to recognize my lack of confidence. As a solution, he told me to think big and play big. Whenever I encountered discouraging moments in graduate school, I reminded myself to think big and study big. Thank you for your violin and life lessons, and rest in true peace, János. Another source of support was Professor Tara Javidi, who strongly encouraged me to pursue doctoral degree. Her recommendation brought me to my current advisor in Illinois.

Professor Olgica Milenkovic welcomed me with interesting problems in bioinformatics, and I thank her for this new direction. Among many qualities, I truly admire her creativity and intellectual curiosity. My co-advisor Professor Jun S. Song is also a great role model. I respect his dignity and meticulousness, and thank him for training me. I would like to call them my academic parents, who had higher expectations for me than my biological parents had. Through their investment and belief in my intellectual growth, I finally felt worthy. Thus, I dedicate this humble thesis to my beloved academic parents.
# TABLE OF CONTENTS

CHAPTER 1  INTRODUCTION .................................................. 1  
  1.1  Introduction to Molecular and Cellular Biology ................. 1  
  1.2  Genomics .............................................................. 2  
  1.3  Sequencing Technologies ............................................ 4  
  1.4  Thesis Overview ..................................................... 6  

CHAPTER 2  METAGENOMIC READ PROCESSING AND COMPRESSION ........ 8  
  2.1  Background ............................................................ 8  
  2.2  Methods ............................................................... 11  
  2.3  Results ................................................................. 20  
  2.4  Discussion ............................................................. 28  

CHAPTER 3  QUANTITATIVE ANALYSIS AND PREDICTION OF G-QUADRUPLEX FOLDING PROPENSITY ....................... 30  
  3.1  Background ............................................................ 30  
  3.2  Methods ............................................................... 33  
  3.3  Results ................................................................. 39  
  3.4  Discussion ............................................................. 53  

CHAPTER 4  DATA STRUCTURES TO ANALYZE AND STORE 3D GENOME MAPS ....................................................... 57  
  4.1  Background ............................................................ 57  
  4.2  Methods ............................................................... 60  
  4.3  Results ................................................................. 65  
  4.4  Discussion ............................................................. 66  

REFERENCES ................................................................. 71
CHAPTER 1

INTRODUCTION

1.1 Introduction to Molecular and Cellular Biology

Before delving into genomics, we first review fundamental concepts in molecular and cellular biology, summarized from parts of Cooper and Hausman [1].

A well-established cell theory in molecular biology states that: 1) all living organisms are composed of one or more cells, 2) the cell is the most basic unit of life, and 3) all cells arise from pre-existing cells. This theory highlights the importance of studying cells in living organisms.

The information coding for cell functions is contained in deoxyribonucleic acid (DNA). DNA is composed of four nucleotides: cytosine (C), thymine (T), adenine (A), and guanine (G), where C and G form a pair through three hydrogen bonds and A and T form a pair through two hydrogen bonds (Figure 1.1). It is worth noting that an extra hydrogen bond between C and G makes it a stronger pair than A and T. These complementary base pairings allows DNA not only to maintain a double-stranded helical structure but also to replicate itself. When the double-stranded DNA (dsDNA) is denatured, i.e., hydrogen bonds are broken, an enzyme may use one strand as a template and add complementary pairs. This process, called DNA replication, produces an exact copy of the original dsDNA.

DNA molecules may be copied into another similar macromolecule called ribonucleic acid (RNA). RNA, composed of thymine (T), uracil (U), adenine (A) and guanine (G), lacks a hydroxyl group and is usually single-stranded. The process of copying the DNA to RNA is called transcription.

A triplet of single-stranded RNA codes for one of 20 amino acids. The amino acids ultimately build proteins, which perform major functions in our cells. This ultimate process of converting RNA to protein is called translation. The central dogma of molecular biology, encompassing replication, transcrip-
Figure 1.1: Complementary base pairing of right-handed double helical DNA strands. Adenine and thymine are paired through two hydrogen bonds, and cytosine and guanine are paired by three hydrogen bonds. Major grooves are wider than minor grooves. Illustration by Zephyris (Own work) [CC BY-SA 3.0 or GFDL], via Wikimedia Commons.

In Section 1.1, we learned that cells’ functions are largely governed by protein, but the information itself is encoded in the DNA. Thus, we must first look at the genome of an organism, the DNA content in the cell. For example, the human genome consists of approximately 3 billion DNA base pairs and has been identified in the early 2000s.

Genomics is the study of genomes. A sequence of DNA encodes a specific protein through transcription and translation, but the functions of each part of the genome have not been fully understood. Furthermore, the amount of protein is often controlled by unknown external factors. One of the goals of genomics is to catalog functional elements of the genome and to understand how the genome is organized to fulfill these functions.

Storing all 3 billion base pairs in a small cell requires efficient hierarchical structures (Figure 1.3). The naked DNA strands are first coiled and
Figure 1.2: Central dogma of molecular biology: a flow from DNA to protein. DNA polymerase replicates the DNA, RNA polymerase transcribes the DNA to RNA, and the ribosome translates RNA to protein products. Illustration by Central Dogma of Molecular Biochemistry with Enzymes.jpg: Dhorspool derivative work: Miguelferig [CC BY-SA 3.0], via Wikimedia Commons.

wrapped around positively charged histone core proteins and form nucleosomes, which look like the “beads” on a string of DNA strands. Addition of the linker histone H1 further compresses to the 30 nm fibers and finally forms a chromosome. The center of the chromosome is called the centromere, and the telomere at the end protects the chromosome.

The central dogma explained in this section relies on the accessibility of machineries to the naked DNA. Both replication and transcription require the addition of complementary nucleotide bases to the template strand. Consequently, parts of the tightly packed genomes have to be unwound for replication and transcription. Epigenomics is a subarea of genomics in which the goal is to understand the mechanisms behind chromatin structures and modifications and their roles in gene regulations. We will revisit this concept in Chapter 4.
Hierarchical organization of chromatin structures. A double-helical DNA is wrapped around histone proteins to form a nucleosome. The sequence of more than one nucleosome resembles beads on a string, and is further compressed into 30 nm fiber. Chromosome represents the most condensed form of our genome. Illustration by Richard Wheeler at en.wikipedia [GFDL or CC-BY-SA-3.0], from Wikimedia Commons.

1.3 Sequencing Technologies

Sequencing refers to the precise reading of the genome. One of the earliest techniques, Sanger sequencing, is the chain-termination method developed by Frederick Sanger in the 1970s (Figure 1.4). Here, each of the four dideoxynucleotides (dNTPs) ddTTP, ddCTP, ddATP, and ddGTP is labeled with red, blue, green and pink fluorescent dye, respectively, and indicates the randomly terminated position of the genome. As a result, we obtain the genome by reading the colors at each position. The Sanger method is accurate, but the process is often laborious especially if we want to sequence a large genome.

There have been improvements to the Sanger and related methods, but a major breakthrough in molecular biology occurred with the invention of high-throughput (formerly next-generation) sequencing. Because it is difficult to read long genomes at once, the technique fragments the genome into smaller pieces. Combined with an amplification step via a polymerase chain reaction (PCR), we obtain many short segments of the genome called the reads. Moreover, paired-end sequencing has been developed to increase the accuracy: a single short read may have come from multiple parts of the genome, but if another pair further away is sequenced, we can identify the precise location by requiring that both pairs have to match the reference genome. In Figure 1.5, the reads are placed such that the sequences of red parts overlap. The genome is then assembled by connecting overlapping parts of the reads,
Figure 1.4: The Sanger method for DNA sequencing. Each of the four dideoxynucleotides ddTTP, ddCTP, ddATP, ddGTP is labeled with red, blue, green and pink fluorescence dye to indicate nucleotide base at a terminating location. With the primer, DNA template, DNA polymerase and dNTPs in mixture, the primer elongates and the chain terminates. By reading the population of labeled strands, we obtain a chromatograph and recover the genome sequence. Illustration by Estevezj (Own work) [CC BY-SA 3.0], via Wikimedia Commons.

and many algorithms have been developed to accurately assemble a genome. High-throughput sequencing technique has been applied to diverse domains in genomics. For example, it can sequence the DNA (whole genome sequencing) and RNA (RNA-seq), target parts of DNA occupied by a specific protein (ChIP-seq), identify modified parts of the chromatin (WGBS), find open chromatin regions (DNase-seq and ATAC-seq), and most recently, obtain genome-wide contact maps (5C, Hi-C, and ChIA-PET). Some of these domains are explored in subsequent chapters.
1.4 Thesis Overview

As seen in the previous section, advanced high-throughput sequencing technologies allow us to access a large amount of genomic data and we can use it to study cancer, evolution, development, and much more. Unfortunately, we face the challenge of processing and storing a large volume of genomic data because sequencing technologies are advancing at a faster rate than computation capabilities. Moreover, the analysis of a large dataset may be both computationally and theoretically challenging because statistical methods have not been developed for new emerging data. In this work, I address some of these problems using tools from information theory and machine learning.

First, I focus on the data processing and storage aspect of the problem. Due to the low cost of sequencing today, at about $1000 to sequence a human genome, researchers generate a massive amount of data. We currently store more than 100 petabytes of sequencing data, and by 2025 we are projected to have 2-40 exabytes of data [2]. There are several options to mitigate this problem. One option is to discard the data after the project has been
completed. However, some biological data may not be easily reproducible, and we may need the data in the future for a related project. A simple solution is to buy more hard drives, but 40 exabytes of hard disks would be extremely costly. The last option, which is pursued by scientists and engineers, is to compress the data.

Data compression, also known as source coding, is a subfield of information and coding theory in which various dictionary-based (Lempel-Ziv) to entropy-based (Huffman) methods have been developed. These encoding algorithms have been widely adopted to compress general text files, such as in gzip and bzip. However, these general tools are not suitable for compressing genomic data because we can get a better rate by exploiting certain properties of a particular dataset. Previously, compression tools for FASTA, FASTQ, and BAM/SAM files have been developed, but no known algorithm existed for metagenomic data. Thus, I present a statistical and information theoretical approach to compress metagenomic reads.

Second, I analyze biological data to extract meaningful information by using regression methods, which are widely used in the machine learning community. Regression analysis is a statistical process for estimating the relationships among variables, and it is different from a classification problem in that we focus on continuous outcome rather than discrete classes. In particular, I use linear regression and Gaussian process regression methods to predict the ability of a DNA sequence to fold into a secondary structure called “G-quadruplex”.

Last, I propose several data formats to store 3D genome contact maps. Recent technologies such as Hi-C, ChIA-PET, and their variants provide pairwise genome interactions information. The data are generally stored as a two-dimensional matrix, but a long genome at a fine resolution produces an extremely large matrix. We can exploit statistical properties of the contact maps, such as symmetry and sparsity in some cases, to efficiently store large data. A natural extension of this work would be to compress the data of the optimal format.

The thesis is organized as follows: in Chapter 2, I present MetaCRAM, a compression tool for metagenomic sequencing reads; in Chapter 3, the regression models are designed to predict G-quadruplex forming sequences with high accuracy; and I propose 3D genome data formats in Chapter 4.
CHAPTER 2

METAGENOMIC READ PROCESSING
AND COMPRESSION

2.1 Background

2.1.1 Introduction to Metagenomics

Metagenomics is an emerging discipline focused on genomic studies of complex microorganisal population. In particular, metagenomics enables a range of analyses pertaining to species composition, the properties of the species and their genes as well as their influence on the host organism or the environment. As the interactions between microbial populations and their hosts plays an important role in the development and functionality of the host, metagenomics is becoming an increasingly important research area in biology, environmental and medical sciences. As an example, the National Institute of Health (NIH) recently initiated a far-reaching Human Microbiome Project [3] which has the aim to identify species living at different sites of the human body (in particular, the gut and skin[4]), observe their roles in regulating metabolism and digestion, and evaluate their influence on the immune system. The findings of such studies may have important impacts on our understanding of the influence of microbials on an individual’s health and disease, and hence aid in developing personalized medicine approaches. Another example is the Sorcerer II Global Ocean Sampling Expedition [5], led by the Craig Venter Institute, the purpose of which is to study microorganisms that live in the ocean and influence/maintain the fragile equilibrium of this ecosystem.

There are many challenges in metagenomic data analysis. Unlike classical genomic samples, metagenomic samples comprise many diverse organisms, the majority of which is usually unknown. Furthermore, due to low sequencing depth, most widely used assembly methods – in particular, those based
on de Bruijn graphs – often fail to produce quality results and it remains a challenge to develop accurate and sensitive meta-assemblers. These and other issues are further exacerbated by the very large file size of the samples and their ever increasing number. Nevertheless, many algorithmic methods have been developed to facilitate some aspects of microbial population analysis: examples include MEGAN (MEta Genome ANalyzer) [6], a widely used tool that allows for an integrative analysis of metagenomic, metatranscriptomic, metaproteomic, and rRNA data; and PICRUSt (Phylogenetic Investigation of Communities by Reconstruction of Unobserved States) [7], developed to predict metagenome functional contents from 16S rRNA marker gene sequences. Although suitable for taxonomic and functional analysis of data, neither MEGAN nor PICRUSt involve a data compression component, as is to be expected from highly specialized analytic software.

2.1.2 Literature review on genomic data compression

In parallel, a wide range of software solutions have been developed to efficiently compress classical genomic data (a comprehensive survey of the state-of-the-art techniques may be found in [8]). Specialized methods for compressing whole genomes have been reported in [9, 10, 11], building upon methods such as modified Lempel-Ziv encoding and the Burrows-Wheeler transform. Compression of reads is achieved by mapping the reads to reference genomes and encoding only the differences between the reference and the read; or, in a de novo fashion that does not rely on references and uses classical sequence compression methods. Quip [12] and CRAM [13] are two of the best known reference-based compression algorithms, whereas ReCoil [14], SCALCE [15], MFCompress [16], and the NCBI Sequence Read Archive method compress data without the use of reference genomes. Reference-based algorithms in general achieve better compression ratios than reference-free algorithms by exploiting the similarity between some predetermined reference and the newly sequenced reads. Unfortunately, none of the current reference-based method can be successfully applied to metagenomic data, due to the inherent lack of “good” or known reference genomes. Hence, the only means for compressing metagenomic FASTA and FASTQ files is through the use of de novo compression methods.
2.1.3 Highlights of MetaCRAM

As a solution to the metagenomic big data problem, we introduce MetaCRAM, the first de novo, parallel, CRAM-like software specialized for FASTA-format metagenomic read compression, which in addition provides taxonomy identification, alignment and assembly information. This information primarily facilitates compression, but also allows for fast searching of the data in the compressive domain and for basic metagenomic analysis. The gist of the classification method is to use a taxonomy identification tool – in this case, Kraken [17] – which can accurately identify a sufficiently large number of organisms from a metagenomic mix. By aligning the reads to the identified reference genomes of organisms via Bowtie2 [18], one can perform efficient lossless reference-based compression via the CRAM suite. Those reads not aligned to any of the references can be assembled into contigs through existing metagenome assembly software algorithms, such as Velvet [19] or IDBA-UD [20]; sufficiently long contigs can subsequently be used to identify additional references through BLAST (Basic Local Alignment Search Tool) [21]. The reads aligned to references are compressed into the standard CRAM format [13], using three different integer encoding methods, Huffman [22], Golomb [23], and Extended Golomb encoding [24].

