FINDING A NOVEL WAY FOR FAST SEQUENCE ALIGNMENT
AND EXPLOITING INFORMATION THEORY IN BACTERIAL
GENOMES AND COMPLETE PHAGES

by

Sajia Akhter

A Dissertation submitted to the Faculty of Claremont Graduate University
and San Diego State University in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computational Science

Claremont Graduate University and San Diego State University
2013

© Copyright by Sajia Akhter, 2013
All Rights Reserved
Approval of the Review Committee

This dissertation has been duly read, reviewed, and critiqued by the Committee listed below, which hereby approves the manuscript of Sajia Akhter as fulfilling the scope and quality requirements for meriting the degree of Doctor of Philosophy.

Dr. Robert A Edwards, Chair
San Diego State University
Professor, Department of Computer Science

Dr. Barbara Bailey, Member
San Diego State University
Professor, Department of Mathematics and Statistics

Dr. Elizabeth A Dinsdale, Member
San Diego State University
Assistant Professor, Department of Biology

Dr. Ali Nadim, Member
Claremont Graduate University
Professor, Department of Mathematics

Dr. Animesh Ray, Member
Keck Graduate Institute of Applied Life Sciences
Professor, Systems Biology
Abstract

Finding a novel way for fast sequence alignment and exploiting information theory in bacterial genomes and complete phages

by

Sajia Akhter

Claremont Graduate University and San Diego State University, 2013

The invention of next generation sequencing technology (NGS) provides the capability of generating high throughput low cost sequencing data, and is used by scientists to address a diverse range of biological problems. Several data analysis algorithms have been developed in last few years to best exploit NGS data. New tools and methods have also been implemented for better understanding of these data.

This dissertation presents several novel techniques involving NGS datasets. The first technique, qudaich is a novel sequence aligner, which can be used as a key part of NGS data analysis. Qudaich generates the pairwise local alignments of a query dataset against a database. Qudaich can efficiently process large volumes of data and is well suited to the next generation reads datasets. This aligner can also handle both DNA and protein sequences and tries to generate the best possible alignment for each query sequence. In contrast to other contemporary aligners, qudaich is more efficient in terms of execution time and accuracy.

Next, in this dissertation, I show different ways to extract useful genomic information from NGS data, which, in turn, shows promising directions to solve some of the existing biological problems like prophage prediction. Prophages are viruses that integrated into, and replicated as part of, the bacterial genome. These genetic elements can have tremendous impact on their hosts. The majority of other phage finding tools mainly rely on homology-based approach for prophage prediction, which limits the de novo discovery of novel prophages. This dissertation presents a novel algorithm, PhiSpy to predict prophages in bacterial genomes. PhiSpy combines similarity based and composition based strategies to identify prophages. It finds 94% of the known prophages in 50 complete bac-
material genomes with a 6% false negative rate and a 0.66% false positive rate. This led to a successful prediction of the largest set of prophages comparing to other prophage finding applications.

Finally, this dissertation also demonstrates that information theory can be effectively applied to find informative sequences, to predict the lifestyle restrictions of an organism, and to analyze the deviation of the amino acid utilization profile in different metabolic processes in different organisms.

Together, these tools will enable the next generation of sequence analyses using next generation sequence data.
Dedication

To my dear parents and husband
Epigraph

Imagination is more important than knowledge.
Knowledge is limited. Imagination encircles the world.

- Albert Einstein
Acknowledgements

First, I would like to thank my adviser Dr. Robert Edwards for his constant support, guidance and inspiration. You put faith on me and always encourage me. Thank you Rob.

I want to acknowledge all of my committee members for their useful feedback and suggestions. Thank you Dr. Bailey for helping me to learn more statistics. Thank you Dr. Dinsdale for your encouragement and support.

I want to thank all present and former members of Edwards lab. You were always supportive to me and I really enjoyed working with you. I specially like to thank Dr. Ramy Aziz for helping me with any biology related problem.

Finally, I thank my dear parents, brother, sister, and my beloved husband for their love and support.
Contents

Introduction 1

1 Introduction to Sequence Alignment and Current Approaches 5
  1.1 Sequence Alignment ............................................. 5
  1.2 Sequence Alignment Approaches .............................. 6
  1.3 Algorithms for Local Sequence Alignment .................. 6
    1.3.1 Algorithms based on Hash Tables ....................... 7
    1.3.2 Algorithms based on Prefix/Suffix Trees ............... 8
    1.3.3 Sequence Alignment based on Probabilistic Framework 13
  1.4 Concerns of Current Sequence Aligners ................... 13

2 Qudaich: A Smart Sequence Aligner for NGS Data 16
  2.1 Overview of Qudaich .......................................... 16
  2.2 Methodology .................................................. 17
    2.2.1 Identifying the Candidate Database Sequences .......... 18
    2.2.2 Implementation of Identifying the Candidate Database Sequences 25
    2.2.3 Generating the Alignment ................................ 30
  2.3 Result Analysis ................................................ 31
    2.3.1 Testing Datasets ........................................... 31
    2.3.2 Accuracy for Finding the Candidate Database Sequences 32
    2.3.3 Real Time Statistics ....................................... 36
  2.4 Conclusion .................................................... 44

3 PhiSpy: A Novel Algorithm for Finding Prophages in Bacterial Genomes 46
  3.1 Prophages ....................................................... 46
  3.2 Prophage Finding Applications ............................... 47
3.3 Materials and Methods .............................................. 49
  3.3.1 Data Collection ............................................. 49
  3.3.2 Data Analysis ............................................... 49
3.4 Results .............................................................. 55
  3.4.1 Transcriptional Strand Orientation ......................... 55
  3.4.2 Customized AT and GC Skew .............................. 57
  3.4.3 Protein Length .............................................. 60
  3.4.4 Abundance of Phage Words ............................... 61
  3.4.5 Importance of Different Characteristics ................. 62
  3.4.6 Performance Analysis .................................... 65
3.5 Discussion .......................................................... 69
3.6 Acknowledgment ................................................... 71

4 Applying Shannon’s Information Theory to Bacterial and Phage Genomes and Metagenomes 72
  4.1 Information Theory in Genomic Data .......................... 72
  4.2 Results ............................................................. 75
    4.2.1 Factors Influencing Differences in Shannon's Uncertainty .... 75
    4.2.2 Shannon's Uncertainty in Metagenomes .................. 81
  4.3 Discussion ........................................................ 82
  4.4 Methods ........................................................... 85
    4.4.1 Retrieval of Genomic and Metagenomic Data .......... 85
    4.4.2 Calculation of Shannon's Uncertainty .................... 85
  4.5 Acknowledgement ............................................... 86

5 Kullback Leibler Divergence in Complete Phage and Bacterial Genomes 87
  5.1 Kullback Leibler Divergence .................................. 88
  5.2 Results ............................................................. 88
    5.2.1 Divergence of Amino Acid Composition in Genomes ...... 88
    5.2.2 GC Content and Amino Acid Variations among Genomes ... 92
  5.3 Discussion ........................................................ 99
  5.4 Methods ........................................................... 103
    5.4.1 Retrieval of Sequence Data ............................... 103
    5.4.2 Calculation of Kullback-Leibler Divergence ............. 103
  5.5 Acknowledgement ............................................... 104
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Summary</td>
<td>105</td>
</tr>
<tr>
<td>A</td>
<td>Supplementary Material</td>
<td>108</td>
</tr>
<tr>
<td>A.1</td>
<td>PhiSpy</td>
<td>108</td>
</tr>
<tr>
<td>A.2</td>
<td>Kullback Leibler Divergence in Complete Phage and Bacterial Genomes</td>
<td>119</td>
</tr>
<tr>
<td>B</td>
<td>Vita</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>125</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Example of a hash table .............................................. 8
1.2 Different data structures for string CATATA ......................... 11
1.3 Sequencing cost per mega base of DNA sequences and the number of metagenomes sequences in SRA at NCBI ......................... 14

2.1 Order of database suffixes and query suffixes in the suffix array .......... 19
2.2 Alignment between query $q$ and database $d, d_1$ ........................... 21
2.3 Example of the selection of candidate database sequence for query $q$ using hypothesis I and hypothesis II ................................. 21
2.4 The order of the database suffixes in suffix array for the suffixes of query $q$ ................................. 24
2.5 Calculation of the longest common prefix (lcp) between topDB and Qry suffixes of a query group ........................................ 27
2.6 Calculation of the longest common prefix between topDB and the query suffixes using dynamic programming ............................... 28
2.7 The prediction of candidate database sequences using qudaich .............. 33
2.8 BLAST score based on the prediction of candidate database sequences using qudaich ........................................ 34
2.9 Comparison of hypothesis I and hypothesis II for the query sequences that have no good alignment ........................................ 36
2.10 Comparison of the prediction of candidate database sequences for MUM-mer and BWA-MEM ........................................ 38
2.11 Comparison of candidate database sequence prediction for protein sequence alignments using qudaich (hypothesis I) and Usearch .................. 43

3.1 Orientation of proteins in 110 bacterial genomes and 600 phages ............. 56
3.2 Amino acid distribution in the predicted proteins encoded in 41 bacterial genomes and their 190 prophages .................................. 57
3.3 Frequency of codon usage in 41 bacterial genomes with 190 prophages
3.4 Customized AT skew for 41 complete bacterial genomes and their prophages
3.5 Customized GC skew for 41 complete bacterial genomes and their prophages
3.6 Average length of bacterial proteins and phage proteins
3.7 Comparison of the abundance of phage words in bacteria and phage genomes
3.8 Comparative analysis of predicted prophages in 412 complete bacterial genomes by phiSpy, phage_finder and prophinder

4.1 Shannon’s indices of 600 complete phage genomes and 94 complete bacterial genomes
4.2 Length distribution of 600 complete phages
4.3 Shannon’s index vs. length for 600 complete phage genomes using word length 6, 9 and 12
4.4 Shannon’s index vs. GC% for 600 complete phage genomes using word length 1 nt to 7 nt
4.5 The relationship between Shannon’s index and | GC% - 0.5 | for 600 complete phage genomes using word length 1 nt to 7 nt
4.6 Cumulative comparison of the uncertainty (for word length 1) in DNA sequences in metagenome samples
4.7 Comparison of Shannon’s uncertainty and the observed similarity to known sequences

5.1 Trends in amino acid composition divergence
5.2 Amino acid divergence varies for each phylogenetic taxon of bacteria and phage bacterial hosts
5.3 Comparison of the divergence of amino acid composition
5.4 Frequency of each of the twenty amino acids in the three domains of life and the most skewed genomes
5.5 Comparison of KLD and GC-content for all bacterial genomes and for individual groups of subsystems
5.6 Frequency of each of the twenty amino acids for phage genomes
5.7 Comparison of KLD and GC-content for all phage genomes
List of Tables

2.1 Percentage of predicted candidate database sequence matches with BLAST results ................................................................. 35
2.2 Time comparison between hypothesis I and hypothesis II for DNA sequence alignment ......................................................... 37
2.3 Real time statistics of BLAST, MUMmer, Usearch, BWA, Bowtie and qudaich (hypothesis I) .................................................. 37
2.4 Detailed comparison between MUMmer, BWA-MEM and qudaich ...... 39
2.5 Real time statistics for DNA sequence alignment using test set 2 ..... 41
2.6 Real time statistics for protein sequence alignment using test dataset 3 ... 42
2.7 Real time statistics for translated nucleotide alignments using test dataset 1 44

3.1 Performance analysis of PhiSpy and comparison with Phage_finder and prophinder .............................................................. 64
3.2 Effectiveness of different characteristics .................................... 66
3.3 Top ten organisms that have the maximum number of prophages .... 68

5.1 The most skewed bacterial genomes ........................................... 91
5.2 The most skewed phage genomes ............................................. 92
Introduction

The invention of next generation sequencing technology (NGS) [1] begins a new era in all branches of biological research, expanding opportunities to have breakthroughs in the areas like whole genome analysis, epigenetics and metagenomics. NGS produces large amounts of data, e.g., in a single sequencing run, it is possible to produce one terabyte of data (Illumina Inc., http://www.illumina.com, 2012). This capability of generating massive amounts of data in a reasonable time and at low cost opens up new opportunities to address an increasingly diverse range of biological problems. However, efficient analyses of data are the key to investigating these new and challenging problems. Although computer processor speeds improve every generation according to Moore's law [2], the rate of growth of NGS data is outpacing improvements in processor speed, demanding more efficient algorithms and novel techniques to best exploit these data [3–5].