MetaCRAM is an automated software with many options that accommodate different user preferences, and it is compatible with the standard CRAM and SAMtools data format. In addition, its default operational mode is lossless, although additional savings are possible if one opts for discarding read ID information. We report on both the lossless and “lossy” techniques in the Methods Section. MetaCRAM also separates the read compression process from the quality score compression technique, as the former technique is by now well understood while the latter is subject to constant changes due to different quality score formats in sequencing technologies. These changes may be attributed to increasing qualities of reads and changes in the correlations of the score values which depend on the sequencing platform. For quality score compression, the recommended method is QualComp [25].

MetaCRAM offers significant compression ratio improvements when compared to standard bzip and gzip methods, and methods that directly compress raw reads. These improvements range from 2-4-fold file size reductions, which leads to large storage cost reductions. Furthermore, although
MetaCRAM has a relatively long compression phase, decompression may be performed in a matter of minutes. This makes the method suitable for both real time and archival applications.

2.2 Methods

2.2.1 Algorithmic overview

The block diagram of the MetaCRAM algorithm is given in Figure 2.1, and the operation of the algorithm may be succinctly explained as follows. The first step is to identify suitable references for compression, which is achieved by identifying dominant taxonomies in the sample. The number of references is chosen based on cut-off abundance thresholds, which themselves are chosen using several criteria that trade-off compression ratio and compression time. Once the references are chosen, the raw reads are aligned to their closest references and the starting positions of the reads are statistically analyzed to determine the best integer compression method to be used for their encoding. Furthermore, reads that do not align sufficiently well with any of the chosen references are assembled using IDBA_UD, and the contig outputs of the assembler are used to identify additional references via BLAST search. Reads not matched with any references after multiple iterations of the above procedure are compressed independently with the MFCompress suite. The results associated with each of the described processing stages are discussed in the next subsections. Note that here and throughout the paper, we use standard terms in genomics and bioinformatics without explanations.

2.2.2 Pre-processing

MetaCRAM accepts both unpaired and paired-end reads. If paired-end reads are given as an input to MetaCRAM, then the first preprocessing step is to append the read IDs with a “1” or a “2” indicating that the read came from the first or second mate, respectively. Another preprocessing step includes filtering out the quality scores in case that the input file is in FASTQ format. This filtering process allows for using new and emerging quality score compression methods without constantly updating the MetaCRAM platform.
Note that the paired end labeling is done automatically, while filtering can be implemented outside the integrated pipeline by the user, based on his/her requirements for quality score lossy or lossless compression goals.

MetaCRAM uses as a default FASTA files that do not contain quality values, in which case the resulting SAM file contains the symbol “I” repeated as many times as the length of the sequence. These symbols amount to about 100 bytes per read, and this overhead increases proportionally to the number of reads. In order to reduce the size of this unnecessary field, MetaCRAM replaces the sequence of “I”s with a single symbol “*”, complying with the standard SAM format. Likewise, read IDs are highly repetitive in nature: for instance, every read ID starts with the data name such as “SRR359032.”, followed by its unique read number. Rather than repeating the data name for every read, we simply store it once, and append it when performing decompression. Both versions of MetaCRAM – one incorporating these two options – and another one without the described features are available to the user. The former version of the methods requires a slightly longer compression and decompression time.

2.2.3 Taxonomy identification

Given the labeled read sequences of a metagenomic sample, the first step is to identify the mixture of species present in the sample. There are several taxonomy identification methods currently in use: the authors of [26] propose to use the 16S rRNA regions for bacterial genome identification, MetaPhyler [27] scans for unique markers exceeding length 20 and provides a taxonomy level as specific as the genus. On the other hand, a new taxonomy identification software known as Kraken [17], based on exact alignment of $k$-mers to the database of known species, often outperforms MetaPhyler and other methods both in terms of speed and discovery of true positives, as indicated by our tests.

MetaCRAM employs Kraken as a default tool in the pipeline. Kraken produces an output report which is automatically processed by MetaCRAM. Part of the report contains information about species present in the sample, as well as their abundance. We rank order the species from the most abundant to the least abundant, where abundance is based on the number of
reads identified to match a species in the database. For downstream analysis, MetaCRAM selects the “most relevant” species and uses their genomes as references. The default definition of “most relevant” is the top 75 species, but one has the option to choose a threshold for the abundance value or for the number of references used. As an illustration, Table 2.2 (p. 22) lists the results of an analysis of the impact of different thresholds on the processing time and the compression ratio.

### 2.2.4 Alignment and assembly

After a group of reference genomes is carefully chosen based on the Kraken software output, alignment of reads to the reference genomes is performed. This task is accomplished by using Bowtie2 [18], a standard software tool for ultra-fast alignment of short reads to long genomes. The alignment information is stored in a SAM (Sequence Alignment/Map) file format and subsequently used for compression via reference-based algorithms.

Due to the fact that many species in a metagenome sample have never been sequenced before, some reads will not be aligned to any of the references, and we collectively refer to them as *unaligned* reads hereafter. In order to discover reference genomes for unaligned reads, we assemble the unaligned reads using a metagenomic assembler. Our metagenomic assembler of choice is IDBA-UD [20], given that in our tests it produced the largest number of contigs leading to new reference identification. Alternatives to IDBA-UD include the Ray Meta software [28].

When the reads have high sequencing depth and large overlaps, the *contigs* produced by the assembler may be queried using BLAST [21] to identify the organisms they most likely originated from. The user may choose to BLAST only the top $n$ longest contigs, where $n$ is a user specified number, but in our analysis we use all contigs. Subsequently, we align the unaligned reads to the newly found references.

### 2.2.5 Distribution of read starting positions

We empirically studied the distribution of integers representing the read positions, variation positions, and paired-end offsets in order to choose the most
Figure 2.1: The block diagram of the MetaCRAM Algorithm for Metagenomic Data Processing and Compression. Its main components are taxonomy identification, alignment, assembly and compression.

suitable compression method. As an example, the distribution of the starting positions for the reads that aligned to JH603150 (genome of Klebsiella oxytoca) in the dataset SRR359032 is shown in Figure 2.2. This distribution was truncated after achieving a 90% coverage of the data (i.e., after only 10% of the read start positions exceeded the depicted maximum length). The empirical distribution is shown in yellow, while a fitted power law distributions is plotted and determined according to [24], with \( P_i = 2^{-\log_m i \left( \frac{1}{2(i(m-1))} \right) } \), where \( i \) is the integer to be encoded, and \( m \) is the divisor in the extended Golomb code. The chosen parameters are \( m = 3 \) and 4. The negative binomial distribution is fitted using Maximum Likelihood Estimation (MLE), while the geometric distribution is fitted by two different means: using MLE and ezfit, a MATLAB script that performs an unconstrained nonlinear minimization of the sum of squared residuals with respect to various parameters.

For single reference alignment methods, it was reported that the best fit for the empirical distribution is a geometric distribution or a negative binomial distribution [29]. However, due to sequencing errors and non-uniform distributions of hydrogen bond breakage (also referred as the “GC bias”), the empirical data often deviates from geometric and negative binomial dis-
tributions [30]. In addition, for metagenomic samples, there exist multiple references which may have good alignments with reads that did not originally correspond to the genomic sample of the reference. This creates additional changes in the read starting position with respect to the geometric distribution. Moreover, one has to encode not only the read positions but also the variation positions and paired-end offsets, making it difficult to claim any one of the fitted distributions is better than others. This observation is supported by Figure 2.2. Since there is no known efficient optimal encoding method for a set of integers with negative binomial distributions, and Golomb and extended Golomb encoding are optimal for geometric distributions and power law distributions, respectively, we use these two methods with $m = 3$. The parameter $m$ is chosen based on extensive experiments, although the user has the freedom to adjust and modify its value.

As the number of unaligned reads that remains after a few iterations of MetaCRAM is relatively small, these reads were compressed using a reference-free tool such as MFCompress [16], which is based on finite-context models. Furthermore, the SAM files produced after running Bowtie2 are converted to the sorted and indexed binary format of a BAM file using SAMtools [31]. Each BAM file is compressed via reference-based compression against its representative to a standard CRAM format. We tested three different modes of the CRAM toolkit [13]: Huffman, Golomb, and Extended Golomb encoding,
all of which are described in the next section. Note that the Extended Golomb encoding method is our new addition to the classical CRAM method, as it appears to offer good compromises between compression and decompression speed and compression ratios.

Intrinsically, SAM files contain quality values and unique read IDs for each read, which inevitably account for a large file size: quality values are characters of length as long as the sequence, and read IDs often repeat the name of the dataset. By default, MetaCRAM preserves all quality values and read IDs as designed in CRAM.

2.2.6 Compression

Compression in the reference-based mode is accomplished by compressing the starting points of references with respect to the reference genomes and the base differences between the reads and references. As both the starting points and bases belong to a finite integer alphabet, we used three different integer compression methods, briefly described below.

Huffman coding is a prefix-free variable length compression method for known distributions [22] which is information-theoretically optimal [32]. The idea is to encode more frequent symbols with fewer bits than non-frequent ones. For example, given an alphabet \( A = \{a, b, c, d, e\} \) and the corresponding distribution \( P = (0.25, 0.25, 0.2, 0.15, 0.15) \), building a Huffman tree results in the codebook \( C = (00, 10, 11, 010, 011) \) (Figure 2.3). Decoding relies on the Huffman tree constructed during encoding which is stored in an efficient manner, usually ordered according to the frequency of the symbol. Due to the prefix-free property, Huffman coding is uniquely decodable and does not require any special marker between words. Two drawbacks of Huffman coding are its storage complexity, since we need to record large tree structures for big alphabet size, and the need to know the underlying distribution a priori. Adaptive Huffman coding mitigates the second problem, at the cost of increased computational complexity associated with constructing multiple encoding trees [33]. In order to alleviate computational challenges, we implemented so called canonical Huffman encoding, which bypasses the problem of storing a large code tree by sequentially encoding lengths of the codes [34].

Golomb codes are optimal prefix-free codes for countably infinite lists of
non-negative integers following a geometric distribution [23]. In Golomb coding, one encodes an integer \( n \) in two parts, using its quotient \( q \) and remainder \( r \) with respect to the divisor \( m \). The quotient is encoded in unary, while the remainder is encoded via truncated binary encoding. Given a list of integers following a geometric distribution with known mean \( \mu \), the dividend \( m \) can be optimized to reduce code length. In [35], the optimal value of \( m \) was derived for \( m = 2^k \), for any integer \( k \). The encoding is known as the Golomb-Rice procedure, and it proceeds as follows: first, we let \( k^* \) = \( \max \left\{ 0, 1 + \left\lfloor \log_2 \left( \frac{\log(\phi - 1)}{\log(\mu + 1)} \right) \right\rfloor \} \), where \( \phi = \frac{\sqrt{5} + 1}{2} \). Unary coding represents an integer \( i \) through runs of \( i \) ones followed by a single zero. For example, the integer \( i = 4 \) in unary is 11110. Truncated binary encoding is a prefix-free code for an alphabet of size \( m \), which is more efficient than standard binary encoding. Because the remainder \( r \) can only take values in \( \{0,1,\ldots, m-1\} \), according to truncated binary encoding, we assign to the first \( 2^{k+1} - m \) symbols codewords of fixed length \( k \). The remaining symbols are encoded via codewords of length \( k + 1 \), where \( k = \lfloor \log_2(m) \rfloor \). For instance, given \( n = 7 \) and \( m = 3 \), we have \( 7 = 2 \times 3 + 1 \), implying \( q = 2 \) and \( r = 1 \). Encoding 2 in unary gives 110 and 1 in truncated binary reads as 10. Hence, the codeword used to encode the initial integer is the concatenation of the two representations, namely 11010.

Decoding of Golomb encoded codewords is also decoupled into decoding of the quotient and the remainder. Given a codeword, the number of ones before the first zero determines the quotient \( q \), while the remaining \( k \) or \( k + 1 \) bits, represents the remainder \( r \) according to truncated binary decoding for an alphabet of size \( m \). The integer \( n \) is obtained as \( n = q \times m + r \).
Golomb encoding is computationally more efficient than Huffman coding because it only requires division operations. Furthermore, one does not need to the distribution \textit{a priori}, although there are clearly no guarantees that Golomb coding for an unknown distribution will be even near-optimal: Golomb encoding is optimal only for integers following a geometric distribution.

An extension of Golomb encoding, termed \textit{extended} Golomb [24] coding, is an iterative method for encoding non-negative integers following a power law distribution. One divides an integer $n$ by $m$ until the quotient becomes 0, and then encodes the number of iterations $M$ in unary, and an array of remainders $r$ according to an encoding table. This method has an advantage over Golomb coding when encoding large integers, such is the case for read position compression. As an example, consider the integer $n = 1000$: with $m = 2$, Golomb coding would produce $q = 500$ and $r = 0$, and unary encoding of 500 requires 501 bits. With extended Golomb coding, the number of iterations equals $M = 9$ and encoding requires only 10 bits. As an illustration, let us encode $n = 7$ given $m = 3$. In the first iteration, $7 = 2 \times 3 + 1$, so $r_1 = 1$ is encoded as 10, and $q_1 = 2$. Since the quotient is not 0, we iterate the process: $2 = 0 \times 3 + 2$ implies $r_2 = 2$, which is encoded as 1, and $q_2 = 0$. Because the quotient is at this step 0, we encode $M = 2$ as 110 and $r = r_2r_1 = 110$, and our codeword is 110110.

The decoding of extended Golomb code is also performed in $M$ iterations. Since we have a remainder stored at each iteration and the last quotient $q_M = 0$, it is possible to reconstruct the original integer. Similar to Golomb coding, extended Golomb encoding is computationally efficient, but optimal only for integers with power law distributions.

There are various other methods for integer encoding, such as Elias Gamma and Delta Encoding [36], which are not pursued in this paper because they do not appear to offer good performance for the empirical distributions observed in our read position encoding experiments.

\section*{2.2.7 Products}

The compressed unaligned reads, CRAM files, list of reference genomes (optional), alignment rate (optional), contig files (optional) are all packaged
into an archive. The resulting archive can be stored in a distributed manner and when desired, the reads can be losslessly reconstructed via the CRAM toolkit.