So far there have been lots of research that focus on NGS data. For example, some research areas involve developing tools that can control the quality of these data. Although the NGS platforms have a quality control pipeline, several sequence artifacts like poor quality reads, primer.adapter contamination remain in the data. Therefore, quality control tools are essential for preprocessing, filtering, trimming and checking the quality of the data, and also generating statistics of the datasets. Most of these tools are automatic and NGS platform independent [6–8].

Another interesting area of work mainly concentrates on developing algorithms for
data analysis. There are varieties of applications available depending on the specific problems being addressed. For example, several sequence assemblers have been developed in the last decades that try to merge the fragments of DNA in order to reconstruct the original DNA sequence [9]. To find the sequence similarity or identify conserved protein regions, several sequence aligners have been designed [10]. In addition, there are other kinds of applications like predicting protein structures, sequence translators, base calling, resequencing, RNA expression profiling, genome annotation, etc each of which requires a unique suite of software tools.

This dissertation demonstrates several novel computational techniques involving NGS datasets. First, a new sequence alignment algorithm has been developed that is specifically designed for NGS datasets, and can be used as a key part of NGS data analysis. Second, we show different ways to extract useful genomic information from NGS data, which opens up alternative approaches to solve some of the existing biological problems. As an application of this, a novel algorithm has been developed to predict unique and unidentified prophages in bacterial genomes.

**Sequence Alignment** Sequence alignment is an important step to analyze and annotate new genomes or metagenomic sequences produced by NGS technology. It is also necessary (either at the DNA/protein level) to make a similarity profile between different genetic samples in different environments. Moreover, to study the common bacterial or viral populations within several metagenomes, sequence alignment is the first step. As the number of metagenomes and other genomic data has been increasing each year, there is always a demand for faster sequence alignment algorithms. This dissertation proposes a new local sequence alignment tool that focuses on NGS data and gives better performance than contemporary tools.

**Prophage Prediction** Another key contribution of this dissertation is to develop a novel way to predict prophages in bacterial genomes. Several thousand bacterial genomes
are available in public databases largely because of the advances of NGS technology. These genomes are well annotated. But the prophages, integrated viral genomes, in bacteria are often overlooked in the process of microbial genome annotation. Prophages are very dynamic and responsible for the majority of horizontal gene transfer between species. These mobile elements can have tremendous impact on their bacterial hosts genomes and phenotypes, which leads to strain emergence and diversification, increased virulence, and antibiotic resistance. However, finding prophages in microbial genomes remains a problem with no definitive solution. The majority of existing tools rely on detecting genomic regions enriched in protein-coding genes with known phage homologs, which hinders the de novo discovery of phage regions. In this dissertation, we design a phage detection algorithm that uses distinctive genomic characteristics along with similarity based approach for a successful prediction of a larger set of prophages including novel and unique prophages.

**Information theory in genomic data** Any genomic data contains inherent information that can be traced and quantified by applying mathematical models and theories. In the third part of this dissertation, we explore how information theory can be effectively applied to evaluate large datasets of bacterial genomes, complete phages, and metagenomes. These analyses can be effectively used to prioritize sequence analysis and annotation, thus saving computational resources.

This dissertation is organized into five main chapters. Chapter 1 introduces different approaches of sequence alignment and describes the key characteristics of these algorithms. Chapter 2 presents the detailed description of the Qudaich sequence aligner. This chapter compares qudaich to current aligners.

Chapter 3 demonstrates how we can use genomic characteristics to design better algorithm for prophage prediction. This chapter describes the algorithm in details and also compares, and analyzes the predicted prophages with contemporary tools.

Chapter 4 discusses how we can apply the information theory to study the information
content within the genomic sequence data. We show that the information content correlates with genomic characteristics and can be effectively applied for finding informative sequences.

Chapter 5 demonstrates that how information theory can be useful to predict the lifestyle restrictions of an organism, skewed amino acid utilization profile between different organisms, and the divergence of amino acid composition in different metabolic processes.

Finally, chapter 6 summarizes the contribution of this dissertation.
Chapter 1

Introduction to Sequence Alignment and Current Approaches

1.1 Sequence Alignment

Sequence alignment is a way to arrange one sequence against another sequence so that they can express the similarity between themselves. This is an important step in many biological problems because the similarity between two biological sequences infers how the sequences are related to each other. According to the theory of evolution, two organisms that evolve from the same ancestor always share some level of sequence similarity because of the conservative behavior of nature, which enforces slow and incremental modifications in genomic sequences. Hence, sequence alignment is essential to define the source of a new organism, to identify the consequence of functional and structural relationships between different genomes, or to find the ancestral correlation between organisms.

Many sequence alignment tools have been developed in the last couple of decades. There are sensitive local alignment tools like BLAST [11] and FASTA [12] that were developed in the 1990’s. A number of sequence alignment methods like SSAHA2 [13], BLAT [14], MUMmer [15–17] have been published since 2000 that focus on speed. Sev-
eral new, fast, and memory efficient algorithms have been developed since 2008 because of the next-generation sequencing technologies that generate millions of short sequence reads. These tools include BWA [18, 19], Bowtie [20], Maq [21], Soap [22] and they mainly focus on aligning short read sequences against long reference sequences. BWA-SW [19] can align comparatively long read against long reference sequences, but it is designed for only DNA sequence alignments.

1.2 Sequence Alignment Approaches

For sequence alignments, computational approaches fall into two categories: global alignments and local alignments. Global alignments aim to align every residue in the query sequence, which causes the alignment to span the entire length of the query sequence. Global alignment is useful when the reference and query sequences are evolutionarily closely related. On the other hand, local alignments concentrate on identifying the highly similar regions between two sequences regardless of the length of the sequences. Local alignment is most useful for searching for conserved domains or local similarities in large sequences and works well for more diverse sequences.

1.3 Algorithms for Local Sequence Alignment

Many approaches are available for local sequence alignments. For local alignments, the Smith-Waterman algorithm [23] (a dynamic programming approach) is guaranteed to give an optimal alignment solution for a set of scoring parameters. However, this method is slow for a large number of sequences as its time complexity is $O(n^2)$ to align two sequences of length $n$. This is the lower bound to get the optimal local alignment, which means it is not possible to produce an optimal alignment better than this time complexity. To gain performance, several heuristic methods were developed, which give close to optimal align-
ment but are significantly faster than dynamic programming. The fundamental hypothesis of these algorithms is that in the alignment there has to be an exact match of some length. Based on this idea, these heuristic algorithms can be grouped broadly into two categories: algorithms based on hash tables and algorithms based on suffix/prefix trees [10].

1.3.1 Algorithms based on Hash Tables

The hash table is a data structure that stores values against keys and can retrieve the value in a constant time for a given key [Figure 1.1]. There are several popular applications for local sequence alignments that use hashing techniques like BLAST [11], FASTA [12], SSAHA [13], BLAT [14], MAQ [21], SOAP [22], BFAST [24] and RAMP [25]. Algorithms based on hash tables have two main steps: searching the seed and then extending that seed. In the first step, a list of short subsequences, called seeds or k-mers, of query sequences are stored in the hash table and then the database sequences are scanned by looking up the hash table for the exact k-mer match. Different tools use different k-mer length e.g., the default parameter of BLAST uses consecutive matches of 11 nucleotides for DNA alignments, while some other tools (like SOAP, MAQ) consider spaced seeds that allow a seed with non-consecutive matches. After finding all the exact matches, each seed is extended on both sides to get the alignment in the second step. However, the extension part also requires $O(n^2)$ time [17]. Therefore most of the tools that use long spaced seeds do not extend the seeds for short read alignments because the spaced seeds cover the whole read; this makes the search much faster. For better accuracy, to extend and join the seeds, BLAST applies the Smith-Waterman algorithm to refine them. To speed up, SSAHA2 and BLAT normally store multiple seeds matched for each extension and this strategy offers faster alignments that BLAST when the query and database are relatively identical. Another approach, used in FANGS [26] is to filter out the unnecessary regions by applying q-gram filtering and the pigeon hole principle. This idea efficiently works for sequences with an error rate less than
or equal to 1%.

Figure 1.1: Example of a hash table for string CATATA

\[
\begin{array}{c|c}
\text{words} & \text{hash function} & \text{indices} \\
\hline
\text{CAT} & 00 & 1, 3 \\
\text{ATA} & 01 & 0 \\
\text{TAT} & 02 & 2 \\
\end{array}
\]

\text{String} = \text{CATATA, Word length} = 3

The memory requirements of hash-based approaches directly depend on the size of the dataset used to make the hash table and the size of the hash length. As the nucleotide sequences contain repeats and are very unlikely to have each possible combination of nucleotides, hash tables are more appropriate for DNA sequences, although depending on the dataset, the hash table might require a large amount of memory. For example, to align two different strains of \textit{E. coli} (4,639,221 bp and 5,528,445 bp), the hash table contains \(4.99 \times 10^7\) \(k\)-mers \((k = 10)\) and requires \(1.66 \times 10^7\) comparisons to find all the maximal matches of length \(\geq 20\) [15].

1.3.2 Algorithms based on Prefix/Suffix Trees

Algorithms in this category mainly rely on suffix and prefix trees [27], suffix array [28] and Burrows-Wheeler transform/FM index [29]. A suffix/prefix tree is a data structure that stores all the suffixes/prefixes of a string. This tree based data structure is widely used for finding the longest common substring between two sequences. To build a suffix or a prefix tree for constant size of alphabet, it takes \(O(n)\) time where \(n\) is the length of
the sequence. It also takes linear time to traverse the tree for finding the longest common suffix/prefix. As this data structure can facilitate the longest-prefix/suffix matching in linear time, whereas, in a hash table, the exact match depends on the seed length, prefix and suffix trees have become more popular than hash based methods. Also, unlike hashing, in tree-based methods, identical suffixes or prefixes collapse on the same path in the tree, reducing the memory requirements.

The space complexity of a suffix tree is linear, but the efficient implementation of suffix tree requires about 17 bytes per nucleotide [30]. This implementation is known as simple linked list implementation (SLLI). In this technique, two tables are required to represent the suffix tree: one for leaf nodes and one for branching nodes [31]. Each leaf node stores a pointer for its right sibling, which requires 1 integer (i.e. 4 bytes). Each branching node stores its first child, right sibling, depth of the node itself, and suffix link, which requires 4 integers, i.e. 16 bytes. For a string of size \( n \), there are \( n \) leaf nodes and \( q \) branching nodes. Therefore, total space requirement is \( 4n+16q \) bytes. On average, \( q \) is less than 0.8\( n \), which makes the total space requirement about 17\( n \) bytes [32]. However, in the worst case, the space complexity is 20 bytes per base pair. Hence, an improved linked list implementation (ILLI) was proposed [31]. This technique splits the branching nodes into two groups: large nodes and small nodes. Each large node requires 5 integers, whereas each small node requires 3 integers. The result is that if there are many small nodes in the suffix tree, the ILLI saves more space than SLLI.

MUMmer was one of the first aligners based on suffix trees and can align both DNA and protein sequences. The first version of MUMmer 1.0 [17] generates a suffix tree using both reference and query sequences to find the maximal unique matches. As a suffix tree demands a large amount of memory, the later version of MUMmer (version 2.1 and 3.0) [15, 16] builds the suffix tree only for reference sequences and then searches the tree with query sequences to find the maximal unique matches [16]. For an alignment, it joins the
gaps between the exact matches. Among published alignment tools, MUMmer is very
efficient in terms of time, but is not sensitive enough. However, in MUMmer, there is an
option to get sensitive alignments, but that requires more time. MUMmer requires 15.4
bytes per bp to construct the suffix tree for human chromosome 2, but for DNA sequences
less than 134 Mbp in length, the suffix tree can be constructed using 12.5 bytes per bp [15].

The efficient implementation of a suffix tree demands a large amount of memory to
build the tree for large genomes like the human genome. To reduce the memory size,
Manber and Myers proposed the idea of the suffix array [28]. A suffix array contains all
the suffixes of a string in lexicographical order. Let \( X = X_1 X_2 \cdots X_n \) be a string of length
\( n \) and the suffixes of \( X \) are in the form of \( X_i = X_i X_{i+1} \cdots X_n \) where \( i = 1, 2, \cdots , n \). The
\( k \)th position of \( SA \), i.e, \( SA[k] \) contains the index of the \( k \)th lexicographically smallest suffix
of \( X \). That means, in the suffix array \( SA \), the suffixes of \( X \) maintain the following order:
\[
X_{SA[1]} \leq X_{SA[2]} \leq X_{SA[3]} \leq \cdots \leq X_{SA[n]}.
\]

Compared to the suffix tree, the suffix array construction algorithm requires 8 bytes per
nucleotide (one integer for the suffix array itself and one additional integer of extra space for
construction). Hence, for a string of size \( n \), it requires \( 2n \) integers. The time complexity of
this algorithm is dominated by the complexity of the sorting, \( O(n \log n) \) [28]. Later, several
studies showed that a direct linear implementation of the suffix array is possible [33–36].
In 2002, Hon and Sadakane proposed an algorithm, which uses a query sequence \( S_1 \) and a
reference sequence \( S_2 \) together to build a suffix array and showed that it is possible to find
the maximum unique matches between \( S_1 \) and \( S_2 \) in close to linear time [37]. However,
their proposed algorithm does not show how to find all the maximal unique matches when
there are multiple query sequences and multiple reference sequences. Using enhanced
suffix array, Segemehl [38] (a DNA sequence aligner) maps short sequencer reads against
a reference genome by allowing mismatch, insertions and deletions.

In 2000, Ferragina and Manzini proposed a space and time efficient data structure
known as an opportunistic data structure, for indexing and searching that combines the
Burrows-Wheeler-transform algorithm (BWT) and the properties of suffix array [39].

Figure 1.2: Data structure for string CATATA. A) Prefix tree. B) Suffix tree. C) Suffix array
and Burrows-Wheeler-transform

To transform a string using BWT, first all the rotations of the input string are sorted
in lexicographical order (suffix array), and then the last character of each sorted rotation
is taken. For example, the string ‘CATATA$’ is transformed into ‘ATTC$AA’ by BWT in
figure 1.2. In this data structure, Ferragina and Manzini introduced the FM-index, which
stores the BWT transformation of a string in a compressed form and allows a backward
search in the suffix array to find a substring [39]. They showed that the time complexity for
finding a substring with FM-index is the same as finding the exact matches in a prefix tree and much faster than hash based approach.

BWA, BWA-SW and Bowtie are based on the FM-index and can only align DNA sequences. They are time and space efficient aligners. BWA and Bowtie are mainly designed to align short reads against long reference sequences like the human genome. In BWA, the reference sequence is used to construct a suffix array and then the BWT is generated from the suffix array. For exact matching, BWA uses the FM index and for inexact match it uses a backward search by allowing few mismatches. Bowtie also follows the same strategy as BWA. The main limitation of these BWT-based methods is that they can only find the alignments within a certain edit distance of the sequence in the reference sequence where edit distance normally represents the number of mismatches or gaps in the alignment [40]. This is problematic for sequences with high error rates, such as, output from some of the next generation sequencing machines. As novel sequencing technologies are trying to decrease the sequencing error rate, this limitation will be less significant when aligning against a long reference genome.

BWA-SW is mainly designed for long read alignments and it combines the BWT based approach and Smith-Waterman algorithm. It constructs an FM index for both reference and query sequences, and mainly represents the reference sequence as a prefix tree and the query sequence as a prefix directed acyclic graph [19]. It allows spaced seeding and finds the seeds using dynamic programming in the FM-index. To extend the seed, it restricts the dynamic programming around the matches with better scores by pruning the low scoring matches at each node. As the best alignments normally have a high alignment score, at each node, BWA-SW only progresses with the top Z best scoring nodes in the prefix tree. This is called Z-best strategy [19].
### 1.3.3 Sequence Alignment based on Probabilistic Framework

An alternative approach for sequence alignment is to use the probabilistic inference methods based on profile hidden Markov models [41]. The main difference of this approach over the standard methods of sequence alignment is that it can characterize an entire family of sequences by allowing position-dependent character distributions and position-dependent insertion and deletion gap penalties.

Algorithms based on probabilistic frameworks usually have two steps. First they align a family of similar sequences using multiple sequence alignment and construct a profile-HMM model from this alignment. This HMM model provides complex position specific comparison values and gap penalties. In the second step, the query sequence(s) are compared to this profile-HMM. If the comparison is better than the comparison between a null model (e.g., random sequence) and the profile-HHM, the query sequence(s) are considered to be homologous to the sequences that were used to build the profile-HMM.

SAM [42], PSI-BLAST [43], and HMMER [44], HHblits [45] follow the profile comparison approach for aligning sequences. Compared to the sequence aligners based on other heuristics approaches (described in section 1.3.1 and section 1.3.2), these aligners are computationally time-consuming. Using heuristics, HMMER 3 recently achieved similar time performance as BLAST. However, HMMER 3 approach actually provides better performance for aligning diverse sequences.

### 1.4 Concerns of Current Sequence Aligners

Most of the sequence aligners developed in the last few years concentrate on aligning next generation sequencing datasets against a long reference genome. None of the aligners focus on generating alignments between the datasets created by NGS. For pairwise comparisons between any two datasets produced by NGS, general-purpose alignment tools like Usearch
or BLAST are more effective. However, the total time required for these comparisons is still a great concern. Some of the fast sequence aligners (like Bowtie, SOAPaligner, BWA-short), which align short reads against long reference sequences might be useful for this purpose, but it is not guaranteed that these aligners can efficiently handle a database containing hundreds of thousands or millions of reads instead of couple of long reference sequences. Also, the next generation sequencing technologies are trying to produce longer read lengths. For example, Illumina/Solexa has already increased to >250 bp from 50 bp and Roche/454 has increased to >500 bp [47–49]. Some of these short read sequence aligners have been updated for longer read length [19, 50], but it is noticeable that within next couple of years, aligners designed for longer read alignment will be more applicable and useful.

Figure 1.3: Sequencing cost per mega base of DNA sequences and the number of metagenomes sequences in SRA at NCBI. The sequencing cost has dropped drastically after 2007 because of the invention of high throughput low cost sequencing technology. The sequencing cost is inversely proportional to the increment of the number of metagenomic datasets.

Another vital concern of current aligners is that most of the fast sequence aligners
mainly produce DNA sequence alignment, but protein sequence alignments are also essential. Protein sequence alignments are used to determine the conserved portion of a sequence in protein level. In protein sequence alignments, there are three possible alignment options, i.e, (i) protein vs. protein alignments (ii) protein vs. translated DNA alignments, and (iii) translated DNA vs. translated DNA alignments. Only BLAST supports all three types of protein sequence alignments. Usearch supports (i) and (ii), and MUMmer supports only (iii). Therefore, for protein sequence aligners, there are still opportunities for improved alignments algorithms that can provide all types of protein sequence alignment options with lower time complexity.

The execution time of current aligners is another great concern. There are large volumes of sequences available in public databases and the number has been increasing over time. For example [Figure 1.3], in the Sequence Read Archive at NCBI [51], there were about 100 publicly available metagenomes in 2008 and by 2012, there were about 2500 publicly available metagenomes; each of those were generated by different technologies and are of different read lengths. To annotate and analyze these large amounts of data, sophisticated tools are required so that these tools can handle these data efficiently. Therefore, there is always a strong demand for faster sequence alignment algorithms.
Chapter 2

Qudaich: A Smart Sequence Aligner for NGS Data

Qudaich (queries and unique database alignment inferred by clustering homologs) is a new sequence aligner that introduces a novel approach to the sequence alignment problem. In this chapter, we discuss qudaich in detail. This chapter is organized as follows: section 2.1 describes the overview of qudaich. Details of our methodology are presented in section 2.2 and section 2.3. Section 2.4 examines the accuracy and the real time statistics of qudaich. Finally, Section 2.5 concludes the work.

2.1 Overview of Qudaich

Qudaich generates the pairwise local alignments between a query dataset against a database. A database usually contains the reference sequences to which we compare the query dataset containing the sequences of interests. The main design purpose of qudaich is to focus on datasets from next generation sequencing. These datasets generally have hundreds of thousand sequences or more, and so, the input database contains large number of sequences. Qudaich handles such databases efficiently.
Qudaich is flexible and its algorithmic structure imposes no restriction on the absolute limit of the acceptable read length, but the aligner gives better performance for read lengths $< 1,000$ base pairs. Qudaich can be used to align both DNA and protein sequences. It can align DNA vs. DNA, protein vs. protein, translated DNA vs. translated DNA, protein vs. translated DNA, and vice versa.

2.2 Methodology

Qudaich performs local sequence alignments in two major steps - i) identifying the candidate database sequence(s) and ii) generating the optimal alignment with those candidate database sequences.

In the first step, qudaich tries to find the candidate database sequence(s) for each query sequence. Here, for a query sequence, $q$, the candidate database sequence refers to the corresponding database sequence, $d$ that gives either the best alignment or close enough to the best alignment with $q$. Thus, if $q$ is aligned against all the database sequences, the alignment score between $q$ and $d$ will be either the best score or close enough to the best score. Qudaich mainly focuses on NGS datasets and such database contains huge number of sequences, therefore any nave approach to find the candidate database sequences are impractical. Qudaich applies heuristics to limit the search space to find the candidate database sequence efficiently. After identifying all the candidate database sequences, in the second step, the optimal alignment is generated for each query sequence with the corresponding candidate database sequence. The Smith-Waterman-Gotoh [52] algorithm is used for this purpose.

The approach taken by qudaich has a key advantage over most of the contemporary aligners that are based on suffix tree or hash based approaches. These aligners normally consider a seed (a match of some length) to generate the alignment, but suffer from the
problem that the optimal alignment may not contain that particular seed. Qudaich, on the other hand, does not apply heuristics to generate the alignments. It only applies heuristics to find the candidate database sequences.

2.2.1 Identifying the Candidate Database Sequences

Qudaich uses a novel algorithmic structure to look for the candidate database sequences efficiently. Most aligners keep the database and query sequences separate, but in qudaich both database and query sequences are organized together. This novel organization has two main advantages. It accelerates the searching of candidate database sequences by clustering several query sequences; and, it allows us to construct powerful heuristics to limit the search space. However, the primary disadvantage of using the query and database sequences together is that the sequence indices need to be rebuilt for every different comparison.

Next, the data organization of qudaich is described and after that the heuristics have been explained in details.

Primary data structure

Qudaich uses a suffix array as the primary data structure. It constructs a single suffix array using all database and query sequences. Among the several approaches or data structures used in the sequence alignment algorithms described earlier, the suffix array is one of the most efficient ways to store or organize the sequences. In 2008, Ge Nong el al [53] provided two efficient algorithms for constructing suffix arrays and later, Yuta Mori (https://sites.google.com/site/yuta256/sais) provided a faster implementation of the suffix array construction library based on those algorithms. This implementation is both time and space efficient [it runs in $O(n)$ and requires $MAX(2n, 4k)$ extra working space where $n$ is the length of the string and $k$ is the size of the alphabet in the string]. We adopted this library
to construct the suffix array.

Suffix array organization in qudaich

In our algorithm, a long string, $S$ is constructed by concatenating all database sequences, the reverse complement of database sequences and then all query sequences are concatenated. Then using $S$, a suffix array, $SA$ is constructed.