2.2.8 Decompression

Lossless reconstruction of the reads from the compressed archive is done in two steps. For those reads with known references in CRAM format, decompression is performed with an appropriate integer decompression algorithm. When the files are converted back into the SAM format, we retrieve only the two necessary fields for FASTA format, i.e., the read IDs and the sequences printed in separate lines. Unaligned reads are decompressed separately, through the decoding methods used in MFCompress.

2.2.9 Post-processing

The two parts of reads are now combined into one file, and they are sorted by the read IDs in an ascending order. If the reads were paired-end, they are separated into two files according to the mate “flag” assigned in the pre-processing step.

2.2.10 Effects of parallelization

One key innovation in the implementation of MetaCRAM is parallelization of the process, which was inspired by parallel single genome assembly used in TIGER [37]. Given that metagenomic assembly is computationally highly demanding, and in order to fully utilize the computing power of a standard desktop, MetaCRAM performs meta assembly of unaligned reads and compression of aligned reads in parallel. As shown in Table 2.1, parallelization improves real, user, and system time by 23 – 40 %.

2.2.11 Test datasets

The datasets supporting the results of this article are available in the National Center for Biotechnology Information Sequence Read Archive repository, un-
Table 2.1: Processing time improvements for two rounds of MetaCRAM on the SRR359032 dataset (5.4GB, without removing redundancy in description lines) resulting from parallelization of assembly and compression.

<table>
<thead>
<tr>
<th>Time</th>
<th>Without Parallelization</th>
<th>With Parallelization</th>
<th>Reduction (% )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real</td>
<td>235m 40s</td>
<td>170m 4s</td>
<td>27.7</td>
</tr>
<tr>
<td>User</td>
<td>449m 40s</td>
<td>346m 33s</td>
<td>22.9</td>
</tr>
<tr>
<td>System</td>
<td>14m 13s</td>
<td>8m 45s</td>
<td>40.1</td>
</tr>
</tbody>
</table>


2.3 Results

We tested MetaCRAM as a stand-alone platform and compared it to MF-Compress, a recently developed software suite specialized for FASTA files, and bzip2 and gzip [38], standard general purpose compression tools (available at http://www.bzip.org). Other software tools for compression of sequencing data such as SCALCE and Quip, and SAMZIP [39] and SlimGene [40], were not tested because they were either for FASTQ or SAM file formats, and not FASTA files.

As already pointed out, MetaCRAM does not directly process FASTQ file formats for multiple reasons: 1) the quality of sequencers are improving significantly, reaching the point where quality scores may contain very little information actually used during analysis; 2) reads with low quality scores are usually discarded and not included in metagenomics analysis – only high quality sequences are kept; 3) there exist software tools such as QualComp [25], specifically designed for compressing quality scores that users can run independently along with MetaCRAM.
2.3.1 Taxonomy identification and reference genome selection

As the first step of our analysis, we compared two metagenomic taxonomy identification programs, Kraken and MetaPhyler in terms of computation time and identification accuracy on synthetic data, as it is impossible to test the accuracy of taxonomy identification on real biological datasets. For this purpose, we created mixtures of reads from 15 species, listed in the Additional File 4. The two Illumina paired-end read files were created by MetaSim [41] with 1% error rate, and they amounted to a file of size 6.7 GB. Kraken finished its processing task in 22 minutes and successfully identified all species within the top 50 most abundant taxons. On the other hand, MetaPhyler ran for 182 minutes and failed to identify Acetobacterium woodii and Haloterrigena turkmenica at the genus level. This example illustrates a general trend in our comparative findings, and we therefore adopted Kraken as a default taxonomy retrieval tool for MetaCRAM.

When deciding how to choose references for compression, one of the key questions is to decide which outputs of the Kraken taxonomy identification tool are relevant. Recall that Kraken reports the species identified according to the number of reads matched to their genomes. The most logical approach to this problem is hence to choose a threshold for the abundance values of reads representing different bacterial species, and only use sequences of species with high abundance as compression references. Unfortunately, the choice for the optimal threshold value is unclear and it may differ from one dataset to another; at the same time, the threshold is a key parameter that determines the overall compression ratio – choosing too few references may lead to poor compression due to the lack of quality alignments, while choosing too many references may reduce the compression ratio due to the existence of many pointers to the reference files. In addition, if we allow too many references, we sacrifice computation time for the same final alignment rate. It is therefore important to test the impact of the threshold choice on the resulting number of selected reference genomes.

In Table 2.2, we listed our comparison results for all five datasets studied, using two threshold values: 75 (high) and 10 (low). For these two choices, the results are colored gray and white, respectively. We observe that we get slightly worse compression ratios if we select too few references, as may be seen for the files ERR321482 and ERR532393. Still, the processing time is
significantly smaller when using fewer references, leading to 30 to 80 minutes of savings in real time. It is worth to point out that this result may also be due to the different qualities of internal hard drives: for example, the columns in gray were obtained running the code on Seagate Barracuda ST3000, while the results listed in white were obtained via testing on Western Digital NAS.

Table 2.2: Analysis of the influence of different threshold values on reference genome selection after taxonomy identification and compression ratios. Columns colored in gray represent a threshold of 75 species, while the columns not colored in gray correspond to a cutoff of 10 species. The results are shown for MetaCRAM-Huffman, with original and compressed file sizes in MB and processing time in minutes. “Aln. %” refers to the alignment rates for the first and second round, and “No. files” refers to the number of reference genome files selected in the first and second iteration.

<table>
<thead>
<tr>
<th>Data</th>
<th>Ori. (MB)</th>
<th>Com. (MB)</th>
<th>Time (min)</th>
<th>Aln. %</th>
<th>No. files</th>
<th>Com. (MB)</th>
<th>Time (min)</th>
<th>Aln. %</th>
<th>No. files</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR321482</td>
<td>1429</td>
<td>191</td>
<td>299m</td>
<td>27.0</td>
<td>211</td>
<td>193</td>
<td>239m</td>
<td>24.2</td>
<td>29</td>
</tr>
<tr>
<td>SRR359032</td>
<td>3981</td>
<td>319</td>
<td>127m</td>
<td>57.7</td>
<td>26</td>
<td>320</td>
<td>93m</td>
<td>57.7</td>
<td>7</td>
</tr>
<tr>
<td>ERR532393</td>
<td>8230</td>
<td>948</td>
<td>639m</td>
<td>45.8</td>
<td>267</td>
<td>963</td>
<td>522m</td>
<td>42.5</td>
<td>39</td>
</tr>
<tr>
<td>SRR1450398</td>
<td>5399</td>
<td>703</td>
<td>440m</td>
<td>7.1</td>
<td>190</td>
<td>703</td>
<td>364m</td>
<td>6.8</td>
<td>26</td>
</tr>
<tr>
<td>SRR062462</td>
<td>6478</td>
<td>137</td>
<td>217m</td>
<td>2.6</td>
<td>278</td>
<td>139</td>
<td>197m</td>
<td>2.1</td>
<td>50</td>
</tr>
</tbody>
</table>

Many of the most abundant references may be from the same genus, and this may potentially lead to the problem of multiple alignment due to subspecies redundancy. The almost negligible effect of the number of reference genomes on alignment rate implies that combining them to remove the redundancy would improve computational efficiency, as suggested in [42]. Nevertheless, extensive computer simulations reveal that the loss due to multiple alignment is negligible whenever we choose up to 75-100 references. Therefore, our recommendation is to use, as a rule of thumb, the threshold 75 in order to achieve the best possible compression ratio and at the same time provide a more complete list of genomic references for further analysis.
2.3.2 Compression performance analysis

Our comparison criteria include the compression ratio (i.e., the ratio of the uncompressed file and the compressed file size), as well as the compression and decompression time, as measured on an affordable general purpose computing platform: Intel Core i5-3470 CPU at 3.2 GHz, with a 16 GB RAM. We present test results for five datasets: ERR321482, SRR359032, ERR532393, SRR1450398, and SRR062462, including metagenomic samples as diverse as a human gut microbiome or a Richmond Mine biofilm sample, retrieved from the NCBI Sequence Read Archive [43].

The comparison results of compression ratios among six software suites are given in Table 2.3 and Figure 2.4. The methods compared include three different modes of MetaCRAM, termed Huffman, Golomb and Extended Golomb MetaCRAM.

The result indicates that MetaCRAM using Huffman integer encoding method improves compression ratios of the classical gzip algorithm 2 – 3-fold on average. For example, MetaCRAM reduces the file size of SRR062462 to only 2% of the original file size. Notably, the users have the options to retrieve the alignment rate, list of reference genomes, contig files, and alignment information in SAM format. This list may be stored with very small storage overhead and then used for quick identification of files based on their taxonomic content, which allows for selection in the compressive domain.

In the listed results, the column named “Qual Value (MB)” provides the estimated size of the quality scores for each file, after alignment to references found by Kraken. In our implementation, we replaced these scores with a single “*” symbol per read and also removed the redundancy in read IDs. The result shows that these two options provide better ratios than the default ratio, as shown in Table 2.3 column “MCH2”. However, since read IDs may be needed for analysis of some dataset, we also report results for the default “MCH1” mode which does not dispose of ID tags.

In terms of the processing time shown in Table 2.4 and Figure 2.5, the MetaCRAM suite is at a clear disadvantage, with processing time 150-fold slower than bzip2 in the worst case. Figure 2.6 presents the average runtime of each stage for all five datasets tested, and illustrates that assembly, alignment, and BLAST search are computationally demanding, accounting for 62 percentage of the total time. This implies that removing the second
Table 2.3: Comparison of compression ratios of six software suites. For short hand notation, we used “MCH” = MetaCRAM-Huffman, “MCG” = MetaCRAM-Golomb, “MCEG” = MetaCRAM-extended Golomb, “MFComp” = MFCompress. “Align. %” refers to the total alignment rates from the first and second iteration. Minimum compressed file size achievable by the methods are written in bold case letters.

<table>
<thead>
<tr>
<th>Data</th>
<th>Original (MB)</th>
<th>MCH (MB)</th>
<th>MCG (MB)</th>
<th>MCEG (MB)</th>
<th>Align. %</th>
<th>bzip2 (MB)</th>
<th>gzip (MB)</th>
<th>MFComp (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR321482</td>
<td>1429</td>
<td>191</td>
<td>312</td>
<td>213</td>
<td>29.6</td>
<td>362</td>
<td>408</td>
<td>229</td>
</tr>
<tr>
<td>SRR359032</td>
<td>3981</td>
<td>319</td>
<td>657</td>
<td>458</td>
<td>61.8</td>
<td>998</td>
<td>1133</td>
<td>263</td>
</tr>
<tr>
<td>ERR532393</td>
<td>8230</td>
<td>948</td>
<td>1503</td>
<td>1145</td>
<td>46.8</td>
<td>2083</td>
<td>2366</td>
<td>1126</td>
</tr>
<tr>
<td>SRR1450398</td>
<td>5399</td>
<td>703</td>
<td>854</td>
<td>729</td>
<td>7.7</td>
<td>1345</td>
<td>1532</td>
<td>726</td>
</tr>
<tr>
<td>SRR062462</td>
<td>6478</td>
<td>137</td>
<td>188</td>
<td>144</td>
<td>2.7</td>
<td>222</td>
<td>356</td>
<td>161</td>
</tr>
</tbody>
</table>

Figure 2.4: The compression ratios for all six software suites, indicating the compression ratio = \( \frac{\text{original file size}}{\text{compressed file size}} \).

and subsequent assembly rounds of MetaCRAM reduces the processing time significantly, at the cost of a smaller compression ratio. Table 2.5 compares the compression ratios of MetaCRAM with one round and with two rounds of reference discovery, and indicates that removing the assembly, alignment and BLAST steps adds 1 to 6 MB to the compressed file size. Thus, the user has an option to skip the second round in order to expedite the processing.
Table 2.4: Comparison of processing (compression) times of six software suites. Times are recorded row by row denoting real, user, and system time in order.

<table>
<thead>
<tr>
<th>Data</th>
<th>Time</th>
<th>MCH</th>
<th>MCG</th>
<th>MCEG</th>
<th>bzip2</th>
<th>gzip</th>
<th>MFCompress</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR321482</td>
<td>real</td>
<td>299m</td>
<td>294m</td>
<td>274m</td>
<td>2m</td>
<td>3m</td>
<td>2m</td>
</tr>
<tr>
<td></td>
<td>user</td>
<td>422m</td>
<td>422m</td>
<td>402m</td>
<td>1m</td>
<td>3m</td>
<td>4m</td>
</tr>
<tr>
<td></td>
<td>sys</td>
<td>12m</td>
<td>8m</td>
<td>12m</td>
<td>1s</td>
<td>1s</td>
<td>13s</td>
</tr>
<tr>
<td>SRR359032</td>
<td>real</td>
<td>127m</td>
<td>129m</td>
<td>128m</td>
<td>5m</td>
<td>10m</td>
<td>8m</td>
</tr>
<tr>
<td></td>
<td>user</td>
<td>245m</td>
<td>247m</td>
<td>253m</td>
<td>5m</td>
<td>10m</td>
<td>13m</td>
</tr>
<tr>
<td></td>
<td>sys</td>
<td>8m</td>
<td>10m</td>
<td>15m</td>
<td>2s</td>
<td>2s</td>
<td>15s</td>
</tr>
<tr>
<td>ERR532393</td>
<td>real</td>
<td>639m</td>
<td>635m</td>
<td>641m</td>
<td>11m</td>
<td>22m</td>
<td>17m</td>
</tr>
<tr>
<td></td>
<td>user</td>
<td>1061m</td>
<td>1069m</td>
<td>1090m</td>
<td>11m</td>
<td>21m</td>
<td>28m</td>
</tr>
<tr>
<td></td>
<td>sys</td>
<td>73m</td>
<td>27m</td>
<td>43m</td>
<td>5s</td>
<td>5s</td>
<td>21s</td>
</tr>
<tr>
<td>SRR1450398</td>
<td>real</td>
<td>440m</td>
<td>439m</td>
<td>440m</td>
<td>7m</td>
<td>14m</td>
<td>10m</td>
</tr>
<tr>
<td></td>
<td>user</td>
<td>866m</td>
<td>865m</td>
<td>865m</td>
<td>7m</td>
<td>14m</td>
<td>18m</td>
</tr>
<tr>
<td></td>
<td>sys</td>
<td>21m</td>
<td>23m</td>
<td>26m</td>
<td>3s</td>
<td>3s</td>
<td>18s</td>
</tr>
<tr>
<td>SRR062462</td>
<td>real</td>
<td>217m</td>
<td>224m</td>
<td>215m</td>
<td>2m</td>
<td>2m</td>
<td>6m</td>
</tr>
<tr>
<td></td>
<td>user</td>
<td>254m</td>
<td>261m</td>
<td>256m</td>
<td>2m</td>
<td>1m</td>
<td>10m</td>
</tr>
<tr>
<td></td>
<td>sys</td>
<td>15m</td>
<td>16m</td>
<td>20m</td>
<td>3s</td>
<td>3s</td>
<td>16s</td>
</tr>
</tbody>
</table>

Figure 2.5: The compression times for all six software suites shown in minutes.