![Figure 2.1: Order of database suffixes and query suffixes in the suffix array. The red boxes show the two database suffixes, one of which has the longest match with the query suffixes marked as light blue](image)

There can be two types of suffixes in a $SA$, because all the database sequences and query sequences are concatenated in $S$. Some suffixes start either a query sequence or with a part of a query sequence. These are called $Qry$ suffixes. The rest of the suffixes start with either database sequences or with a part of the database sequences or with the reverse complement of the database sequence. These suffixes are considered as $DB$ suffixes. In $SA$, all consecutive $Qry$ suffixes construct a $Qry$ group. Similarly, all consecutive $DB$ suffixes make a $DB$ group, where each group contains one or more suffixes. As shown in the figure
2.1, the suffix array, \( SA \) contains a sequence of \( Qry \) groups and \( DB \) groups. So, \( SA \) will be a sequence of alternating \( Qry \) groups and \( DB \) groups.

**Algorithmic Overview**

All the suffixes in the suffix array stay in sorted order. Based on this characteristic and the arrangement of \( Qry \) groups and \( DB \) groups, it is possible to find the longest exact match for all the query suffixes with the database suffixes.

Let's assume a \( Qry \) group occurs in position \( SA[i] \) to \( SA[j] \) in the suffix array \( SA \) where \( i \leq j \). The best match of all the \( Qry \) suffixes of this group will be the immediately prior \( DB \) suffix (\( topDB \)), \( SA[i-1] \) or/and the immediately subsequent \( DB \) suffix (\( bottomDB \)), \( SA[j+1] \) of the group, because all the suffixes in \( SA \) are in lexicographically sorted order. Therefore, only two \( DB \) suffixes need to be considered for each group of query suffixes instead of all \( DB \) suffixes. This statement holds for all the \( Qry \) groups in the suffix array. So the search space is limited to all the \( Qry \) groups and their surrounding two \( DB \) suffixes. Note that, in the suffix array construction, the reverse complement of the database sequences is used instead of using the reverse complement of query sequences to minimize the number of \( Qry \) groups will be minimized and reduce the processing time.

The suffix array data organization in qudaich finds the best exact match for each query suffix. This property can be further exploited to construct two different hypotheses that can be used to find the candidate database sequences.
Figure 2.2: Alignment between query $q$ and database $d, d_1$. Suffixes of $q$ and $d$ start at positions 1 4 and 6 8 have the common prefix match (of length at least one) between them. The common prefix match between suffixes of $q$ and $d_1$ start at position 1-2, 4 and 6-8. The query sequence $q$ has better alignment with $d$ and the number of common prefix match between $q$ and $d$ is greater than the number of common prefix match between $q$ and $d_1$.

Figure 2.3: Example of the selection of candidate database sequence for query $q$ using hypothesis I and hypothesis II. Both hypotheses select $d_1$ as candidate database sequence for $q$.  

<table>
<thead>
<tr>
<th>Query Group 1</th>
<th>Query Group 2</th>
<th>Query Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>$d_1$</td>
<td>$d_1$</td>
</tr>
<tr>
<td>ACCGGCGTGA...</td>
<td>GGTGAAAC...</td>
<td>CCAACACCA...</td>
</tr>
<tr>
<td>ACCGGCGTGC...</td>
<td>GGTGCAAC...</td>
<td>CCAACACCA..</td>
</tr>
<tr>
<td>ACCGGCAGG...</td>
<td>GCGAGGAA...</td>
<td>CCAACGCT..</td>
</tr>
<tr>
<td>ACCGGGGGA...</td>
<td>GGGGATAC...</td>
<td>CCAACCGT...</td>
</tr>
<tr>
<td>$d_2$</td>
<td>$d_2$</td>
<td>$d_3$</td>
</tr>
<tr>
<td>ACTCTCAACG...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hypothesis I
- $\text{frequency}(q,d_1) = 3$
- $\text{frequency}(q,d_2) = 2$
- $\text{frequency}(q,d_3) = 1$

Hypothesis II
- $\Sigma \text{lcp}(q,d_1) = 4 + 1 + 6 = 11$
- $\Sigma \text{lcp}(q,d_2) = 2 + 4 = 6$
- $\Sigma \text{lcp}(q,d_3) = 1$
Hypothesis I: Query q has the best alignment with database sequence d, if suffixes of d are the most frequent closest suffixes in all the query groups containing all the suffixes of q.

This hypothesis is validated by the following facts:

i. If q (of length n) has the best alignment with d (of length m), the number of the common prefix match (of length at least one) between the suffixes of q and the corresponding suffixes of d will be the maximum among the number of the common prefix match between the suffixes of q and the suffixes of any other database sequence. Considering the simplest example, if n = m and there is only one mismatch between q and d, then n-1 suffixes of q and d have the common prefix match of at least one length [Figure 2.2].

ii. Since all the suffixes are lexicographically sorted in SA, in most of the cases suffixes of d (that have some match with suffixes of q) will be either topDB or bottomDB of the Qry groups containing the corresponding suffixes of q.

Hence, according to hypothesis I, the candidate database sequence of q is the database sequence, whose suffixes are the most frequent among all topDB or bottomDB of all suffixes of q [Figure 2.3].

Hypothesis II: Query q has the best alignment with database sequence d, if hypothesis I satisfies and ∑lcp(suffixes of q, suffixes of d) is maximal.

For hypothesis II, a weighted frequency is used. Instead of considering the presence of the DB suffix as topDB and bottomDB, the longest common prefix (lcp) between the query suffix and the DB suffix is also measured. This hypothesis is validated by the following fact:
i. If $q$ has the best alignment with $d$, in their alignment, the % identity and % coverage will be the maximum comparing the alignment between $q$ and any other database sequence. This indicates that the summation of the length of longest common prefix between the suffixes of $q$ and the corresponding suffixes of $d$ will also be the maximum.

Other possible hypotheses

The hypotheses described above mainly use the $topDB$ and $bottomDB$ to find the candidate database sequences. However, this might not give the best accuracy for the following reason. For convenience, we demonstrate the concept assuming that there is a query sequence $q$ and we have the longest exact matches for each suffix of $q$ (that is with $topDB$ and $bottomDB$).

As shown in figure 2.4, it might happen that a suffix of $q$, say $SA[k]$ in a $Qry$ group from $SA[i]$ to $SA[j]$ where $i \leq k \leq j$, has the longest match with $topDB$ at $SA[i-1]$ and the match length is $a$. But $SA[i-2]$ (which is a DB suffix) also has a match up to length $a$ with $SA[k]$. However, the database sequence (say $d_1$), whose suffix is at $SA[i-2]$, has a better alignment with $q$. Since $SA[i-2]$ was not considered in hypothesis I and II, $d_1$ might not be predicted as a candidate database sequence. In the ideal case, more than one DB suffix in the DB group needs to be considered for the all query suffixes in the corresponding query group. Hence, another possible hypothesis is discussed here, which actually incorporates other facts in hypothesis I and hypothesis II. Considering these facts might give better accuracy for finding candidate database sequences at the cost of increased run time. However, in qudaich, only hypothesis I and hypothesis II were implemented.
Figure 2.4: The top table shows the information of each query suffix of query $q$ after finding the longest exact matches and the bottom table shows the ambiguity order of the suffix array. The query $q$ has a match with DB 10001 from position 1 to 100 and there is a mismatch at position 101 and then there is a match from position 102 to 151. The position 102 and position 103 shows that suffixes start from these positions have match with DB 10031 instead of DB 10001. This happens because the suffix that matches at position 102 of query $q$, is identical up to 150 bp with both DB 10031 and DB 10001.
Possible hypothesis III: In this hypothesis, more than one DB suffix will be considered while calculating hypothesis I and hypothesis II. For a query group $SA[i]$ to $SA[j]$, where $i \leq j$, both hypotheses I and II consider only topDB at $SA[i-1]$ and bottomDB at $SA[j+1]$. In this hypothesis, $x$ DB suffixes will be considered from both top and bottom DB group of the query group. That means instead of just considering topDB at $SA[i-1]$ and bottomDB at $SA[j+1]$, $SA[i-1]$ to $SA[i-1-x]$ will be considered as topDBs and similarly, $SA[j+1]$ to $SA[j+1+x]$ will be considered as bottomDBs.

Initial empirical testing has suggested that appropriate values for $x$ are less than 5, and thus the additional overhead for this calculation does not appear to benefit the algorithm. However, if it is possible to design an ideal hypothesis, which picks the candidate database sequences that always gives the best alignment, qudaich will always generate a 100% accurate alignment. This is the key benefit of the algorithmic approach of qudaich.

In qudaich, hypotheses I and II were implemented to find the candidate database sequences. The implementation details of these hypotheses are discussed below.

### 2.2.2 Implementation of Identifying the Candidate Database Sequences

Two different procedures have been followed in the implementation of hypothesis I and hypothesis II. Detailed descriptions of the implementation of both hypotheses are described.

#### Implementation of Hypothesis I

To find the candidate database sequences using hypothesis I, the suffix array needs to be traversed only once. While traversing $SA$, the following steps were followed.

1. For all $Qry$ suffixes of each query group, store the corresponding $topDB$ and $bottomDB$.

2. For each query sequence, sort the frequency of the database sequences.
3. Consider the most frequent database sequence as the candidate database sequence for each query sequence.

Suppose there are \( r \) query sequences of length \( n \). The above procedure requires \( O(r \cdot x \log(x)) \) time where \( x \) is the number of different database sequences which appeared as \( topDB \) or \( bottomDB \). \( x \) will be less than or equal to \( 2n \) because there are \( n \) suffixes for each query sequence and for each query suffix two \( DB \) suffixes are stored. In the worst case, when there is no good alignment for a query sequence, \( x \) will be close to or equal to \( 2n \). Otherwise, \( x \) should be small. Thus in the worst case, the time complexity will be \( O(r \cdot n) \) while in the best case it will approach \( O(r) \). The space requirement will be \( O(r \cdot n) \).

Implementation of Hypothesis II

For this hypothesis, the longest common prefix (lcp) between two suffixes needs to be calculated. A lcp array holds the lcp information of two consecutive suffixes. That means, \( lcp[i] \) holds the length of the longest common prefix between \( SA[i] \) and \( SA[i-1] \), \( lcp[i+1] \) contains the lcp between \( SA[i] \) and \( SA[i+1] \) and so on, where \( 1 \leq i \leq \text{length of } SA \). It is possible to construct the lcp array while constructing the suffix array. Using Yota Mori’s suffix array implementation, Johannes Fischer efficiently computed the lcp array. This implementation outperforms other lcp array construction algorithms. Therefore, we adopted this library for hypothesis II [https://github.com/elventear/sais-lite-lcp].

We need to find the lcp for all the suffixes in each query group with their \( topDB \) and \( bottomDB \). But the lcp array only provides the lcp between two consecutive suffixes. Therefore, a dynamic programming approach was designed for this purpose. The lcp between a query suffix at position \( i \) and a \( DB \) suffix will be computed using the following recurrence relation:
\[\text{lcp (Qry}_i\text{ and topDB)} = \min[\text{lcp(Qry}_i\text{ and Qry}_{i-1}), \text{lcp(Qry}_{i-1}\text{ and topDB})]\]

\[\text{lcp (Qry}_i\text{ and bottomDB)} = \min[\text{lcp(Qry}_i\text{ and Qry}_{i+1}), \text{lcp(Qry}_{i+1}\text{ and bottomDB})]\]

<table>
<thead>
<tr>
<th>Match with topDB</th>
<th>lcp</th>
<th>topDB</th>
<th>lcp</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>Qry\textsubscript{1}</td>
<td>AAACGG\textsubscript{G}CATT\textsubscript{AAA}</td>
</tr>
<tr>
<td>\text{min}</td>
<td>7</td>
<td>Qry\textsubscript{2}</td>
<td>AAACGG\textsubscript{G}GATT\textsubscript{CAA}</td>
</tr>
<tr>
<td>\text{min}</td>
<td>11</td>
<td>Qry\textsubscript{3}</td>
<td>AAACGG\textsubscript{G}GATT\textsubscript{GCA}</td>
</tr>
<tr>
<td>\text{min}</td>
<td>5</td>
<td>Qry\textsubscript{4}</td>
<td>AAACCT\textsubscript{G}\textsubscript{CATTGAA}</td>
</tr>
<tr>
<td>\text{min}</td>
<td>8</td>
<td>Qry\textsubscript{5}</td>
<td>AAACCT\textsubscript{G}\textsubscript{C}TTTTGA</td>
</tr>
</tbody>
</table>

Figure 2.5: Calculation of the longest common prefix (lcp) between \text{topDB} and \text{Qry} suffixes from textit\text{Qry}_1 to \text{Qry}_5. The lcp between \text{Qry}_1 and \text{topDB} is obtained from the lcp array and that is 10. The lcp between \text{Qry}_2 and \text{topDB} is the minimum value between \text{lcp(Qry}_1, \text{Qry}_2) and \text{match(Qry}_1 and \text{topDB}), which is 7. Similarly, the lcp between \text{Qry}_3 and \text{topDB}, \text{Qry}_4 and \text{topDB}, and \text{Qry}_5 and \text{topDB} are 7, 5 and 5 accordingly.

Suppose, there is a query group from \text{SA}[i] to \text{SA}[j] where \(i < j\). The \text{topDB} is at \text{SA}[i-1] and \text{bottomDB} is at \text{SA}[j+1]. We need to calculate the lcp between \text{SA}[i-1] (\text{topDB}) and query suffixes from \text{SA}[i] to \text{SA}[j]. From lcp[i], we get the lcp between \text{SA}[i-1] and \text{SA}[i], say lcp[i] = x. lcp[i+1] holds the lcp between \text{SA}[i+1] and \text{SA}[i] (say lcp[i+1] = y). The lcp between \text{SA}[i-1] and \text{SA}[i+1] will be the minimum between x and y, because the suffix at \text{SA}[i-1] matches up to x length with the suffix at \text{SA}[i] and the suffix at \text{SA}[i] matches up to y length with the suffix at \text{SA}[i+1]. As all the suffixes are lexicographically sorted in SA, when x < y, the suffix at \text{SA}[i+1] has to match up to x length with the suffix at \text{SA}[i-1] and vice versa when y > x. Now, say lcp between \text{topDB} and \text{SA}[i+1] is z. To find the lcp between \text{topDB} and \text{SA}[i+2], the minimum between z and lcp[i+2] will be calculated based on the above fact. So by calculating the lcp between \text{SA}[i] and \text{topDB}, we can calculate the lcp between \text{SA}[i+1] and \text{topDB} and so on. The same procedure is followed for calculating the lcp between \text{bottomDB} and query suffixes from \text{SA}[i] to \text{SA}[j].
Figure 2.6: To compare the query suffixes with one database suffix. The numbers on the left show the number of comparisons required for finding the exact match length between each query suffix and the database suffix. The underlined portion of the query suffixes represents the exact match with the database suffix. The left comparisons show the linear approach to find the exact match, which requires 32 comparisons, whereas the right comparisons use dynamic programming approach that eliminates 22 matches.

[Figure 2.5].

The above procedure is linear in terms of time complexity. But it requires more time to generate the lcp array while constructing the suffix array. We have tested that producing lcp array adds approximately 50% to the time required to construct the SA (detail in section 2.3.3), and therefore the additional overhead does not benefit the algorithm. Also, the lcp array generates the lcp information for consecutive DB suffixes which is unused in our algorithm. When the query dataset is significantly smaller than database, the above algorithm requires more time to produce some unnecessary information, because a small number of query sequences might end up with a small number of query groups (although this hypothesis is not tested). Therefore, when the query dataset << database, the following algorithm is designed to find the lcp information without generating the lcp array. In this case, the suffix array is constructed with database and query sequences without the lcp information.
Hypothesis II (query dataset << database): To check the length of the longest common prefix, a linear comparison between two DB suffixes (SA[i-1] and SA[j+1]) and each query suffixes in the Qry group (SA[i] to SA[j]) is needed, where \( i \leq j \).

We can make this comparison more efficient by applying dynamic programming [Figure 2.6]. As SA[i-1] to SA[j+1] is in lexicographically sorted order, SA[i-1] has the shortest length match with SA[j]. Say that the match length is \( a \). That means, all suffixes from SA[i] to SA[j] match at least of length \( a \) with SA[i-1]. So, we first compare SA[i-1] with SA[j] and they match up to length \( a \). Then compare SA[i-1] with SA[j-1] from length \( a+1 \) and say they match up to \( a+b \) length. To compare SA[i-1] with SA[j-2], we need to start the comparison from length \( a+b+1 \) and so on. Here, the total number of comparisons is actually equal to the match length between SA[i] and SA[i-1]. To calculate the exact match length with DB suffix SA[j+1], the same procedure is applied.

Time complexity: Suppose there are \( n \) reference sequences where each of them is of length \( l \) and there are \( m \) query sequences where each of them is of length \( r \). To make a suffix array \( S \) of \( n \) sequences of length \( l \) requires \( O(n.l) \) time. To find whether a string of length \( r \) is a substring of \( S \) requires \( O(r + \log(n)) \) time in a suffix array. There are \( m.r \) suffixes of \( m \) query sequences of length \( r \). To find the longest match for \( m.r \) suffixes requires \( O(m.r^2 + m.r \log(n)) \) in suffix array.

On the other hand, using the approach described above, to make the suffix array for \( n \) reference sequences of length \( l \) and \( m \) query sequences of length \( r \) requires \( O(n.l+m.r) \) time. To find the longest exact match, we need to traverse the suffix array and compare the Qry groups with DB suffixes. To compare one Qry group, it requires \( O(r) \) time because there cannot be a match of greater than \( r \) length since the maximum length of the query sequence is \( r \). The time complexity of this part is \( O(P.r) \) where \( P \) is the number of Qry groups in the suffix array. In the worst case, if each Qry group has only one query suffix then \( P = m.r \) and the complexity will be \( O(m.r^2) \) and it is the same complexity as the conventional
approach. But this scenario is highly unlikely. In the average case, our approach should run much faster than the conventional approach that needs to check each suffix separately and thus requires more time. In our approach we use dynamic programming to reuse the match length information of one suffix to calculate the match length information of the other suffixes, reducing the overall run time [Figure 2.6]. To compare the time complexity calculation with the conventional approach, we can write:

\[
O(P.r) = O\left(\frac{\text{number of suffixes}}{\text{avg. group size}}\cdot r\right) = O\left(\frac{m.r}{\text{avg. group size}}\cdot r\right) = O\left(\frac{m.r^2}{\text{avg. group size}}\right)
\]

According to the above equation, the time complexity of the conventional approach is divided by \text{avg. group size} in our implementation. As there are only four letters in DNA sequences, it is very unlikely that \text{avg. group size} will be one. Therefore, this step speeds up our algorithm in practical cases.

### 2.2.3 Generating the Alignment

For each query sequence, the candidate database sequence(s) is identified in a previous step. In this step, the Smith-Waterman-Gotoh algorithm is applied to generate the optimal alignment between each query sequence and their candidate database sequence using the user defined scoring matrix and gap penalties. In general, the Smith-Waterman-Gotoh algorithm produces the optimal alignment with affine gap penalties and the time complexity is quadratic. Suppose there are \(m\) query sequences of length \(r\) and \(n\) database sequences of length \(l\). In our case, this algorithm will not be costly, because the time complexity will be \(O(mrl)\) instead of \(O(mrnl)\) where \(r\) and \(l\) will typically be \(< 1000\) bp and \(m, n\) will be in excess of \(10^4\) and often exceeds \(10^6\).
2.3 Result Analysis

Qudaich generates alignments based on Smith-Waterman-Gotoh algorithm, which produces the optimal alignment. Therefore, the accuracy of the alignment test is unnecessary, and thus the main concern is to prove that how accurately the candidate database sequences are selected.

2.3.1 Testing Datasets

Three different test sets were used to test the accuracy and the speed of the aligner.

Test set 1: Two mosquito metagenomes were used as the first test set. The first mosquito metagenome with 32,965,046 base pairs was used as the database and the other mosquito metagenome with 54,339,014 base pairs was used as the query dataset. The database contained 318,477 sequences with an average sequence length of 103 basepairs. The average sequence length of query dataset was also similar, 101 basepairs and the dataset contained 533,720 sequences.

Test set 2: Two complete genomes were considered as the second test set. *Escherichia coli* O157:H7 str. EDL933 (5,528,445 bp) was used as the database and *Salmonella enterica* subsp. *enterica* serovar Typhimurium str. LT2 (4,857,432 bp) was used as the query. To create shotgun and amplicon sequence libraries based on these two genomes, Grinder is used [54]. Using Grinder, two pairs of datasets were created. For both genomes, in one set, there were 2.5 million sequences of 200 base pairs (total 1 billion bp) and in the second set, there were about 1 million sequences of 500 bp (total 1 billion bp).

Test set 3: The third test set was designed to test protein sequence alignments. Genemarker [55] was used to identify open reading frames from the sequences in test set 1, and these were converted to amino acid sequences. The query mosquito metagenome contains 343,611 sequences of length 31 aa (on average) and the database contains 177,418
sequences of average length 32 aa.

2.3.2 Accuracy for Finding the Candidate Database Sequences

Test dataset 1 was used to test the accuracy of finding the candidate database sequences, and BLAST output was used to check the accuracy, because of its wide acceptance. Figure 2.7 shows the prediction of candidate database sequences based on the BLAST results. When there is a good alignment (high % identity and high % coverage) between the query and the database sequences, there is a strong correlation between our predictions and those of BLAST. Candidate database sequences prediction matches 98.0% and 99.3% for hypothesis I and hypothesis II accordingly when % coverage and % identity is greater than 80% [Table 2.1]. This percentage increases when we consider the top two database sequences per query sequence instead of just one while calculating the candidate database sequence. The candidate database sequences that do not have a match with blast outputs basically have poor alignment scores with low % identity and low % coverage [Figure 2.7 and Figure 2.8]. Therefore, testing the two different hypotheses for selecting the database sequences showed that both approaches are effective at identifying the candidate database sequences. This suggested that the assertions underlying the two hypotheses are validated by the data. However, when there is no strong match between the sequences, hypothesis II gives better prediction than hypothesis I.
Figure 2.7: The prediction of candidate database sequences. The left panel shows the prediction based on hypothesis I, and the right panel shows the prediction based on hypothesis II. The x-axis represents the % identity and y-axis shows the % coverage of the alignment of each query sequence based on BLAST output. Different color shows the prediction of the candidate database sequence for each query sequence. Red and orange indicate those query sequences where the candidate database sequence based qudaich matches with the database sequence selection based on BLAST approach. The red color denotes those alignments that have the best score based on BLAST output. Blue indicates those queries where the database sequence predicted by BLAST does not match with the candidate database sequence predicted by qudaich.
Figure 2.8: BLAST score based on the prediction of candidate database sequences. The color indicates the same group of candidate database sequences described in figure 2.7
To investigate the reason why hypothesis I gives poor performance for low % identity and low % coverage, some alignments were manually checked. We will explain this discrepancy using a hypothetical example. Suppose that for a given query sequence $q$, database sequence $d_1$ was chosen with hypothesis I while database sequence $d_2$ was chosen with hypothesis II, and let us assume that these are the only two sequences with homology to $q$ in the database.

When we examine the alignment between $q$ and $d_2$, we find a couple of longer exact matches. In contrast, when we examine the alignment between $q$ and $d_1$ we find many shorter length matches. Hypothesis I uses the frequency that the sequence occurs to identify the best match, so $d_1$ will dominate because of many short matches. In contrast, hypothesis II uses the total length of the longest common prefix to identify the candidate database sequence; therefore, sequence $d_2$ will dominate [Figure 2.9].

In the alignment result, only the good alignments (having high % coverage and high % identity) are useful and both hypotheses actually find the similar candidate database sequence in this case. Therefore, qudaich can successfully predict the candidate database sequences.

<table>
<thead>
<tr>
<th>Approach</th>
<th>All alignments</th>
<th>Coverage &gt; 60% Identity &gt; 60%</th>
<th>Coverage &gt; 80% Identity &gt; 80%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis I</td>
<td>71.6%</td>
<td>96.7%</td>
<td>97.98%</td>
</tr>
<tr>
<td>Hypothesis I (top 2 db)</td>
<td>84.4%</td>
<td>98.5%</td>
<td>99.1%</td>
</tr>
<tr>
<td>Hypothesis II</td>
<td>87.9%</td>
<td>98.8%</td>
<td>99.3%</td>
</tr>
<tr>
<td>Hypothesis I (top 2 db)</td>
<td>94.1%</td>
<td>99.5%</td>
<td>99.7%</td>
</tr>
</tbody>
</table>

Table 2.1: Percentage of predicted candidate database sequence matches with BLAST results
2.3.3 Real Time Statistics

A computer with an Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz was used to compare and measure the execution time of qudaich and other contemporary aligners. A single core/thread was used for all tests so that the results are comparable.