Likewise, Table 2.6 and Figure 2.7 illustrates that the retrieval time of MetaCRAM is longer than that of bzip2, gzip, and MFCompress, but still
Figure 2.6: Average Runtime of Each Stage of MetaCRAM. Detailed distribution of the average runtimes of MetaCRAM for all five datasets tested. We used “_1” to indicate the processes executed in the first round, and “_2” to denote the processes executed in the second round.

highly efficient. In practice, the processing time is not as relevant as the retrieval time, as compression is performed once while retrieval is performed multiple times. For long term archival of data, MetaCRAM is clearly the algorithm of choice since the compression ratio is the most important criteria.

We also remark on the impact of different integer encoding methods on the compression ratio. Huffman, Golomb, and extended Golomb codes all have their advantages and disadvantages. For the tested datasets, Huffman clearly achieves the best ratio, as it represents the optimal compression method, whereas Golomb and Extended Golomb compression slightly improve compression time. However, the parallel implementation of MetaCRAM makes the comparison of processing time of the three methods slightly biased: for example, if we perform compression while performing assembly, compression
Table 2.5: Comparison of compressed file sizes of MetaCRAM-Huffman using 2 rounds and 1 round. For short hand notation, we used “MCH-2rounds” = MetaCRAM-Huffman with 2 rounds, “MCH-1round” = MetaCRAM-Huffman with 1 round. We also used the shortcut “MFComp” = MFCompress and “Align. %” refers to the percentage of reads aligned during 2 rounds and 1 round, respectively, for MCH-2rounds and MCH-1round.

<table>
<thead>
<tr>
<th>Data</th>
<th>Original (MB)</th>
<th>MCH-2rounds (MB)</th>
<th>Align. %</th>
<th>MCH-1round (MB)</th>
<th>Align. %</th>
<th>gzip (MB)</th>
<th>MFComp (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR321482</td>
<td>1429</td>
<td>191</td>
<td>29.6</td>
<td>192</td>
<td>27</td>
<td>408</td>
<td>229</td>
</tr>
<tr>
<td>SRR359032</td>
<td>3981</td>
<td>319</td>
<td>61.8</td>
<td>315</td>
<td>57.7</td>
<td>1133</td>
<td>263</td>
</tr>
<tr>
<td>ERR532393</td>
<td>8230</td>
<td>948</td>
<td>46.8</td>
<td>952</td>
<td>45.8</td>
<td>2366</td>
<td>1126</td>
</tr>
<tr>
<td>SRR1450398</td>
<td>5399</td>
<td>703</td>
<td>7.7</td>
<td>707</td>
<td>7.1</td>
<td>1532</td>
<td>726</td>
</tr>
<tr>
<td>SRR062462</td>
<td>6478</td>
<td>137</td>
<td>2.7</td>
<td>143</td>
<td>2.6</td>
<td>356</td>
<td>161</td>
</tr>
</tbody>
</table>

Table 2.6: Comparison of retrieval (decompression) times of six software suites. Times are recorded row by row denoting real, user, and system time in order.

<table>
<thead>
<tr>
<th>Data</th>
<th>Time</th>
<th>MCH</th>
<th>MCG</th>
<th>MCEG</th>
<th>bzip2</th>
<th>gzip</th>
<th>MFComp</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR321482</td>
<td>real</td>
<td>23m</td>
<td>25m</td>
<td>24m</td>
<td>57s</td>
<td>17s</td>
<td>2m</td>
</tr>
<tr>
<td></td>
<td>user</td>
<td>16m</td>
<td>16m</td>
<td>17m</td>
<td>45s</td>
<td>9s</td>
<td>4m</td>
</tr>
<tr>
<td></td>
<td>sys</td>
<td>9m</td>
<td>10m</td>
<td>10m</td>
<td>2s</td>
<td>1s</td>
<td>4s</td>
</tr>
<tr>
<td>SRR359032</td>
<td>real</td>
<td>12m</td>
<td>11m</td>
<td>13m</td>
<td>2m</td>
<td>1m</td>
<td>8m</td>
</tr>
<tr>
<td></td>
<td>user</td>
<td>11m</td>
<td>11m</td>
<td>12m</td>
<td>2m</td>
<td>28s</td>
<td>15m</td>
</tr>
<tr>
<td></td>
<td>sys</td>
<td>2m</td>
<td>1m</td>
<td>3m</td>
<td>4s</td>
<td>2s</td>
<td>19s</td>
</tr>
<tr>
<td>ERR532393</td>
<td>real</td>
<td>48m</td>
<td>47m</td>
<td>56m</td>
<td>5m</td>
<td>2m</td>
<td>15m</td>
</tr>
<tr>
<td></td>
<td>user</td>
<td>39m</td>
<td>40m</td>
<td>43m</td>
<td>4m</td>
<td>55s</td>
<td>29m</td>
</tr>
<tr>
<td></td>
<td>sys</td>
<td>15m</td>
<td>13m</td>
<td>29m</td>
<td>7s</td>
<td>5s</td>
<td>17s</td>
</tr>
<tr>
<td>SRR1450398</td>
<td>real</td>
<td>28m</td>
<td>28m</td>
<td>29m</td>
<td>3m</td>
<td>2m</td>
<td>10m</td>
</tr>
<tr>
<td></td>
<td>user</td>
<td>30m</td>
<td>29m</td>
<td>30m</td>
<td>3m</td>
<td>37s</td>
<td>19m</td>
</tr>
<tr>
<td></td>
<td>sys</td>
<td>7m</td>
<td>5m</td>
<td>7m</td>
<td>5s</td>
<td>3s</td>
<td>26s</td>
</tr>
<tr>
<td>SRR062462</td>
<td>real</td>
<td>23m</td>
<td>22m</td>
<td>26m</td>
<td>1m</td>
<td>1m</td>
<td>6m</td>
</tr>
<tr>
<td></td>
<td>user</td>
<td>21m</td>
<td>21m</td>
<td>22m</td>
<td>42s</td>
<td>22s</td>
<td>10m</td>
</tr>
<tr>
<td></td>
<td>sys</td>
<td>4m</td>
<td>4m</td>
<td>10m</td>
<td>4s</td>
<td>3s</td>
<td>26s</td>
</tr>
</tbody>
</table>
will take much more time than compressing while running an alignment algorithm. As the processing and retrieval time is not consistent among the three methods, we recommend using Huffman coding for archival storage.

2.4 Discussion

In what follows, we comment on a number of useful properties of the MetaCRAM program, including compatibility, losslessness, partial assembly results and compressive computing.

Compatibility. MetaCRAM uses well established and widely tested genomic analysis tools, and it also follows the standard genomic data compression format CRAM, hence making the results of downstream analysis compatible with a current standard for genomic compression.

Lossless compression principle. By its very nature, MetaCRAM is a lossless compression scheme as it encodes the differential information between the reference and the metagenomic reads in a 100% accurate fashion. Nevertheless, we enabled a feature that allow for some partial loss of information, such as the read ID tags. It is left to the discretion of the user to choose suitable options.

CRAM versus MFCompress. MFCompress achieves good compression ratios when compressing highly redundant reads. MetaCRAM consistently
achieves a rate proportional to the alignment rate because it only encodes the small difference between the reference genome and the read. As more microbial genome become available, MetaCRAM will most likely offer higher compression ratio than other tools in general. Note that only on one data file - SRR359032 - did MFCompress achieve better compression ratios than MetaCRAM, most likely due to the redundancy issues previously mentioned.

**Metagenomic assembly.** Metagenomic assembly is a challenging task, and there is a widely accepted belief that it is frequently impossible to perform meaningful assembly on mixture genomes containing species from related genomes. Nevertheless, we are using assembly mostly as a means for identifications, but at the same time its output provides useful contigs for gene transfer analysis and discovery. In the case that assembly fails on a dataset, we suggest skipping the assembly step so as to trade off computation time with discovery of new reference genomes and contigs.

**Compressive computing.** There has been an effort towards computing in the compressed domain, in order to eliminate the need for persistent compression and decompression time when all one needs to perform is simple alignment [44]. Similarly, MetaCRAM offers easy retrieval and selection based on the list of references stored as an option. For example, suppose we perform MetaCRAM on all available human gut metagenome data. If we want to analyze the datasets with a concentration of *Escherichia coli*, we avoid sacrificing retrieval time by quickly scanning the list of reference files and only retrieving the datasets with *E. coli*.
CHAPTER 3

QUANTITATIVE ANALYSIS AND PREDICTION OF G-QUADRUPLEX FOLDING PROPENSITY

3.1 Background

The G-quadruplex (GQ) is a non-canonical DNA secondary structure arising from two or more stacked sets of four guanine (G) nucleotides (G-tetrads) interacting in a plane (Figure 3.1A), although three G-tetrads comprise the most common form in which the four sets of guanine triplets form a four-stranded structure through Hoogsteen base pairing coordinated by monovalent cations. GQ DNA can assume various folding configurations including parallel, antiparallel, and hybrid conformations dictated by ion conditions and loop sequence compositions [45, 46, 47, 48]. A surge of interest in the GQ structure has followed the recent findings, suggesting its multifaceted role in key processes within the central dogma of biology [49, 50, 51, 52, 53, 54, 55]. In particular, it is hypothesized that the formation of GQs modulates gene expression through a physical interaction between the GQ structure and transcription-related protein complexes [56]. In support, recent work has confirmed the capability of GQs to form stably within the genome [57, 58]. Thus, GQs may prove to be an important component in the regulation of specific genes and, as such, may serve as an effective pharmaceutical target for a wide range of diseases [59, 60, 61, 62]. Putative GQ forming sequences are unevenly distributed throughout the human genome, with their presence increased in select gene regulatory regions, such as promoters of oncogenes and immunoglobulin switch regions [63, 64]. This irregular distribution highlights the challenge in identifying functional sequences that can actually form GQ structures in vivo.

GQ forming sequences are frequently modeled following the pattern: \( GG\cdot N_{L1}GG\cdot N_{L2}GG\cdot N_{L3}GG \), where \( N \) can be adenine (A), cytosine (C), or thymine (T), and \( L1, L2, \) and \( L3 \) are positive integers indicating the
lengths of the intervening sequences that correspond to loops in the folded GQ structure (Figure 3.1A). We note that loops can contain G bases, although we do not consider this possibility in our current study. Typical upper limits on loop length have been suggested to be between 7 and 9 bases within a single-stranded DNA (ssDNA) context, but a maximal loop length has not yet been established in a double-stranded DNA (dsDNA) context [65, 66, 67, 68]. Even with such restricted pattern assumptions, determining how nucleotide content and intervening loop lengths control the GQ formation potential of more than 400,000 candidate genomic sequences remains a challenging task. This ambiguity in GQ characterization complicates the identification of true GQ forming sequences implicated in essential biological activities.

The discovery of stable genomic GQ formation coupled with the significant number of potential GQ sequences located within the human genome underscores the need for new tools that can accurately predict folding propensity. Owing to the seemingly regular pattern found in GQ forming sequences, many bioinformatics studies have been conducted on putative GQ sequences [69, 70, 71, 72]. Generally, these studies simply searched for recurring patterns of putative GQs or developed models describing folding propensity based on GQ experiments in ssDNA. As a result, the methods may be biased towards known patterns and miss novel GQ folding sequences. Previously, we showed that the GQ folding propensity is substantially diminished in dsDNA and that, unlike ssDNA, dsDNA has limited ability to form only into parallel GQs [73]. These considerations highlight the need for a new model that can predict GQ folding propensity specifically in a dsDNA context, which is more representative of genomic DNA than ssDNA.

We performed a survey of systematically designed GQ forming sequences to identify folding propensity within a dsDNA context. The survey contained more than 400 putative GQ forming sequences with loops composed entirely of A, C, or T with total loop length ranging up to 12 bp. Quantitative measurement of parallel GQ formation was obtained by N-methyl mesoporphyrin IX (NMM) fluorescence assay that was established in our previous work [73]. The NMM intensity measurements were complemented by single-molecule fluorescence resonance energy transfer (smFRET) experiments, which enabled direct quantitation of molecules comprising both the GQ-folded and unfolded populations (Figure 3.1B). We utilized these com-
plementary methods to categorize each sequence as one of “strongly folding”, “non-folding”, or “combined” classes, providing a simple metric for comparing the folding propensities of specific putative GQ sequences. Furthermore, by analyzing the impact of loop lengths and compositions on the NMM intensity measurement, we identified GQ-driving loop parameters. These results were combined in regression models that can predict GQ folding propensity with high accuracy. Our GQ folding experimental platform and computational models will serve as a useful reference that facilitates the investigation of potential genomic GQs in the future.

Figure 3.1: An overview of G-quadruplex structure and the NMM technique. A) A schematic of a parallel GQ structure is depicted. The guanine-guanine Hoogsteen base pairing between each guanine triplet is shown for the sequence $GGGN_{L1}GGGNN_{L2}GGGNN_{L3}GGG$, where N denotes the nucleotide component and $L1, L2, L3$ are the three loop lengths. B) GQ folding propensity is investigated through an induced fluorescence based assay. The molecule NMM shows a specific increase in fluorescence signal upon binding to a parallel GQ sequence. C) A plate is filled with strong folding sequences in high intensity, combined folding and non-folding sequences in a lower intensity, and non-folding sequences in low intensity.
3.2 Methods

3.2.1 Experimental data

For a given sequence, three readings of NMM measurements were recorded and the average intensity value was used throughout the analysis. We represented the loop components of a GQ sequence using the length vector \((L_1, L_2, L_3)\) and nucleotide content \(N\). For instance, \((4,1,2)\) and \(N = A\) encodes the sequence \(GGGAAAAGGGAGGAAGGG\). We only considered the cases where all nucleotides in the loops are the same, in order to fully characterize the rules governing these simple, yet poorly understood cases. The total length of intervening sequences is denoted as \(L = L_1 + L_2 + L_3\). We considered combinations of \(L_1, L_2,\) and \(L_3\) such that \(L \leq 12\), and \(N\) is allowed to be A, C or T. For each \(N\), there are 4 sequences corresponding to \(L_1 = L_2 = L_3\) and \(26 \times 3\) sequences corresponding to the case where exactly two of the lengths are equal, accounting for \(4 + 26 \times 3 = 82\) total points in which at least two of the intervening sequences are repeated. There are a total of 138 possible combinations of loop lengths, such that \(L_1, L_2,\) and \(L_3\) are distinct and \(L \leq 12\), but we subsampled 64 cases for our measurements in order to reduce the dimension, as explained in Table 3.1. Thus, we have a total of \((82 + 64) \times 3 = 438\) readings, corresponding to 146 combinations of loop lengths for three different nucleotides.