In qudaich, two different algorithms are designed for each of the two hypotheses. Using test dataset 1, the time requirements for both algorithms were measured. The algorithm based on hypothesis II required more time to construct suffix array and to find candidate database sequences because of the longest common prefix (lcp) calculation. According to table 2.2, Smith-Waterman-Gotoh (SWG) algorithm to generate the alignments requires 71.4% and 64.2% of total execution time for hypothesis I and hypothesis II accordingly. Hence, using a SIMD implementation of SWG algorithm will probably reduce the total execution time of qudaich significantly.

Several methods were developed during the last couple of decades to allow fast sequence alignments. Six popular aligners from these were compared using dataset 1. As shown in table 2.3, qudaich generated alignments faster than Usearch and BWA-SW. But MUMmer, bowtie and BWA-MEM (http://bio-bwa.sourceforge.net/bwa.shtml) were faster than qudaich. However, qudaich produced alignments for all the query sequences i.e. for
Table 2.2: Time comparison between hypothesis I and hypothesis II for DNA sequence alignment

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Construct Suffix Array</th>
<th>Find Candidate DB</th>
<th>Generate Alignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis I</td>
<td>15.3 sec</td>
<td>11.1 sec</td>
<td>1min 8.6 sec</td>
</tr>
<tr>
<td>Hypothesis II</td>
<td>24.9 sec</td>
<td>13.5 sec</td>
<td>1min 8.9 sec</td>
</tr>
</tbody>
</table>

Table 2.2: Time comparison between hypothesis I and hypothesis II for DNA sequence alignment

533,720 sequences. On the other hand, bowtie generated alignments only for 614 query sequences, MUMmer produced alignments for 16,219 query sequences and BWA-MEM aligned 89,073 query sequences. As bowtie only generated very few alignments, it seems that bowtie is not a good option to align two NGS datasets and this approach was not used in further tests. The alignments generated by MUMmer and BWA-MEM were investigated further.

Table 2.3: Real time statistics of BLAST, MUMmer, Usearch, BWA, Bowtie and qudaich (hypothesis I)

<table>
<thead>
<tr>
<th>Aligner</th>
<th>Time</th>
<th># Alignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAST+</td>
<td>126 min 27 sec</td>
<td>8,900,641</td>
</tr>
<tr>
<td>USEARCH</td>
<td>11 min 33 sec</td>
<td>96,530</td>
</tr>
<tr>
<td>BWA-SW</td>
<td>2 min 5 sec</td>
<td>57,734</td>
</tr>
<tr>
<td>Bowtie</td>
<td>1min 17 sec</td>
<td>614</td>
</tr>
<tr>
<td>MUMmer</td>
<td>1 min 9 sec</td>
<td>16,219</td>
</tr>
<tr>
<td>BWA-MEM</td>
<td>53 sec</td>
<td>89,073</td>
</tr>
<tr>
<td>Qudaich</td>
<td>1 min 36 sec</td>
<td>533,720</td>
</tr>
</tbody>
</table>

Table 2.3: Real time statistics of BLAST, MUMmer, Usearch, BWA, Bowtie and qudaich (hypothesis I)
Figure 2.10: Comparison of the prediction of candidate database sequences for MUMmer and BWA-MEM. The top panel shows the prediction based on MUMmer, and the bottom panel shows the prediction based on BWA-MEM. The x-axis represents the % identity and y-axis shows the % coverage of the alignment of each query sequence based on BLAST output. Different colors show the prediction of the candidate database sequence for each query sequence. Red and orange indicates whether the candidate database sequence matches with the database sequence selection based on BLAST approach. The red color specifically denotes those alignments that have the best score based on BLAST output. Blue indicates when the database sequence predicted by BLAST does not match with the candidate database sequence predicted by MUMmer or BWA-MEM. Green indicates those cases where BWA-MEM and MUMmer do not report an alignment for a query sequence.
Table 2.4: Detailed comparison between MUMmer, BWA-MEM and qudaich

<table>
<thead>
<tr>
<th>Aligner</th>
<th>Time (sec)</th>
<th>No. of Alignments</th>
<th>Missing Alignments Coverage &gt; 90% Identity &gt; 90%</th>
<th>Missing Alignments Coverage &gt; 95% Identity &gt; 95%</th>
<th>#Candidate Database match with BLAST(^x)</th>
<th>#Candidate Database match with BLAST(^y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUMmer</td>
<td>69</td>
<td>16,219</td>
<td>3,209</td>
<td>633</td>
<td>12,612</td>
<td>3,392</td>
</tr>
<tr>
<td>BWA-MEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(seed len 19)</td>
<td>53</td>
<td>89,073</td>
<td>262</td>
<td>50</td>
<td>78,769</td>
<td>3,130</td>
</tr>
<tr>
<td>BWA-MEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(seed len 13)</td>
<td>215</td>
<td>98,387</td>
<td>178</td>
<td>49</td>
<td>88,319</td>
<td>1,677</td>
</tr>
<tr>
<td>Quaich</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(hypothesis I(^a))</td>
<td>42.5</td>
<td>107,738</td>
<td>83</td>
<td>0</td>
<td>72,474</td>
<td>29,703</td>
</tr>
<tr>
<td>Quaich</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(hypothesis I(^b))</td>
<td>46.4</td>
<td>138,478</td>
<td>43</td>
<td>0</td>
<td>91,188</td>
<td>34,957</td>
</tr>
<tr>
<td>Quaich</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(hypothesis II(^c))</td>
<td>52.4</td>
<td>99,661</td>
<td>109</td>
<td>0</td>
<td>71,271</td>
<td>23,826</td>
</tr>
<tr>
<td>Quaich</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(hypothesis II(^d))</td>
<td>56.7</td>
<td>131,173</td>
<td>44</td>
<td>0</td>
<td>94,865</td>
<td>28,198</td>
</tr>
</tbody>
</table>

\(^x\) indicates those alignments that have the best score based on blast results
\(^y\) indicates those alignments that is reported on blast results

\(^a\) indicates that only those alignments were generated that have frequency \(>=9\)
\(^b\) indicates that only those alignments were generated that have frequency \(>=7\)
\(^c\) indicates that only those alignments were generated that have summation of lcp \(>= 150\)
\(^d\) indicates that only those alignments were generated that have summation of lcp \(>= 100\)
Both MUMmer and BWA-MEM output were compared with the BLAST output [Figure 2.10]. MUMmer produced only those alignments that have very high % identity and % coverage. BWA-MEM produced more alignments than MUMmer, but it also did not do any poor alignments. If qudaich was used to generate only the significant alignments like BWA-MEM, qudaich required less time than MUMmer and BWA-MEM. Table 2.4 shows the detailed comparisons between them.

We scored the alignments that have greater than 90% coverage and greater than 90% identity as similar while those that have both coverage and identity greater than 95% as very similar.

MUMmer required 69 seconds to produce 16,219 alignments. It missed about 3,209 similar alignments where 633 were very similar alignments. BWA-MEM required 53 seconds to generate 89,073 alignments where it missed 262 similar alignments and 50 of them were very similar alignments. With reduced seed length (=13), BWA-MEM required 215 seconds and generated about 9000 more alignments but very few of them were similar alignments, and it still missed 49 very similar alignments. On the other hand, qudaich took only 42.5 seconds to produce 107,738 alignments where it reported all very similar alignments and only missed 83 similar alignments. Depending on different parameters, qudaich always generated all very similar alignments with some variation of execution time.

Table 2.4 also reports the quality of the predicted candidate database sequences. Using parameter $b$ in hypothesis I and using parameter $d$ in hypothesis II, qudaich generated more alignments for 12,419 and 16,096 query sequences accordingly than BWA-MEM (seed length 19) that were considered as the best alignments for those query sequences according to the BLAST bit score. Using the same parameter, qudaich also produced more alignments for 31,827 and 25,068 query sequences accordingly than BWA-MEM that were reported in BLAST results. If we decrease the seed length of BWA-MEM to increase the sensitivity, it required almost 4 times more execution time than that of qudaich but still produced less
number of useful alignments (having a match with BLAST results) than qudaich. Qudiach reported more alignments that match with BLAST results using parameter $b$ and $d$ than using parameter $a$ and $c$; however, parameter $a$ and $c$ required less processing time than parameter $b$ and $d$. This suggests that qudaich is flexible enough to generate results based on users’ demand.

As test dataset 1 is a comparatively smaller dataset and has shorter read length, dataset 2 was used to compare the real time statistics between qudaich and BWA-MEM. As shown in table 2.5, qudaich is about 1.8x faster than BWA-MEM using both hypotheses. Also, to find the candidate database sequences from these large datasets, qudaich required less than 30% of total execution time using both hypotheses. This indicates that, in less processing time, qudaich can effectively sort out and filter the database by finding only those database sequences that can guarantee a good alignment (with the corresponding query dataset).

<table>
<thead>
<tr>
<th>Aligner</th>
<th>2,500,000 x 200 bp</th>
<th>1,000,000 x 500 bp</th>
</tr>
</thead>
<tbody>
<tr>
<td>BWA-MEM (seed len 19)</td>
<td>53 min 42 sec</td>
<td>100 min 8 sec</td>
</tr>
<tr>
<td># alignments: 918,386</td>
<td></td>
<td># alignments: 585,192</td>
</tr>
<tr>
<td>Qudaich (hypothesis i)</td>
<td>24 min 59 sec</td>
<td>54 min 27 sec</td>
</tr>
<tr>
<td>(5 min 24 sec + 19 min 35 sec)</td>
<td></td>
<td>(6 min 40 sec + 47 min 47 sec)</td>
</tr>
<tr>
<td># alignments: 2,500,000</td>
<td></td>
<td># alignments: 1,000,000</td>
</tr>
<tr>
<td>Qudaich (hypothesis ii)</td>
<td>27 min 52 sec</td>
<td>56 min 44 sec</td>
</tr>
<tr>
<td>(8 min 17 sec + 19 min 35 sec)</td>
<td></td>
<td>(8 min 57 sec + 47 min 47 sec)</td>
</tr>
<tr>
<td># alignments: 2,500,000</td>
<td></td>
<td>#alignments: 1,000,000</td>
</tr>
</tbody>
</table>

Table 2.5: Real time statistics for DNA sequence alignment using test set 2
Protein sequence alignment:
Qudaich can also align protein sequences. For protein sequence alignments, promer (integrated in mummer) and usearch are currently the most efficient aligner in terms of execution time compared to other protein sequence aligners.

To compare protein vs. protein sequence alignments, test set 3 was used. Table 2.6 shows that qudaich was faster than Usearch. From the query sequences presented in BLAST results, 87.3% cases, the candidate database sequences predicted by qudaich matched with the database sequences reported in BLAST results and 76.7% cases, BLAST considered the alignment with the candidate database sequences as the best alignment according to bit score. For Usearch, 83.9% cases the database sequences matched with BLAST outputs where 80.2% case BLAST considered the alignments as best alignments. The quality of the prediction of candidate database sequences using qudaich and Usearch is shown in figure 2.11.