We fitted the histogram of intensity values to a mixture of two or three Gaussian distributions by using the Expectation-Maximization algorithm (“mixtools” package in R) and plotted individual values using the “colorRamps” and “calibrate” packages in R. Categorical histograms based on the nucleotide composition or the minimum loop length composition were plotted, and the distribution of a given subset of categories was compared to the rest of the categories via the one-sided unpaired Wilcoxon rank sum test. Finally, we applied the two-sided Kolmogorov-Smirnov test to compare the distributions of T, C, and A pairwise.

3.2.2 Linear regression

As a first step, we naïvely applied a linear regression model of the NMM intensity against the predictor variables \(L_1, L_2, L_3, seqT, seqC\) and an in-
Table 3.1: List of all possible data points with unique $L_1$, $L_2$, and $L_3$, such that $L_1 + L_2 + L_3 \leq 12$, in the ascending order of $\text{min}L$, $\text{med}L$, and $\text{max}L$ without any redundancy in permutation. The test data comprise all 6 permutations of 10 rows highlighted in gray, along with (3,4,5), (5,3,4), (4,3,5), and (4,5,3), summing to data points for each of $N = A, C, T$.

<table>
<thead>
<tr>
<th>No.</th>
<th>MinL</th>
<th>MedL</th>
<th>MaxL</th>
<th>No.</th>
<th>MinL</th>
<th>MedL</th>
<th>MaxL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>13</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>14</td>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>15</td>
<td>1</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>16</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>17</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>18</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>19</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>20</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>21</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>22</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>23</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

tercept term, where seqT and seqC are indicator variables for T and C nucleotides, respectively. Note that seqA was omitted due to the linear constraint $\text{seq}A = 1 - \text{seq}C - \text{seq}T$. We then examined an alternative model by replacing $L_1, L_2, L_3$ with $\text{min}L, \text{med}L, \text{max}L$, where $\text{min}L, \text{med}L$, and $\text{max}L$ correspond to the minimum, median, and maximum of the three loop lengths. We trained both models on all 438 sequences’ NMM intensities to obtain interpretable coefficients and model prediction. This analysis showed that the second model outperformed the first approach, and we thus used the predictor variables $\text{min}L, \text{med}L$, and $\text{max}L$ thereafter. Subsequently, we performed six-fold cross-validation to demonstrate that our model is robust. We randomly partitioned the population into 6 groups, each containing 73 points. Using one group as test data and the remaining five groups as training data, we computed the average coefficient of determination for both test and training data. We adopted the following definition of the coefficient of determination: $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$, where $\hat{y}_i$ is the predicted value and $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$ is the mean of $n$ samples used for calculating $R^2$. For example, $n = 365$ for training data, and $n = 73$ for test data. Likewise, the residual for each sample $i$ is defined as $y_i - \hat{y}_i$, i.e., the difference between the observed
and predicted values. The linear regression method has many advantages such as its simplicity and the interpretability of the coefficients. However, it has the limitation of assuming linearity of the response in predictor variables.

3.2.3 Gaussian process regression

Gaussian process regression (GPR) is a flexible non-parametric regression method that does not assume linearity of the response in predictor variables [74]. A Gaussian process $f$ is defined on a set $X$ by specifying that the values of $f$ on any finite number of points in $X$ form random variables following a joint Gaussian distribution, with mean 0 and fixed covariance $k(x, x')$ at $x, x' \in X$. Thus, we only need to define the covariance function $k(x, x')$ in order to specify a Gaussian process; $k(x, x')$ is a kernel that measures the similarity between inputs $x$ and $x'$. The choice of covariance function plays an important role in model prediction, and a popular choice is the squared exponential function:

$$k_{SE}(x, x') = \sigma_f^2 e^{-\frac{(x-x')^2}{2l^2}} + \sigma_n^2 \delta(x, x'),$$

where the hyperparameters $\sigma_f^2$ and $\sigma_n^2$ are the variance of the process and experimental measurement, respectively, $l$ is the length scale of fluctuation, and $\delta(x, x')$ is the Kronecker delta function. For $n$ training data points $(x_i, y_i), i = 1, \ldots, n$, we construct an $n$ by $n$ covariance matrix

$$K = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{bmatrix}.$$

For a test data point $x_*$, we define

$$K_* = \begin{bmatrix} k(x_*, x_1) & k(x_*, x_2) & \cdots & k(x_*, x_n) \end{bmatrix}$$

and $K_{**} = k(x_*, x_*)$. Then, the joint distribution of the observed output $y$ and predicted output $y_*$ is assumed to be

$$\begin{bmatrix} y \\ y_* \end{bmatrix} \sim N \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K & K_*^T \\ K_* & K_{**} \end{bmatrix} \right),$$
and the predictive distribution is \( y_* | y \sim N(K_* K^{-1} y, K_{**} - K_* K^{-1} K_{*}^T) \). We subsequently obtain our prediction as the mean \( \bar{y}_* = K_* K^{-1} y \). The above methods were all implemented using GPML MATLAB package [75].

**Choice of covariance functions.** A valid covariance function \( k(x, x') \) requires the function to be symmetric and positive semi-definite. In addition, many of the widely used kernels are stationary, i.e., it is a function of only the distance \( r = |x - x'| \). Two examples of stationary covariance functions are a noiseless squared exponential \( k_{SE}(r) = e^{-\frac{r^2}{2\ell^2}} \), with length-scale parameter \( \ell \), and a Matérn class \( k_{Mat,\nu}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left( \frac{\sqrt{2\nu} r}{\ell} \right)^\nu K_\nu \left( \frac{\sqrt{2\nu} r}{\ell} \right) \), with positive parameters \( \nu \) and \( \ell \), and a modified Bessel function of the second kind \( K_\nu \). For half-integer \( \nu \), the Matérn function \( K_\nu \) is a product of an exponentially decaying function and a polynomial, with \( \nu = \frac{1}{2} \) giving a non-smooth process. As \( \nu \to \infty \), the Matérn function behaves similarly to the squared exponential function, which is smooth. We used parameters of different length for each predictor variable, adding flexibility to the input space.

Denoting our predictors \((\text{minL}, \text{medL}, \text{maxL}, \text{seqA}, \text{seqC}, \text{seqT})\) as \((x_1, x_2, x_3, x_4, x_5, x_6)\), we defined our noiseless covariance function as

\[
k((x_1, \ldots, x_6), (x'_1, \ldots, x'_6)) = x_4 x'_4 \sigma_{f,A}^2 k_{\text{Mat},\nu=5/2}((x_1, x_2, x_3), (x'_1, x'_2, x'_3)) + x_5 x'_5 \sigma_{f,C}^2 k_{SE}((x_1, x_2, x_3), (x'_1, x'_2, x'_3)) + x_6 x'_6 \sigma_{f,T}^2 k_{\text{Mat},\nu=3/2}((x_1, x_2, x_3), (x'_1, x'_2, x'_3)),
\]

where \( k_{\text{Mat},\nu}() \) is the Matérn kernel with specific \( \nu \), \( k_{SE}() \) is the squared exponential kernel, and \( \sigma_{f,A}^2, \sigma_{f,C}^2, \sigma_{f,T}^2 \) correspond to the variance of the process for seqA, seqC, and seqT, respectively. This combination has been derived by testing the squared exponential and Matérn class with \( \nu = \frac{1}{2}, \frac{3}{2}, \frac{5}{2} \) separately for seqA, seqC, seqT, and choosing the best function for each nucleotide.

**Estimation of hyperparameters.** There are four hyperparameters \( \sigma_{f,N}, l_{1,N}, l_{2,N}, l_{3,N} \) (the length scale for minL, medL, maxL, respectively) for each nucleotide N, summing to 12. A common method to estimate a set of hyperparameters \( \theta \) is by maximizing the marginal log-likelihood

\[
\log p(y|X, \theta) = -\frac{1}{2} y^T K_{y}^{-1} y - \frac{1}{2} \log |K_{y}| - \frac{n}{2} \log(2\pi),
\]

where \( K_{y} = K + \sigma_n^2 I \) and \( x \) and \( y \) are predictor and response variables for the training data. We
also adopted this method, but implemented it in two steps. First, for each individual nucleotide, we initialized each of $l_{1,N}$, $l_{2,N}$, $l_{3,N}$ and $\sigma_{f,N}$ to be 5, and obtained an estimate by using a conjugate gradient method. Note that there are three separate estimates for $\sigma_{f,N}$, obtained for each of $l_{1,N}$, $l_{2,N}$, and $l_{3,N}$, and that we let the final estimate be the average of the three. We then initialized all 12 hyperparameters with the values obtained from the previous step and maximized the marginal log-likelihood over all lengths and nucleotides. This approach allows for more flexibility in each length scale than treating each loop length with an equal weight. Finally, we estimated $\sigma_n^2 = 18$ as the empirical covariance of our replicate experimental NMM intensity measurements. Table 3.2 contains the estimated hyperparameters used for fitting the entire validation set, and the same estimation method was repeated for each cross validation set.

Table 3.2: Estimated hyperparameters for all 438 data points used in GPR. There are four parameters for each nucleotide base: length scales corresponding to minimum, median, and maximum loop length, and the standard deviation for each process.

<table>
<thead>
<tr>
<th>Hyperparameters estimated</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_{1,A}$ (minL, seqA)</td>
<td>4.5528</td>
</tr>
<tr>
<td>$l_{2,A}$ (medL, seqA)</td>
<td>3.8257</td>
</tr>
<tr>
<td>$l_{3,A}$ (minL, seqA)</td>
<td>9.3157</td>
</tr>
<tr>
<td>$\sigma_{f,A}$ (seqA)</td>
<td>469.1853</td>
</tr>
<tr>
<td>$l_{1,C}$ (minL, seqC)</td>
<td>2.1615</td>
</tr>
<tr>
<td>$l_{2,C}$ (medL, seqC)</td>
<td>5.4847</td>
</tr>
<tr>
<td>$l_{3,C}$ (maxL, seqC)</td>
<td>3.3017</td>
</tr>
<tr>
<td>$\sigma_{f,C}$ (seqC)</td>
<td>463.0511</td>
</tr>
<tr>
<td>$l_{1,T}$ (minL, seqT)</td>
<td>7.5167</td>
</tr>
<tr>
<td>$l_{2,T}$ (medL, seqT)</td>
<td>16.3854</td>
</tr>
<tr>
<td>$l_{3,T}$ (maxL, seqT)</td>
<td>23.4539</td>
</tr>
<tr>
<td>$\sigma_{f,T}$ (seqT)</td>
<td>599.0285</td>
</tr>
</tbody>
</table>

3.2.4 Spectral analysis

To examine the effect of permuting loop lengths on GQ folding propensity, spectral decomposition was performed on the NMM intensity values treated
as functions defined on the symmetric group $S_3$, the group of all permutations $\pi$ of a three-element set. For a given sequence with nucleotide $N$ and loop lengths $(L_1, L_2, L_3)$, we defined the following group of 6 elements using the cycle notation for $S_3$ in Table 3.3.

Table 3.3: Group of 6 elements using the cycle notation for $S_3$.

<table>
<thead>
<tr>
<th>$\pi$</th>
<th>$L_1$</th>
<th>$L_2$</th>
<th>$L_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>minL</td>
<td>medL</td>
<td>maxL</td>
</tr>
<tr>
<td>(2 3)</td>
<td>minL</td>
<td>maxL</td>
<td>medL</td>
</tr>
<tr>
<td>(1 2)</td>
<td>medL</td>
<td>minL</td>
<td>maxL</td>
</tr>
<tr>
<td>(1 3 2)</td>
<td>maxL</td>
<td>minL</td>
<td>medL</td>
</tr>
<tr>
<td>(1 2 3)</td>
<td>medL</td>
<td>maxL</td>
<td>minL</td>
</tr>
<tr>
<td>(1 3)</td>
<td>maxL</td>
<td>medL</td>
<td>minL</td>
</tr>
</tbody>
</table>

We let $f(\pi)$ be the NMM intensity value for the loop length configuration given by $\pi$, and denoted the three irreducible representations of $S_3$ as the trivial, sign, and two-dimensional representation $\rho$. By defining $\sqrt{2}\rho(\pi^{-1}) = \begin{pmatrix} a(\pi) & b(\pi) \\ c(\pi) & d(\pi) \end{pmatrix}$, we have listed (Table 3.4) and plotted (Figure 3.2) the matrix elements $a(\pi), b(\pi), c(\pi), d(\pi)$ for each element $\pi$ in $S_3$. We then took the inverse Fourier transform and expanded the intensity values in terms of these irreducible representations as follows: $f(\pi) = \frac{1}{6}(\hat{f}(\text{triv}) + \text{sign}(\pi) \cdot \hat{f}(\text{sign}) + \sqrt{2}[a(\pi)\hat{f}(a) + b(\pi)\hat{f}(b) + c(\pi)\hat{f}(c) + d(\pi)\hat{f}(d)])$, where the Fourier transform $\hat{f}$ is defined as $\hat{f}(r) = \sum_{\pi \in S_3} r(\pi) f(\pi)$ for an irreducible representation $r$. By examining the Fourier coefficients $\hat{f}$ of $a(\pi), b(\pi), c(\pi), d(\pi)$ in this expansion and keeping terms with the largest coefficient in magnitude, we obtained a dominant mode decomposition of the behavior of $f$ on loop length permutations, allowing us to easily isolate which loop configuration contributes to GQ folding the most.
Table 3.4: Irreducible representations for each permutation \( \pi \). triv is the trivial representation, sign is the sign representation, and \( \rho \) is the two-dimensional representation where \( \sqrt{2}\rho(\pi^{-1}) = \begin{pmatrix} a(\pi) & b(\pi) \\ c(\pi) & d(\pi) \end{pmatrix} \).

<table>
<thead>
<tr>
<th>( \pi )</th>
<th>id</th>
<th>(2 3)</th>
<th>(1 2)</th>
<th>(1 3 2)</th>
<th>(1 2 3)</th>
<th>(1 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho(\pi) )</td>
<td>\begin{pmatrix} 1 &amp; 0 \ 0 &amp; 1 \end{pmatrix}</td>
<td>\begin{pmatrix} \frac{\sqrt{3}}{2} &amp; \frac{-1}{2} \ \frac{-1}{2} &amp; \frac{\sqrt{3}}{2} \end{pmatrix}</td>
<td>\begin{pmatrix} -1 &amp; 0 \ 0 &amp; 1 \end{pmatrix}</td>
<td>\begin{pmatrix} -\frac{1}{2} &amp; \frac{\sqrt{3}}{2} \ -\frac{\sqrt{3}}{2} &amp; -\frac{1}{2} \end{pmatrix}</td>
<td>\begin{pmatrix} -\frac{1}{2} &amp; -\frac{\sqrt{3}}{2} \ -\frac{\sqrt{3}}{2} &amp; -\frac{1}{2} \end{pmatrix}</td>
<td>\begin{pmatrix} 1 &amp; -\frac{\sqrt{3}}{2} \ -\frac{\sqrt{3}}{2} &amp; 1 \end{pmatrix}</td>
</tr>
<tr>
<td>triv(( \pi ))</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>sign(( \pi ))</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>a(( \pi ))</td>
<td>( \sqrt{2} )</td>
<td>(-\frac{1}{\sqrt{2}} )</td>
<td>(-\sqrt{2} )</td>
<td>(-\frac{1}{\sqrt{2}} )</td>
<td>(-\frac{1}{\sqrt{2}} )</td>
<td>(-\frac{1}{\sqrt{2}} )</td>
</tr>
<tr>
<td>b(( \pi ))</td>
<td>0</td>
<td>( \frac{\sqrt{3}}{2} )</td>
<td>0</td>
<td>(-\frac{\sqrt{3}}{2} )</td>
<td>( \frac{\sqrt{3}}{2} )</td>
<td>(-\frac{\sqrt{3}}{2} )</td>
</tr>
<tr>
<td>c(( \pi ))</td>
<td>0</td>
<td>( \frac{\sqrt{3}}{2} )</td>
<td>0</td>
<td>(-\frac{\sqrt{3}}{2} )</td>
<td>( \frac{\sqrt{3}}{2} )</td>
<td>(-\frac{\sqrt{3}}{2} )</td>
</tr>
<tr>
<td>d(( \pi ))</td>
<td>( \sqrt{2} )</td>
<td>(-\frac{1}{\sqrt{2}} )</td>
<td>( \sqrt{2} )</td>
<td>(-\frac{1}{\sqrt{2}} )</td>
<td>(-\frac{1}{\sqrt{2}} )</td>
<td>(-\frac{1}{\sqrt{2}} )</td>
</tr>
</tbody>
</table>

Figure 3.2: Four components of the 2D irreducible representation \( \rho \). For a permutation of three elements, the four components \( a(\pi), b(\pi), c(\pi), d(\pi) \) of the two-dimensional irreducible representation \( \rho \) are plotted according to Table 3.4.

3.3 Results

3.3.1 Pilot study establishes cutoff for GQ folding

We designed a series of GQ forming dsDNA constructs by following the conventional pattern \([G G G N_{L1} G G G N_{L2} G G G N_{L3} G G G]\) as defined above (Fig-
We have excluded the loop lengths that would not support GQ folding based on our previous study that revealed a significantly diminished GQ folding potential in dsDNA compared to ssDNA. As a pilot study, we designed 246 sequences that satisfied the following three conditions. First, the total loop length, $L = L_1 + L_2 + L_3$, was restricted to be 12 bases or less. Second, all loops consisted entirely of only one nucleotide, A, C or T. Third, at least two loop lengths were of equal length. NMM was applied to each DNA in 96 well plates, and the induced fluorescence from NMM was measured to assess the GQ folding potential (Figure 3.1B, C). The NMM measurement was repeated three times per DNA and the results were highly reproducible (average standard deviation = 18). The NMM-based fluorescence assay allows detection of parallel GQ structure, which is the only form of GQ that can form in dsDNA. The NMM signal induced by potential GQ-dsDNA indicates the degree of its GQ folding. We expect a high NMM signal for DNA that primarily forms into a GQ, intermediate intensity for a combined population of folded and non-folded GQs, and no signal if all DNA molecules become duplexed (Figure 3.1C).

Based on the NMM intensity, we roughly categorized the folding propensity of the 246 sequences into folding (> 254) and non-folding (< 254) classes by using a Gaussian mixture model (Figure 3.3A). The Kolmogorov-Smirnov test did not detect a statistically significant deviation of the model from the data (two-sided $p$-value = 0.315), supporting the goodness of fit. The NMM intensity cutoff of 254, estimated from the transition point in the ratio of posterior class probabilities, corresponded to 52% and 48% of the sequences as folding and non-folding, respectively. In order to check whether all three nucleotide types yield similar NMM intensity distributions, we grouped the data by the nucleotide content of loop sequences and plotted the empirical cumulative distribution for each group (Figure 3.3B). The distribution for T was clearly shifted to the right, strongly suggesting that T loops induce a stronger GQ folding potential than A and C loops. This effect is further analyzed and discussed below.
Figure 3.3: Pilot study of NMM fluorescence data points and relationship with smFRET scores. A) The population of 246 sequences is separated into non-folding (blue) and folding (red) classes via the Gaussian mixture model. Dotted line shows the marginal (total) distribution of NMM intensities in the fitted mixture model. B) The empirical cumulative distribution functions (CDF) are plotted for three nucleotides, A (red), C (green), and T (blue).

3.3.2 Expansive coverage of candidate sequences identifies loop length and composition dependence of folding trends

In order to further investigate the dependence of GQ-folding trends on loop lengths and the nucleotide content, we visualized our initial data by constructing color-weighted NMM intensity graphs. For a clear illustration of the previously observed GQ-folding pattern, we first partitioned the data into three groups according to the loop length composition. The loop lengths \((L_1, L_2, L_3)\) were encoded in a two-dimensional space, instead of three dimensions, by defining the variable \(Z\) to denote the length that is repeated at least twice, and \(V\) the remaining length. Using these two variables, the three possible permutations of loop lengths considered were coded as \((Z,Z,V)\), \((Z,V,Z)\), and \((V,Z,Z)\) (Figure 3.4A). Each of these three groups were then further partitioned into three classes based on the loop sequence, T, C and A, thereby visually capturing the experimental NMM intensities of all 246 sequences via 9 different subgraphs (Figure 3.4B). High GQ-induced NMM fluorescence levels were displayed in red (warm) colors, while low intensi-
ties were shown in blue (cool) colors. The sequences with nucleotide T and loop pattern (V,Z,Z) are shown in Figure 3.4A. This representation clearly demonstrates an inverse relation between the intensity and minimum length ($minL$), as shown by the similar colors for sequences with the same $minL$ and the color gradient with respect to increasing $minL$ (red and yellow for the 14 sequences with $minL = 1$, mostly green for the 10 sequences with $minL = 2$, and dark blue for $minL > 2$). By contrast, the correlation between intensity and total length ($L$) remained weak, as shown by the wide fluctuation of colors for sequences with the same $L$. For example, the sequences in each group with $7 \leq L \leq 12$ displayed colors ranging from red to blue, providing little insight on the likelihood of a particular group of sequences to fold. In order to validate this observation more rigorously, we computed the partial correlation between $minL$ and intensity given a controlling variable $L$, and obtained -0.64. By contrast, the partial correlation between $L$ and intensity given $minL$ was -0.36, implying that there is stronger correlation between $minL$ and intensity than between $L$ and intensity.

We compared the subgraphs to further investigate the effect of nucleotide content and length distributions on the GQ folding intensity (Figure 3.4B). Comparing the three rows pairwise revealed that C and A loop compositions generally showed a lower folding pattern than T, consistent with our previous observation (Figure 3.3B). For example, in all three permutations of the loop lengths (3,2,2), T exhibited yellow or green colors (in the range 400 to 500) whereas C and A displayed light or dark blue (less than 250); according to the NMM intensity cutoff value of 254 derived in the previous section, only the T-containing sequences were folding in these cases, thus exemplifying the overall diminished GQ folding for C and A compared to T. Examining the effect of loop lengths on folding, we found that the inverse relation between the minimum loop length and intensity observed in Figure 3.4A was present in all groups: sequences with $minL = 1$ generally displayed high intensity, whereas the intensity values rapidly dropped as $minL$ increased. Furthermore, even though the intensities were generally not affected by the ordering of loop lengths, we noticed that for T and A, the sequence arrangements of (1,maxL,1) were less likely to fold than (1,1,maxL) or (maxL,1,1), as the 10 data points along the left-most vertical line in the (Z,V,Z) column exhibited cooler colors than those in the (V,Z,Z) and (Z,Z,V) columns. Likewise, the diminished intensity of 5 data points along the bottom-most horizontal line
of the \((Z,V,Z)\) column indicated that \((\text{max}L,1,\text{max}L)\) was less likely to fold than \((1,\text{max}L,\text{max}L)\) and \((\text{max}L,\text{max}L,1)\) for all \(T\), \(C\), and \(A\) loops.

In order to test and validate our observations from the initial data, we next expanded the study design to include sequences with unique loop lengths in all three positions, while keeping the total loop length at 12 base pairs or less. Of 138 such possible combinations, we subsampled 64 combinations for each nucleotide by selecting every other point in each of 7 unique combinations of \(\text{min}L\) and \(\text{med}L\) in the ordered list (Table 3.1). This choice allowed us to reduce the number of new cases by roughly half, yielding a total of \(246 + 64 \times 3 = 438\) sequences. When applied to the NMM fluorescence assay, the new 192 data points with unique loop lengths yielded an intensity dis-
distribution pattern that differed from the first 246 pilot DNA sequences tested above. Instead of the bimodal distribution seen in the previous pilot data (Figure 3.3A), the new set of DNA displayed a broad single peak centered around 300 (Figure 3.5A). This difference is likely due to the change in the distribution of loop lengths for the new sets of DNA. The loops in the pilot DNA were constrained to possess at least two repeated lengths, while the loops in the new design had unique lengths in the three positions. As a result, the two sets had similar minimum loop length distributions and significantly different median and maximum loop length distributions (two-sided Kolmogorov-Smirnov test p-value = 0.0044, $2.4 \times 10^{-11}$, $5.66 \times 10^{-15}$ for minL, medL, maxL, respectively; Figure 3.6). Compared to the pilot data, the new set contained a substantially higher fraction of sequences with long medL and maxL loop lengths, likely contributing to the broad peak in the mid-to-low range of NMM intensity. We subsequently confirm this hypothesis using regression models. When the data were grouped by individual bases, we again observed the highest GQ folding potential for T, followed by C and A (Figure 3.5B). The same set of data analyzed by the colorimetric mapping still followed the same trend as previously observed: short minL and nucleotide T both led to high folding propensity (Figure 3.7). Hereafter, we used this comprehensive data set to verify our observations using rigorous statistical methods and to devise predictive regression models applicable to a general set of sequences.

### 3.3.3 GQ folding depends on minimum loop length and nucleotide T

As an initial means to understand the combined data set, we first categorized the 438 experimentally generated NMM intensity values into three classes based on a mixture of three Gaussian distributions fitted via the Expectation-Maximization algorithm (Figure 3.8A). This partitioning was based on the two peaks observed in pilot data (Figure 3.3A) and the third peak in the second data set (Figure 3.5A), and the two-sided Kolmogorov-Smirnov test p-value of 0.88 confirmed a good model fit. Comparing the ratios of posterior class probabilities suggested the following three GQ folding categories: (1) Intensity < 151 for non-folding, (2) 151 < Intensity < 412 for combined
Figure 3.5: Overview of new set of 192 sequences tested. A) The density of the new set is plotted and the Gaussian distribution is overlaid in orange, where mean and variance are calculated from the 192 intensity values. B) The empirical cumulative distribution functions (CDF) are plotted for sequences in A (red), C (green), and T (blue).

folding and non-folding, and (3) intensity > 412 for strong folding. Each of the non-folding, combined, and strong folding categories contained 31%, 39%, and 30% of the data, respectively.

Using the above threshold values as a guideline, we investigated the role of loop nucleotide content on folding. The three nucleotide-specific histograms of NMM intensity clearly showed that sequences containing T had a greater tendency to fold than those containing C or A (one-sided unpaired Wilcoxon rank sum test p-value = $4.3 \times 10^{-13}$ for sequences containing T vs. those containing C or A; Figure 3.8B). Moreover, the overall distribution for T was significantly different from that for C or A (two-sided Kolmogorov-Smirnov (KS) test p-value = $2.8 \times 10^{-7}$ and $2.008 \times 10^{-9}$ for C and A, respectively; Figure 3.9), while the distribution for C was not significantly different from that for A (two-sided KS test p-value = 0.13; Figure 3.9).

The three loop lengths L1, L2, and L3 have been previously proposed to modulate GQ folding, but the rule governing their effect remains unknown. Inspection of the intensity plots in Figure 3.4 revealed that an informative feature was the minimum of loop lengths (minL). Indeed, the intensity histograms plotted for different $minL$ values showed that the sequences with $minL = 1$ spanned all three folding categories, although slightly skewed to-
wards the strong folding region, those with $\min L = 2$ were either non-folding or combined, and those with $\min L > 2$ were mostly non-folding (one-sided unpaired Wilcoxon rank sum test p-value $< 2.2 \times 10^{-16}$ for $\min L = 1$ vs. $\min L > 1$; Figure 3.8C). The NMM intensity thus decreased dramatically as the minimum loop length increased, suggesting that transforming the loop lengths $L_1, L_2,$ and $L_3$ to order statistics $\min L, \med L,$ and $\max L$ may help predict GQ intensity. Our regression models in the subsequent section explore this transformation, after attempting a simpler linear fit with $L_1, L_2,$
Figure 3.7: Comprehensive intensity of all 438 data points. A) 41 data points corresponding to (minL, medL, maxL) and T are plotted. The graph is separated into 4 regions according to the minimum loop length (minL=1, 2, 3, 4), with the horizontal axis as the median length (medL) and the vertical axis as the maximum length (maxL). Of the 41 data points, 30 points are from the pilot study: the left-most vertical line in each region corresponds to the case minL = medL, and the 45 degrees off diagonal line corresponds to medL=maxL. NMM intensity values are colored according to the color bar provided on the left. B) The plot is divided into 18 subgraphs according to three nucleotides N=T,C,A, in order, and six possible permutations for ordered loop lengths. Each subgraph is plotted in the same structure as Part A.

and L3.
Figure 3.8: Histograms of 438 comprehensive NMM fluorescence data points. A) The density of NMM intensities is plotted and the Gaussian mixture model separates the population into 3 separate classes: non-folding (blue), combined (green), and strong folding GQs (red). Dotted line shows the marginal distribution of NMM intensities in the fitted mixture model. B) Three independent histograms of the NMM intensities are provided for each loop composition T, C, and A. Bars are colored according to their GQ classification from part A: blue if intensity < 151, green if 151 < intensity < 412, and red if intensity > 412. C) Histograms of the NMM intensities are provided for sequences with minimum loop length 1, 2, and greater than 2, and the bars are colored according to the GQ class that they belong to.