<table>
<thead>
<tr>
<th>Aligner</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLASTP</td>
<td>37 min 9 sec</td>
</tr>
<tr>
<td>Usearch</td>
<td>11 sec</td>
</tr>
<tr>
<td>Qudaich (hypothesis l)</td>
<td>6.1 sec</td>
</tr>
</tbody>
</table>

Table 2.6: Real time statistics for protein sequence alignment using test dataset 3
Figure 2.11: Comparison of candidate database sequence prediction for protein sequence alignments using qudaich (hypothesis I) and Usearch. The top panel shows the prediction based on qudaich, and the bottom panel shows the prediction based on Usearch. The x-axis represents the % identity and y-axis shows the % coverage of the alignment of each query sequence based on BLASTP output. Different colors show the prediction of the candidate database sequence for each query sequence. Red and orange indicates whether the candidate database sequence matches with the database sequence selection based on BLAST approach. The red color specifically denotes those alignments that have the best score based on BLAST output. Blue indicates when the database sequence predicted by BLAST does not match with the candidate database sequence predicted by qudaich or Usearch. Green indicates those cases where qudaich and Usearch do not report an alignment for a query sequence.
In this work, real time statistics for translated nucleotide alignment was also done. Very few aligners support translated nucleotide alignments and promer is one of them. Qudaich is 3 times faster than promer for translated nucleotide alignment [Table 2.7].

<table>
<thead>
<tr>
<th>Aligner</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>tBLASTx</td>
<td>757 min 43.8 sec</td>
</tr>
<tr>
<td>Promer</td>
<td>4 min 19 sec</td>
</tr>
<tr>
<td>Qudaich (hypothesis I)</td>
<td>1 min 25.8 sec</td>
</tr>
<tr>
<td>(Generate alignments for all query reads)</td>
<td>(52.8 sec + 33 sec)</td>
</tr>
</tbody>
</table>

Table 2.7: Real time statistics for translated nucleotide alignments using test dataset 1

The above results show that qudaich produces more useful results for both DNA and protein sequence alignments but with less time than the contemporary aligners take. This performance gain results from the novel algorithmic structure of qudaich that effectively reduces the search space for each query sequence to find the candidate database sequence. So, qudaich is particularly suited to NGS datasets with huge numbers of database sequences.

### 2.4 Conclusion

In this chapter, we present a novel approach for local sequence alignment, which tries to find the best possible alignment for each query sequences. In this approach, both database and query sequences are organized together. Based on this organization, the candidate database sequences are identified so that the optimal alignment can be generated using Smith-Waterman-Gotoh algorithm. In most of the cases, the alignments with the candidate database sequences are the best possible alignment against the whole database. Here, we
propose two heuristics for efficient searching of the candidate database sequences. Our algorithmic structure allows to plug in different heuristics.

We showed that qudaich is more efficient in terms of execution time and accuracy than contemporary aligners. An interesting characteristic of qudaich is that, except the suffix array construction, the time complexity mainly depends on the size of the query datasets. As the suffix array construction is linear, qudaich's execution time should not be costly for large databases. This characteristic has a key advantage over the conventional approaches that find and extend or join seeds to construct alignments. In those cases, the execution time will be highly dependant on database size. Aside from constructing the suffix array part, the other steps of the algorithm can be easily parallelized.

Finally, we hope qudaich will help the researchers to produce their results in less time, power, and cost.
Chapter 3

PhiSpy: A Novel Algorithm for Finding Prophages in Bacterial Genomes

3.1 Prophages

Phages, viruses that infect bacteria, have two lifestyles: lytic and lysogenic. During lysogenic growth, phages infect their host and then remain inside the microbial cell replicating with the genome. In this state they are called prophages. These prophages will be part of the bacterial DNA in future cell divisions until appropriate environmental conditions cause them to release from their host and enter into a virulent lifestyle. The advantages of a lysogenic lifestyle for phages are numerous, including increased fecundity and increased survival within the protective bacterial environment. Integrated prophages can constitute up to 20% of a bacterial genome [56–58] and play a key role in the bacterial life cycle. Prophage integration can regulate bacterial populations, make inactive or alter the expression of some bacterial genes, and can convert non-pathogenic bacteria into pathogens and some virulent into hyper-virulent strains [59–61].

A prophage normally integrates into a genome by site-specific recombination, which is catalyzed by a family of proteins called integrases [62]. These proteins recognize se-
quences on both the phage (attP, attachment site in the phage genome) and bacterial (attB, attachment site in the bacterial genome) genomes, and homologous recombination between these sites results in duplication of a short stretch of DNA in the continuity of the chromosome, resulting in the duplicated sites, attL and attR, flanking the inserted prophage and ready for the reverse reaction, excision of the phage from the chromosome. The att regions vary widely in total length and in the extent of the resulting duplication, which depends on the phage and its specific integration site within a bacterial genome [56, 63–66]. Phages often integrate into tRNA/tmRNA genes but do not exclusively use those loci as the target site for integration [67].

### 3.2 Prophage Finding Applications

Identification of prophages in bacterial genomes is a difficult process. Current methodology of automated prophage identification usually relies on protein similarity searches to identify clusters of protein encoding genes that have some similarity to known or predicted phage genes. Based on this approach, PhageFinder [67] was one of the first automated applications for detecting prophages. PhageFinder screens the bacterial genome with a fixed window size of 10 Kb and searches (using hidden Markov models and BLAST [11]) for windows with at least four hits against a collection of bacteriophage proteins. These windows are then extended gene-by-gene if the annotated gene belongs to tRNAs, integrase gene, etc [67]. ACLAME ProPhinder is another successful phage-finding algorithm that combines protein similarity and statistical methods [68, 69]. ProPhinder starts by determining phage-like coding sequences in an input bacterial genome by BLASTP similarity analysis against the ACLAME phage protein database. Then it evaluates each phage-like genomic segment for the presence of potential prophages using statistical methods. Because these applications use homology-based approaches, they are limited to finding
known prophages and it is difficult to locate those prophages that are not similar to known phages. An alternative approach for detecting prophages (DRAD) that depends on the dinucleotide relative abundance instead of sequence similarity was able to locate some of those prophages found by Prophinder and PhageFinder as well as some novel prophages [70]. No single tool is able to find all prophages in all bacterial genomes [70]. This suggests that combining multiple methods or different characteristics of prophages may identify a larger set of prophages.

In this study, a bioinformatics tool (PhiSpy) was developed for identifying prophages, which focuses on the characteristics of prophages that exhibit no similarity to sequenced genomes. In particular, five distinctive similarity-agnostic characteristics were identified and their relative capabilities to define prophages were tested in the absence of homology to known phage proteins. These characteristics are protein length, transcription strand directionality, customized AT and GC skew, and the abundance of unique phage DNA sequence words. Optimized metrics were designed to quantify each of these characteristics and the random forest classification algorithm was used to predict prophages by ranking genomic regions based on those characteristics. In addition to each of these metrics, phiSpy also uses similarity-based approaches, thus enabling a complete identification of prophages in a genome. Finally, each predicted prophage region was evaluated by the identification of duplicate att sites and by phage protein similarity. PhiSpy found 94% of prophages in 50 bacterial genomes with a 6% false negative rate and a 0.66% false positive rate.
3.3 Materials and Methods

3.3.1 Data Collection

All bacterial genomes used in this analysis were retrieved from the Phage Annotation Tools and Methods server (Phantome server: http://www.phantome.org). As of March 2010, the server contained 547 complete bacterial genomes (at most 20 contigs) of which only 41 bacterial genomes (Supplemental Table 1) had 190 manually annotated prophages. All other lytic and lysogenic phage genomes were also collected from the Phantome server.

3.3.2 Data Analysis

PhiSpy publicly available at http://phispy.sourceforge.net/ was written in python and C++. It has four steps (Supplemental Figure 1 is a flow chart of each step). Each step is described below.

Calculation of different characteristics

The first step calculates different parameters for the whole genome. The calculation of these parameters depends on a group of genes rather than a single gene. Therefore, for a complete genome, these parameters were computed using a sliding window of n genes. The average number of genes of the 190 known prophages is 39; so a window size of 40 genes was considered. The parameters are as follows:

Customized AT and GC skew: The customized AT/GC skew was calculated by modifying the cumulative skew calculation [71, 72]. For a group of consecutive genes, the average skew of A, C, G and T were measured using the following formula:
Skew of $A = \frac{1}{n} \sum_{i} \frac{A_i}{A_i + T_i}$

Skew of $T = \frac{1}{n} \sum_{i} \frac{T_i}{A_i + T_i}$

Skew of $G = \frac{1}{n} \sum_{i} \frac{G_i}{C_i + G_i}$

Skew of $C = \frac{1}{n} \sum_{i} \frac{C_i}{C_i + G_i}$

where $n$ is the number of genes, $A_i$ is the number of A nucleotide in the $i$th gene and so on. The customized AT and GC skews (described under Results) were developed and were calculated as follows:

Customized AT skew $= \frac{1}{n} \sum_{i} \frac{A_i}{A_i + T_i} - \frac{1}{n} \sum_{i} \frac{T_i}{A_i + T_i} = \frac{1}{n} \sum_{i} \frac{A_i - T_i}{A_i + T_i}$

Customized GC skew $= \frac{1}{n} \sum_{i} \frac{G_i}{G_i + C_i} - \frac{1}{n} \sum_{i} \frac{C_i}{G_i + C_i} = \frac{1}{n} \sum_{i} \frac{G_i - C_i}{G_i + C_i}$

This customized version combines AT/GC and compensates for local deviations in the composition due to, for example, strand bias.

---

**Difference in median protein length:** The median (M) of the lengths of all the proteins in a bacterial genome was calculated. For a group of proteins in a given window, the median protein length (m) was calculated and the difference in median length was computed as (M-m).
Transcription strand orientation: For a given window size, the genes were partitioned in such a way so that all consecutive genes in a particular partition pointed in same direction. The sum of the number of genes in the two largest partitions was taken for the window to maximize the number of consecutive genes in the same direction.

Abundance of phage words: A word is defined as a set of 12 consecutive base pairs. Each gene was split into twelve base pair long non-overlapping words (four consecutive amino acids each). A unique phage word library was built based on the 41 bacterial genomes that have well annotated prophages. The library was constructed as follows:

Bacterial words, \( B = \{ \text{the words of all bacterial genes of those 41 genomes but not including genes in prophages} \} \)

Phage words, \( P = \{ \text{the words of all prophage genes of those 41 genomes} \} \)

Unique phage word library = \( P - B \)

To measure the abundance of phage words, Shannon’s index [73, 74] and the frequency of the presence of phage words were calculated.

Shannon’s index was calculated by the following formula:

\[
H = - \sum_i p_i \log(p_i)
\]

where \( p_i \) is the frequency of those words which are present in the phage word library. The frequency of words (F) of a window was calculated by dividing the number of available phage words with the total number of words. For a given window, the abundance of phage
words is $F/H$.

**Homology:** In a window of 40 genes, if there are at least ten genes whose functional description is in phage subsystems (i.e., phage functional categories in the Phantome SEED database: http://www.phantome.org/PhageSeed/seedviewer.cgi), the window was considered a prophage window otherwise it was considered a bacterial window.

**Classification Algorithm**

The second step of PhiSpy is to classify a window as a bacterial or a prophage window using random forests [75]. A random forest is a classification algorithm that consists of multiple independent decision trees. The random forest requires a training set with multiple variables to build the forest of decision trees. In this case there were five parameters whose values vary among distantly related genomes. If the similarities between two genomes were evolutionary significant, then they were considered as closely related genomes; otherwise they were considered as distantly related genomes (the SEED API was used to determine relatedness [76]). Therefore, for every group of closely related genomes, a different training set was constructed.

**Training/test set:** In the Phantome server, there were 547 complete bacterial genomes that had 20 contigs or fewer (as of March 2010). From these 547 bacterial genomes, 19 groups of closely related genomes were constructed, where each group has at least one genome from the set of 41 bacterial genomes with annotated prophages. These 19 groups included 114 out of 547 bacterial genomes. For each group, one genome with manually annotated prophages was used as the training set for the rest of the genomes of that group [Table 3.1, Supplemental Table 2]. The genomes that did not belong to any group and had no manually annotated prophages were tested using a universal generic training set (constructed in the same way described above but using all 41 bacterial genomes). The
parameter abundance of phage word was ignored in the universal generic training set [Table 3.2].

The statistical software program, R (http://www.r-project.org), was used to implement the random forest [77]. The random forest produces a rank for each window of the whole genome that suggests whether the window consists of bacterial or phage genes.