3.3.4 Regression models predict GQ folding propensity

To learn how GQ folding propensity depends on the characteristic features of intervening loops, we first fitted the experimental NMM intensities using a linear regression model with the following five predictor variables: \( L_1, L_2, L_3, seqT, \) and \( seqC. \) The \( seqT \) and \( seqC \) are indicator variables for the T and C nucleotides, respectively, and \( seqA = 1 - seqT - seqC \) is omitted due to its linear dependency on \( seqT \) and \( seqC. \) Training on all 438 sequences, we obtained an \( R^2 \) value of 0.35, implying that our model could predict only 35 percent of the total variance in NMM intensities. By transforming the three loop lengths to the order statistics \( minL, medL, maxL, \) our \( R^2 \) value significantly improved to 0.80. Thus, our subsequent anal-
Figure 3.9: Two-sided Kolmogorov-Smirnov test on pairwise distributions of intensities for T,C,A. A) Empirical Cumulative Distribution Function (ECDF) is plotted for intensities of sequences with T and C separately. The red dots indicate the points in which the maximum gap between two ECDF occurs, and the red dotted lines are the maximum gap. B) ECDF for intensities of sequences T and A. C) ECDF for intensities of sequences C and A.

yses are based on this transformation. The predicted mean intensity was \( \hat{y} = 679 + 149 \text{seq}T + 27 \text{seq}C - 147 \text{min}L - 74 \text{med}L - 4 \text{max}L \). Among the regression coefficients, the two largest magnitudes corresponded to \( \text{seq}T \) and \( \text{min}L \), confirming that the two main driving factors of GQ folding are the T loop composition and the minimal loop length. By contrast, \( \text{seq}C \) and \( \text{max}L \) had the smallest magnitudes and had the least significant p-values of 0.008 and 0.125, respectively, suggesting that neither contributes substantially to folding. The fact that the coefficient for \( \text{seq}C \) was relatively small also indicated that there was very little difference between A and C nucleotides. By contrast, the effect of T on folding was more than five-fold greater than that of C. These results are consistent with the similarity in intensity distribution between C and A, and the distinction from T previously detected by the Kolmogorov-Smirnov test (Figure 3.9).

To test the generalizability of our model, we performed six-fold cross validation. The dataset of 438 points was randomly partitioned into 6 groups, and each group was tested using parameters trained from the remaining 5 groups. As a result, we obtained an \( R^2 \) value of 0.796 ± 0.005 for the training set and a comparable value of 0.784 ± 0.023 for the test set, supporting that our model is robust. We plotted the average absolute values of residuals, defined as the difference between the observed and the predicted values, in
order to visualize how well the model fits each data point (Figure 3.10). De-
spite the simple nature of our model, most of our predictions did not deviate
substantially from the observed true values, as indicated by the overall blue
colors (|residuals| < 150). There were, however, some outlier data points
for A and C nucleotides showing a poor fit when at least two lengths were
repeated. Moreover, the most critical issue for all nucleotides was that the
points (1,1,1), (2,2,2), (3,3,3) and (4,4,4) had large absolute residuals, most
likely due to non-linear behaviors of their intensities. In order to improve
our prediction accuracy, especially at these outlier points, we developed a
Gaussian process regression (GPR) model.

Compared to the linear regression model’s $R^2$ value of 0.80, the GPR model
trained with the same predictor variables on all 438 sequences showed a sub-
stantial improvement to $R^2 = 0.92$. Six-fold cross validation using the same
partition groups from the linear regression analysis yielded $R^2 = 0.918\pm0.002$
for training and $R^2 = 0.878\pm0.039$ for test data, which, on average, improved
the linear model results by 0.12 and 0.09, respectively. To visualize the over-
all performance of the GPR method and compare it with that of the linear
model, the average absolute values of residuals for GPR were again plotted
(Figure 3.11; cf. Figure 3.10). The plot was generally cooler than Figure 3.10,
especially at the data points that were problematic with the linear regression
approach, e.g. the A-containing sequences with loop lengths (1,1,maxL) and
(2,2,maxL). Additionally, we observed significant improvements in predicting
(1,1,1), (2,2,2), (3,3,3), and (4,4,4) for all nucleotides, thus addressing the
major difficulties encountered in the linear model. Overall, the only data
points with large prediction errors were (2,2,2) for sequence T, and (1,8,1)
and (2,2,2) for A, with absolute residuals of around 200, compared to the
rest being less than 100.

Although GPR does not directly provide easily interpretable coefficients as
in linear regression, the estimated hyperparameters do confirm our findings
from the linear model (Table 3.2). For the squared exponential and Matern
class covariance functions, the length parameter $l$ controls the effect size of
the difference in the corresponding predictor variable, and its large value
suggests that the response variable is not very sensitive to the corresponding
feature. Consistent with the linear regression result, we observed that the
length parameters $l_1,A$, $l_1,C$, $l_1,T$, for minL were shorter than those for maxL,
implying that the intensity depended on minL more than on maxL, with the
Figure 3.10: Linear regression predictions absolute value of residuals, $|\text{observed} - \text{predicted}|$, are plotted for linear regression, averaged over 6 cross-validations for each data point. The rows represent 6 different permutations of ordered loop lengths and columns are the three nucleotides, and the values of $|\text{residuals}|$ are colored according to the color bar provided on the right.
effect being most notable for the T nucleotide.
3.3.5 Spectral analysis reveals pattern in pilot study

Investigating the effects of permutation of lengths through spectral decomposition method revealed a distinct pattern in the 246 pilot study folding intensities where a loop length was repeated. We first observed that when minL was repeated twice, either $\hat{f}(b)$ or $\hat{f}(d)$ was the largest Fourier coefficient. In particular, the NMM intensity of the permuted sequences of (1,1,maxL) was expressed only in terms of $b(\pi)$ and $d(\pi)$, i.e., $f(\pi) = \frac{1}{6}(\hat{f}(\text{triv}) + \text{sign}(\pi) \cdot \hat{f}(\text{sign}) + \sqrt{2}[b(\pi)\hat{f}(b) + d(\pi)\hat{f}(d)])$. For nucleotides T and A, $\hat{f}(b) < 0$ and $\hat{f}(d) > 0$ in more than 70% of the 9 possible combinations of (1,1,maxL), where maxL=2,…,10. The two corresponding terms in the expansion took values of $b(\pi) = 0$ and $d(\pi) = \sqrt{3}$ for id and (1 2), setting the negative effect of $\hat{f}(b)$ to zero and thus promoting folding in the (1,1,maxL) configuration; they took values of $b(\pi) = \frac{\sqrt{3}}{\sqrt{2}}$ and $d(\pi) = -\frac{1}{\sqrt{2}}$ for (2 3) and (1 2 3), making the term $[b(\pi)\hat{f}(b) + d(\pi)\hat{f}(d)]$ negative and thus inhibiting folding in the (1,maxL,1) configuration; and, they had mixed effects in the (maxL,1,1) configuration, taking values $b(\pi) = -\frac{\sqrt{3}}{\sqrt{2}}$ and $d(\pi) = -\frac{1}{\sqrt{2}}$ for (1 3 2) and (1 3) (Figure 3.12). Hence, (1,maxL,1) clearly exhibited lower intensities than (maxL,1,1) or (1,1,maxL) for nucleotides T and A. In contrast, this pattern was absent in sequences with nucleotide C, because $\hat{f}(b) < 0$ and $\hat{f}(d) > 0$ in only 3 out of the 9 cases; in the remaining 6 cases, the configuration (1,maxL,1) was not disfavored. Consequently, no distinct pattern was found in the permuted sequences of (1,1,maxL) for nucleotide C.

Similarly, the permuted sequences of (1,maxL,maxL) generally comprised large positive values of $\hat{f}(a)$ and $\hat{f}(b)$, and negative $\hat{f}(c)$ and $\hat{f}(d)$ with smaller magnitudes. Consequently, the two-dimensional representation $\rho$ had overall positive contributions to the NMM intensity $f(\pi)$ for permutations id, (2 3), (1 2 3), and (1 3), and thus promoted folding in the (1,maxL,maxL) and (maxL,maxL,1) configurations. By contrast, the intensity was negatively effected by $\rho$ for the permutations (1 2) and (1 3 2) and thus inhibited folding in the (maxL,1,maxL) configuration (Figure 3.13).

3.4 Discussion

We have developed a simple model that can explain the GQ folding potential of a large set of sequences. The model is based on studying the distribution
Figure 3.12: Boxplot of linear combination of coefficients obtained from spectral analysis for (1,1,maxL), (1,maxL,1), and (maxL,1,1). The central mark in red is the median, the edges of the box are the 25th and 75th percentiles, and whiskers extend to the most extreme data points not considered outliers. The outliers are marked as red ‘+’. A) Loop composition T, B) Loop composition C, C) Loop composition A.

Figure 3.13: Boxplot of linear combination of coefficients obtained from spectral analysis for (1,maxL,maxL), (maxL,1,maxL), and (maxL,maxL,1). The central mark in red is the median, the edges of the box are the 25th and 75th percentiles, and whiskers extend to the most extreme data points not considered outliers. A) Loop composition T, B) Loop composition C, C) Loop composition A.

of NMM intensity values measured in over 400 putative GQ sequences; this comprehensive sampling spans the potential folding space of loop parameters that cover the generally accepted range of GQ folding sequences. Our results suggest that the most significant composition property that facilitates GQ folding is the minimum loop length. For example, sequences with minimum loop length (minL) of 1 constitute 63% and 97% of the combined and strong
folding populations, respectively, implying that those with minL longer than 1 are not as likely to fold into GQ (Figure 3.8C). This result is consistent with the finding from a recent in vivo study that GQs containing at least one loop length of 1 are preferentially associated with genomic replication errors [76]. Furthermore, there is a significant folding propensity bias among base compositions, with T promoting the highest level of GQ formation. Our computational predictive models based on the order statistics of loop lengths and sequence compositions accurately capture these rules, and cross-validation shows that these models can predict unseen GQ forming sequences with high accuracy.

Our regression model is based on the order statistics of loop lengths and thus assumes that the folding propensity is invariant under the permutation of loop lengths. However, a recent study suggests that having a long middle loop may disfavor folding; specifically, it is shown that the (1,maxL,1) configuration has reduced GQ folding potential compared to the shuffled configurations (1,1,maxL) and (maxL,1,1) [76]. Our NMM data also exhibits slightly diminished intensities for (1,maxL,1) compared to (1,1,maxL) and (maxL,1,1) for nucleotides T and A, but not for C. Similarly, in our experiments, the configuration (maxL,1,maxL) exhibits lower intensities than (1,maxL,maxL) and (maxL,maxL,1) for all nucleotides. These two cases suggest that our model assumption of permutation symmetry may not hold for some GQ sequences and may lead to prediction errors (Figure 3.4B). In order to investigate the impact of rearranging loop lengths on folding potential, one can decompose the NMM intensities into Fourier modes that are basis functions defined on the 6 permutations of (minL, medL, maxL) (explained in our Spectral Analysis section); this approach mathematically characterizes the dominant fluctuating behavior of NMM values on permutation elements (Figure 3.2, Table 3.4). Implementing this analysis shows no consistent pattern for 192 sequences containing unique loop lengths, but uncovers the pattern previously observed for sequences with repeated loop lengths. That is, the Fourier decomposition of NMM intensities identifies two dominant modes that combine to reduce intensity in the (1,maxL,1) configuration for T and A, but not for C nucleotides (Figure 3.12; one-sided unpaired Wilcoxon rank sum test for (1,1,maxL), (maxL,1,1) vs. (1,maxL,1) p-value=7.8 × 10^{-4}, 0.705, 0.003 for T, C, A, respectively). A similar analysis finds reduced folding potential in (maxL,1,maxL) compared to its permuted
configurations for all nucleotides (Figure 3.13; one-sided unpaired Wilcoxon rank sum test for (1,maxL,maxL), (maxL,maxL,1) vs. (maxL,1,maxL) p-value = 0.002 for all T,C,A). However, our data and mathematical analysis clarify that these patterns of reduced folding potential do not generalize to sequences with minimum loop length greater than 1.

We note that the interpretation of our result may be limited by several factors. Even though our two regression models can predict GQ folding propensity with high accuracy, both models have limitations. First, our models, as they currently stand, cannot be directly applied to sequences that contain any guanine bases in a loop, because of the ambiguity in assigning guanines to either a loop or G-tetrads. Second, our models have been validated only on sequences with a single uniform base composition in the loops. For sequences containing more than one type of base, it may require modeling not only the concentration of each nucleotide, but also the specific ordering of the nucleotides. Thus, future research directions include developing a predictive model that can handle sequences with intervening loops consisting of a combination of A, C, G, and T. For such a set of complex sequences, the flexibility of Gaussian process regression will likely provide additional advantages over the linear regression approach. As an important step towards achieving these goals, our work provides a reliable experimental and computational framework that greatly reduces the search space for potential GQ forming sequences and quantitatively predicts the likelihood of folding for a broad range of candidate sequences.
4.1 Background

In Chapter 1 and Chapter 3, we pointed out that the DNA strings do not always preserve their linear double helical structure, but are rather dynamically packaged in the cell. Recent years have witnessed the development of a myriad of new methods for detecting physical interactions between genomic regions based on high-throughout sequencing technologies: examples include the chromosome conformation capture (3C) method [77], and its extension, the Hi-C method [78]. Emerging 3C and Hi-C datasets may enable a paradigm shift in our understanding and modeling of three-dimensional DNA structures, their time dynamics, and roles in transcriptional regulation.

Experimental techniques generally involve cross-linking the interacting DNA and digesting it through specific restriction enzymes (Figure 4.1). Through intramolecular ligation and reverse cross-linking, we obtain a linear strand that contains sequences of interacting regions shown in blue and red. The complex is then sequenced via microarrays or more recently, using high-throughput sequencing. In addition, ChIA-PET (Chromatin Interaction Analysis by Paired-End Tag Sequencing) [79] incorporates chromatin immunoprecipitation based enrichment to analyze genome-wide long-range chromatin interactions, and methods have also been developed to probe RNA-chromatin interactions. Most techniques generate reads that are sequenced from both ends of the ligated fragment, which are then individually mapped to the reference genome. Using the restriction site information, we can count the number of reads that had one pair mapped to loci $i$ and another pair to loci $j$ and the total number be contact frequency. A two-dimensional matrix called a contact map contains contact frequencies of every pair of the genomic region. Among a number of experimental techniques, we focus on
the established Hi-C data.

Figure 4.1: Overview of chromosome conformation capture technology. An interacting pair of DNA strands are cross-linked and digested by restriction enzymes. Subsequently, intramolecular ligation and reverse cross-linking result in a linear strand that contains both sequences of the interacting complex. Through microarray or high-throughput sequencing, we quantify the pairwise contact frequencies. Illustration by Kangyun1985 (Own work) [Public domain], via Wikimedia Commons.

Ideally, the pairwise contact frequency is measured at a base pair level. However, capturing interactions at a fine scale requires extremely deep sequencing which is expensive. Instead, we define resolution to be the bin size of the genome, which is often chosen such that at least 80 percent of the bins have more than 1000 contacts. Hi-C probes can currently achieve resolutions as small as 1 kilobase. Consequently, one faces the challenge of processing, transferring and storing large contact maps. For example, a Human 5 kb resolution genome-wide map can result in 720 GB of data in a plain text format. Given the increasing number of replicates and different conditions, the
amount of data can be overwhelming. Thus, we must represent the contact map in an efficient binary format.

The problem has trade-offs between storage and computation. Clearly, we want to reduce the file size through compression. To this end, we may transform an $n \times n$ matrix into a sparse matrix format and subsequently encode digits in binary. Coding schemes ranging from entropy-based to dictionary-based methods may successfully reduce the file size. However, the raw contact maps are often post-processed and used in downstream analyses. For example, one may normalize the matrix to be doubly stochastic as means of correcting for experimental biases; identification of the topologically associated domains (TADs) may be important; over-expressed long-range interactions need to be called either locally (by comparing neighboring pixels) or globally (by treating every entry in the map equally). Thus, the file format must accommodate fast random access to the data and enable computation in the compressed domain.

To address this new problem, researchers have proposed several solutions, based on the assumption that the contact maps are symmetric and sparse. HiC-pro [80] stores the matrix in a coordinate list (COO), also known as sparse matrix encoding (SME). However, it does not accommodate binary encoding, and results in large file sizes. More recently, Juicebox [81] and Juicer [82] proposed the .hic file format, where the large contact maps are divided into subblocks of varying lengths and only the upper right triangular parts are transformed into compressed sparse row matrix format and subsequently encoded in binary. This approach can accommodate fast random access and guarantees decent storage savings, but only when the matrix is sparse. In addition, binary encoding methods can be further optimized to fit statistical properties of each dataset.

Unlike other genomic data, contact map data formats are currently not standardized. The 4D Nucleome project consortium sponsored by the National Institutes of Health (NIH) is examining various algorithms to evaluate their performances and standardize a format. As a step towards this goal, we have tested existing methods on public data and also proposed our own approach.

Our scheme exploits the unique structure of contact maps: proximal regions are likely to interact, while distant locations are less likely to do so. This leads to dense block substructures placed along the diagonal, and sparse
submatrices appearing at off-diagonal entries. Because contact maps are symmetric, we first cluster the entries of the upper triangular matrix into dense and sparse parts. We subsequently read the dense components using a simple linear space-filling curve and implement suitable encoding methods for the obtained one-dimensional strings such as Huffman and Golomb encoding. The sparse block components are treated differently: we encode only the non-zero values of the submatrices using compressed sparse row encoding.

To allow for fast random access like Juicebox, we subdivided the contact map into 16 blocks of similar sizes and obtained statistics for different datasets, resolutions, and chromosome. Our result indicates that high-resolution data are indeed generally sparse, but sparsity depends on the particular dataset, resolution, and block position. Thus, we propose that each block employ a more suitable approach to accommodate its own statistic.

4.2 Methods

We first analyze chromosome 18 of the mouse embryonic stem cell (mES) Hi-C data generated at 40 kb resolution by Dixon et al. [83] (GSE35156). By plotting the 2270 by 2270 contact map as a heat map and histogram of the entries, we noticed a strong diagonal line indicative of local interactions of neighboring regions known as TADs. Approximately 40 percent of the map were non-zero entries, and they were clustered along the diagonal with values larger than 5 almost exclusively near the diagonal (Figure 4.2). This figure shows that the matrix is dense on the diagonal and sparse elsewhere, suggesting different schemes may be suitable for these two regions.

4.2.1 Storing a sparse matrix

A matrix is typically stored as a two-dimensional array, where each entry in the array is an element $a_{ij}$ of matrix $A$. However, this format requires memory on the order of $m^2$ for an $m$ by $m$ matrix, which is explosive for a large $m$. As a solution to this problem, several formats exist to store large and sparse matrices that contain many zero entries. They all store only the non-zero entries, but store the locations in various ways.

Sparse matrix encoding (SME) / coordinate list (COO) format stores (row,
Figure 4.2: Contact map of chromosome 18 from GSE35156. A) only the entries with nonzero values are indicated in dark color. B) contact map indicating values greater than 5.

column, value) for each non-zero entries, where rows and columns are labeled 0, 1, \ldots, m − 1. For example, given a sparse matrix

\[
A = \begin{bmatrix}
0 & 0 & 0 & 0 \\
5 & 8 & 0 & 0 \\
0 & 0 & 3 & 0 \\
0 & 6 & 0 & 0 \\
\end{bmatrix},
\]

SME stores four triplets: (1, 0, 5), (1, 1, 8), (2, 2, 3), (3, 1, 6). Alternatively, one may organize the triplets into three arrays,

\[
\begin{align*}
\text{rows} &= \begin{bmatrix} 1 & 1 & 2 & 3 \end{bmatrix} \\
\text{cols} &= \begin{bmatrix} 0 & 1 & 2 & 1 \end{bmatrix} \\
\text{vals} &= \begin{bmatrix} 5 & 8 & 3 & 6 \end{bmatrix}.
\end{align*}
\]

Here, SME encodes an m by m matrix A with 3\times\text{NNZ} entries, where \text{NNZ}
is the number of non-zeros. Hence, SME saves space only when

\[ 3 \times \text{NNZ} < m^2 \]
\[ \implies 3 \times (1 - \text{sparsity}) \times m^2 < m^2 \]
\[ \implies (1 - \text{sparsity}) < \frac{1}{3} \]
\[ \implies \text{sparsity} > \frac{2}{3} \]

Sparsity is defined as the fraction of zero entries in the entire matrix A.

However, compressed sparse row (CSR) encoding saves even more space by noticing that row values are always increasing and their largest value is \( m - 1 \). We modify the COO format in the following way: instead of repeating the row label every time a non-zero entry appears in the same row, we essentially perform run-length encoding by counting the number of non-zero elements in the previous row. For example, we repeated rows 1 twice in the previous example. Instead, the modified rows array would be

\[ \text{rows2} = [0 \ 0 \ 2 \ 3 \ 4] \]

of length \( m + 1 \) instead of NNZ. The value and column arrays remain. Hence, CSR encodes an \( m \) by \( m \) matrix A with \( 2 \times \text{NNZ} + m + 1 \) entries.

### 4.2.2 Linear curve along the diagonals

With an observation that diagonal lines contain most of the non-zero values, we separated the upper triangular contact map into dense and sparse parts (Figure 4.3). Letting the width of the dense part \( \tau \) be the point at which we see 60 percent or more zero entries along the diagonal, our \( \tau \) was between 500 and 600. We encoded the sparse part using compressed sparse row encoding and treated the dense part differently. We read the entries through linear curves along the diagonal iteratively going down and up. The subsequent one-dimensional array was encoded in binary via Huffman and Golomb codes described in Chapter 2.
Figure 4.3: Partitioning of the contact map. For contact maps with large values on the diagonal, we separated the map into dense and sparse parts, denoted by D and S, respectively. This figure is chromosome 18 of Dixon et al. mES data at 40 kb resolution.

4.2.3 Enabling random access via partitioning the matrix into subblocks

Subdividing a large contact map into blocks often allows fast random access to a particular data. In our implementation, we divided the upper triangular part into 16 segments, which is equivalent to dividing the matrix into 16 blocks and encoding only the upper right triangular parts (Figure 4.4A). For instance, if we want to access row 700 and column 720, we quickly retrieve that the information is encoded in the 8th block and the search space is now 1/16 of the original matrix. Juicebox also divides the contact map into blocks, but encodes only the upper right triangular part of every block. In Figure 4.4, we show that these two approaches store the same information.

The existing methods HiCPro and Juicebox heavily rely on the assumption that the contact maps are sparse. As seen earlier, sparse matrix formats
Figure 4.4: Partitioning of the contact map into subblocks. A) Our implementation partitions only the upper right triangular parts of the map into 16 blocks. B) Juicebox first partitions the whole map and then encode only the upper right triangular part of each block. Two methods are equivalent.

such as SME and CSR save space only if half or less of the entries are non-zero. This assumption of sparsity needs to be validated before deciding on a data format. Thus, we examine the statistical properties of contact maps of different cell types, chromosomes, and resolutions.

For each block, we count the fraction of entries that are zero and express it as the sparsity $S(X) = \frac{\sum_{x \in X} I(x=0)}{|X|}$, where $X$ is a set of all $N$ entries in the block. Hence, highly sparse matrix would have $S(X)$ close to 1, and dense matrix would be close to 0. Another metric is the entropy $H(X) = \sum_{x \in X'} p(x) \log \frac{1}{p(x)}$, where $X'$ is a subset of $X$ containing only unique elements and $p(x)$ is the probability of observing $x$ in $X$. The entropy not only measures the information content of our data, but can also provide lower bound on the average codelongth needed to losslessly encode $X$.

We then test how much sparsity and entropy affect the file sizes of blocks when encoded in COO, CSR, and dense on the Dixon et al. data. On a larger IMR90 intrachromosomal data chromosome 1 (Rao IMR90 [84]) , we examine the effects of varying resolutions on sparsity and entropy.

All codes in this study are written in Python and MATLAB.
4.3 Results

On three chromosomes of Dixon mES data, chr1, chr10, and chr18, we observed that CSR does reduce the file size by about 30 percent compared to SME (Figure 4.5). Scanning linear curves on the dense part computed with Huffman and Golomb encoding and encoding with CSR on the sparse part saved moderate space, whereas running standard zip on all four file reduced the size by at least 4-fold. We note that the data tested at 40kb resolution has about 40-60 percent non-zeros.

![Figure 4.5: Comparison of file sizes produced by different matrix representations. Three chromosomes from Dixon 2014 data are represented in four different formats: sparse matrix encoding (SME), compressed sparse row (CSR), linear curve Huffman with CSR (LH/CSR), and linear curve Golomb with CSR (LG/CSR). The four represented files are further compressed using zip and file sizes are plotted for each of 8 resulting files for a given chromosome.](image)

This result does save storage, but hinders computational capabilities. All proposed formats require searching through the whole matrix space and furthermore, zipped files need to be decompressed in order to access matrix entries. By contrast, subdividing the matrix allows us to randomly access the entries more efficiently. We observe that most of the blocks are sparse over all three chromosomes, except for block 1, 8, 13, and 16 (Figure 4.6A). These four blocks also do have higher information contents (Figure 4.6B),
requiring more bits to encode losslessly. We note that blocks 1, 8, 13, and 16 correspond to the diagonal upper right triangles in the contact map (Figure 4.6C).

We next tested how much sparsity affects the file sizes of COO, CSR, and Dense formats. In Figure 4.7, we see that chromosome 1 may benefit from coding in dense format whenever the sparsity is less than 0.7 (corresponding to blocks 1, 2, 8, 13, and 16). Likewise, results for chromosomes 10 and 18 also seem to be in agreement that thresholding sparsity at 0.7 would provide more advantages to dense encoding than sparse encoding. Moreover, this result agrees with our previous calculation that in order for COO to outperform dense format, sparsity has to be greater than $\frac{2}{3} \approx 0.7$.

Last, we study the effects of resolution on sparsity and entropy. From Rao IMR90 data, we generated an intrachromosomal contact map for chromosome 1 at 7 different resolutions: 5k, 10k, 25k, 50k, 100k, 1000k, and 2500k. A general trend is that maps of resolution higher than 25kb are dense, and each block contains more information than lower resolution maps (Figure 4.8). Moreover, because centromeric regions are unmappable, we see no contacts in the “cross shaped” regions in the contact map. This property also affects the block sparsity and may be incorporated when designing appropriate format.

### 4.4 Discussion

#### 4.4.1 Summary and immediate goals

In this chapter, we reported several file formats to store a two-dimensional matrix containing contact frequencies of the genome loci. An initial approach treated the contact map as a whole and represented it in SME, CSR, and a combination of linear curve and CSR. Although file sizes were small enough, this first approach does not allow fast retrieval of matrix for downstream processes. To allow random access, we subdivided the matrix into 16 blocks of similar sizes. Calculating the sparsity and entropy of these 16 blocks of diverse datasets revealed that high-resolution data are sparse, but low-resolution maps can be quite dense. Moreover, the location of centromeres and formation of TADs may also affect statistics of the block. Finally, the comparison of file sizes represented in SME, CSR and dense formats illus-
Figure 4.6: Dixon mES statistic. A) Sparsity and B) Shannon entropy are plotted for each of 16 blocks of chromosomes 1, 10 and 18. The locations of blocks are illustrated in C), overlaying on chromosome 18 contact map.
Figure 4.7: Dixon mES block file size. Three matrix representations, COO (blue), CSR (orange), and Dense (gray), are applied to each block of chromosomes 1, 10, and 18. The heights of bars indicate file size in MB according to the left y-axis, and sparsity is plotted in red line, scaled according to the right y-axis. A), B), and C) correspond to chromosomes 1, 10, and 18, respectively.
Figure 4.8: Rao IMR90 block statistics. Intrachromosomal contact map of chromosome 1 of Rao IMR90 data are partitioned into 16 blocks at 7 different resolutions. A) sparsity and B) entropy of each block is plotted for all 7 resolutions. Block locations are illustrated in C).

It is demonstrated that dense format may outperform sparse encoding when sparsity is less than 0.7. The existing tools held a strong assumption of sparsity and implemented SME and CSR. However, our findings suggest that some blocks and resolutions may benefit from dense encoding.

Using these preliminary findings, we can implement a new format both to save storage and allow fast computing. Building on the idea of blocks, we can carefully select varying block lengths to implicitly separate dense and sparse parts and use different matrix encoding formats. For instance, Hilbert space-filling curves on dense parts may preserve locality and compress better than other methods.

4.4.2 Future directions

The future directions are exciting in this area, and concepts from image processing may be adopted. For example, contact maps are generated at multiple resolutions in order to first grasp general domains and then to identify specific long-range interactions between two loci. Progressive encoding stores the low-resolution information before higher resolutions, and it can
accelerate loading time in visualizations. Using rate distortion theory, parts of the contact map can also be quantized to both remove noise and to save storage.

Efficient file formats can then accelerate our understanding of biological phenomena. We can develop algorithms to detect peaks, cluster genomic regions, and integrate them with other genomic data to accompany these powerful experimental techniques.
REFERENCES