**Processing the final rank for each gene**

The third step of PhiSpy provides a prediction status - either 0 (for non prophage genes) or 1 (for prophage genes) for each gene in the genome. If the window size was \( n \), each gene contributed to 1 to \( n \) windows. So the final rank of a particular gene was measured by taking the average rank of the window in which the gene participated. The prophage prediction status was calculated from the final rank. If the final rank was greater than half of the maximum rank of any gene in the genome, then the gene was considered as a phage gene; otherwise it was considered as a bacterial gene.

**Evaluation of the prediction**

The final step is to define the \( att \) sites for the predicted prophages and the overall evaluation of the prophages. When phages integrate into their hosts genome they are usually bounded by two \( att \) sites - a short repeated sequence that flanks the insertion site. To find this insertion site, for each predicted prophage region (considered an initial prediction), the following steps were followed:

i. Extending the predicted region up to 2,000 base pairs on both sides.

ii. Identifying all duplicate short DNA sequences in that region.

iii. Finding the repeated pair that has minimum distance (less than 1000 bp) from either
integrase/recombinase or tRNA/tmRNA genes or both. If there are multiple repeated pairs, the pair that covers the largest region was considered as the potential att sites. If no integrase/recombinase or tRNA/tmRNA genes were found, then the initially predicted region was considered.

After identifying the att sites, the next step is verifying the att sites. If the att sites lie inside the initial prediction, the number of phage-like proteins was counted for the two gaps (between attL and the start of the initial prediction and between attR and the end of the initial prediction). If the function of one-quarter of the genes in those two gaps belongs to phage subsystems, the initial prediction was considered as the final prediction otherwise the region covered by att sites was considered as the final prediction. If the att sites were outside the initial prediction, the same procedure was followed.

After verifying the att sites, the predicted prophages were evaluated by checking the function of all proteins in that region. If there are more than five proteins whose functions belong to the phage subsystems or are unknown and the number of phage-like/unknown proteins is at least half of the total number of proteins in the predicted region, then the region was considered as a potential prophage. However, if a group of proteins, whose functions belong to the phage subsystems, was not considered in the classification step as a probable prophage, then this region was also considered as a potential prophage.

**Calculation of false positives and false negatives**

The manually curated phage subsystems were used to evaluate the accuracy of the approach. A two-step program was designed to automatically calculate the error rate of the prophage prediction (for those genomes which have no information about prophages in their original genome analysis paper). In the first step of the program, true positives (TP) and false positives (FP) were predicted. If the predicted region consists of at least six phage
proteins or 50% of the proteins within the predicted region belong to phage subsystems or are unknown, the predicted region was considered a true positive prophage otherwise the region was considered a false positive and not a prophage (those limits were determined by empirically). Prophages considered as true positives were divided into two groups: (i) known prophages if the region contains phage-like proteins; we considered that it would be identified by similarity based approaches and therefore denoted it as a known and (ii) undefined prophages if the region has no phage-like protein; thus this would unlikely to be called a prophage. In the second step of the program, a region was considered as a false negative (FN) if there were at least six consecutive genes, whose functions belonged to the phage subsystems and the region was unidentified as a potential prophage. However, hypothetical proteins were ignored in this case, because the presence of several hypothetical proteins was not sufficient to predict a region as a prophage region.

3.4 Results

3.4.1 Transcriptional Strand Orientation

The orientation of transcriptional units along the genome highly correlates with the direction of replication [78]. Near the replication origin, genes are oriented in such a way that the direction of transcription coincides with the direction of replication [79]. In a bacterial genome, which typically has a single origin of replication within its circular DNA, two replication forks can proceed independently [80]. Phage genes cluster along the genome as they are organized into transcriptional units that are co-regulated [81]. This causes a large cluster of phage genes to be oriented in the same direction, even if it collides with DNA replication [81]. To test this hypothesis, 600 complete phage genomes (both lytic and lysogenic) and 110 complete bacterial genomes were analyzed. For both phage and bacterial genomes, the longest stretch of consecutive genes in the same direction was calculated as a
percentage of the number of genes in the genome (Supplemental Figure 2). Likewise, the number of gene transcription strand changes was calculated as a percentage of the number of genes in the genome. For phage genomes, most of the consecutive genes are encoded on the same strand [Figure 3.1]. In contrast, for bacterial genomes, the longest consecutive cluster of genes (presumably a co-transcribed region) is a small fraction of the genome, and genes frequently change their transcriptional direction. For bacterial genomes, these clusters of genes that have the same transcriptional orientation are most likely operons although this cannot be determined from the sequence alone. Other studies have shown that the average bacterial operon size is three genes [82] but the average phage operon size has not been reported.

Figure 3.1: Orientation of proteins in 110 bacterial genomes (△) and 600 phages (x). Most of the phages have a large group of proteins facing in same direction and fewer proteins change their transcriptional directions. Bacteria, in contrast, cluster fewer proteins in the same direction and have high number of transcriptional direction changes.
3.4.2 Customized AT and GC Skew

Amino acid composition and codon usage: Several articles have discussed how the adaptation of phages towards their hosts plays an important role in viral evolution [83, 84]. By comparing the 190 prophages in 41 bacterial genomes, it was apparent that the overall amino acid usage in prophages and their hosts is very similar [Figure 3.2], although for some amino acids (notably Asp, Glu, Phe, Gly, Lys, Pro, Arg), codon usage differs between prophages and their hosts genomes [Figure 3.3]. For Lys and Phe, the frequency of AAA (in Lys) and TTT (in Phe) is higher in phages than in bacteria, which is probably caused by the different usage of nucleotides A and T in prophages. Similarly there are six codons that encode arginine CGT, CGC, CGA, CGG, AGA and AGG from which CGC is more frequently used and AGA is less frequently used in bacteria. Presumably this skew maintains the balance of G and C nucleotides in Arg codons.

Figure 3.2: Amino acid distribution in the predicted proteins encoded in 41 bacterial genomes (■) and their 190 prophages (□). The amino acid utilization is similar for both but the standard deviation (vertical bars) is higher for prophages than for bacteria.

The GC skew of bacterial chromosomes directly correlates with the direction of replication [85]. Local changes or distortion in the cumulative skew distribution may result from the insertion of foreign DNA into the chromosome [71]. Therefore, customized AT
and GC skew profiles were designed. Unlike the conventional calculations of cumulative DNA skew [72], the customized skew was designed not only for identifying local distortions but also for quantifying the variation of the codon usage in the window of genes.

The customized AT and GC skews (Materials and Methods) were calculated separately for 41 bacterial genomes and their 190 prophages. For all genomes except Xylella fastidiosa, prophages have different AT and GC skews (either positive or negative) than their hosts [Figure 3.4, 3.5]. If there were no bias between the two DNA strands for mutation or selection, the base composition within each strand should be such that A=T and G=C [86, 87]. This implies that the customized AT or GC skew of the whole bacterial genome would be very small. In contrast, it was hypothesized that the customized skew of prophages should be different than that of the whole bacterial genome. To test this hypoth-
esis, two independent samples were constructed. The first sample consisted of the absolute difference between the customized AT/GC skew of prophage genes and the customized AT/GC skew of regions immediately flanking the prophage insertion. The manually curated prophages were used to construct the first sample, and so the sample size was 190. To construct the second sample, 800 different bacterial regions were randomly selected from the 41 bacterial genomes that have manually curated prophages. The absolute differences of the customized AT/GC skew of these regions and the customized AT/GC skew of the flanking genes of these regions was calculated for the second sample. A permutation test was used to analyze whether these two samples are statistically different (Supplemental Materials) [88]. The customized AT skew was significantly different at the 1% level (using both the mean and the median of the sample) between prophage genes and their flanking genes when compared to random chromosomal segments and their flanking genes. The customized GC skew was also significantly different between these samples at the 1% level using the mean value of the sample (but only at 5% level using the median of the sample). Therefore the calculation of the customized AT/GC skew is useful to distinguish prophage
3.4.3 Protein Length

Anecdotal evidence from the study of mycobacterial phages [89] suggests that phages typically have shorter genes than bacterial genes. The reasons are not clear, but phage genomes may enrich for smaller genes because of selective constraints on genome size i.e. for faster replication or more efficient packaging. The mean protein length was calculated for 41 bacterial genomes and their prophages [Figure 3.6] and the result supports the previous study. However, our testing demonstrates that the median length works better than the mean length for discriminating prophage and bacterial genes. As each characteristic was calculated for sliding window (of several genes), using median length calculation, a sharp change occurs at the beginning of a prophage region, but using mean length, the change occurs gradually. The difference between the median of all protein lengths in a genome and the median length of phage proteins in that genome is much higher than the same
calculation for bacterial proteins (Supplemental Figure 4).

Figure 3.6: Average length of bacterial proteins (■) and phage proteins (□) for 41 bacterial genomes and their prophages. Phage proteins are smaller than bacterial proteins. The x axis is not sorted.

3.4.4 Abundance of Phage Words

To find a signature pattern of prophages, the oligo nucleotide composition between prophages and their hosts was analyzed for 190 prophages from 41 bacterial genomes. Each gene was split into twelve base pair non-overlapping words i.e. four non-overlapping codons (from empirical testing, we found that nine bp sequences had too few combinations to be discriminatory and 15 bp sequences were too rare for accurate statistical modeling). For a DNA sequence of length 12, there are 412 different possible combinations. However, only 27% of the words from these combinations are present in our dataset of 41 bacterial genomes with prophages. In total, 25% of the words (4,223,854) are present in bacterial genes, 0.65% of the words (109,533) are present in phage genes and 1.34% of the words (226,228) are common to both phage and bacterial genes. To verify whether these 0.65% words (phage word library) represent the phageness and are uncommon in bacterial genomes, Shannon’s index and the frequency of the presence of these words (Materials and Methods) were calculated for 600 complete phage genomes and 400 complete bacterial genomes. Shannon’s index was used to measure the presence of the different combination of phage words, while
the frequency measurement was used for the presence of phage words. For all bacterial genomes, Shannon’s index (H) is less than 1 and the frequency of phage words (F) is less than 6%. In contrast, for phages, H varies from 0 to 5.5 and F varies from 0 to 45% [Figure 3.7]. The relation between H and F was given by an equation $F = 8.57H + 0.047$ for phages (regression coefficient $R^2 = 0.995$) and $F = 5.85H + 0.014$ for bacterial genomes (regression coefficient $R^2 = 0.993$). The constant term for both equations is negligibly small, and the difference between the two slopes is statistically significant ($p < 0.001$; details in Supplemental Materials). The abundance of phage words was calculated as the slope (F/H) and the value distinguishes phages and bacterial genomes [Figure 3.7]. This indicates that the words from the phage word library are more frequent in phage genomes than bacterial genomes.

### 3.4.5 Importance of Different Characteristics

All the characteristics described above were used to predict prophages in bacterial genomes. The importance of each characteristic varies between different organisms and depends on the training genomes [Table 3.2]. If a training genome and a test genome are closely related, then for most cases, the abundance of phage words is the most important characteristic. The directionality of the transcriptional strand is a strong indicator in all cases, although short phages are missed when this criterion is used alone. Protein length gives better performance in closely related genomes and can work for distantly related genomes when the genome has similar protein length with the training genome. The customized AT and GC skew calculation works better for closely related genomes and gives better performance in bacteria with extreme AT or GC composition, rather those with approximately even distribution of bases. In general, identification of prophages, even without similarity, was strongly assisted by training sets of closely related genomes with well-characterized prophages.
Figure 3.7: Comparison of the abundance of phage words in bacteria (∆) and phage genomes (x). (A) The Shannon’s index (H) versus the frequency (F) of the presence of phage words for 600 complete phage genomes and 400 randomly chosen complete bacterial genomes. Both H and F are very small for bacterial genomes compared to phage genomes. The relationship between H and F for phages is \( F = 8.57H + 0.047 \) with a regression coefficient \( R^2 = 0.995 \) and for bacterial genome the relation is \( F = 5.85H + 0.014 \) with a regression coefficient \( R^2 = 0.993 \). (B) The ratio of the frequency and Shannon’s index, i.e. F/H for 600 complete phage genomes and 400 randomly chosen complete bacterial genomes. There is a statistically significant difference in F/H (abundance of phage words) between phages and bacteria.
Table 3.1: Performance analysis of PhiSpy and comparison with Phage finder and prophinder. The genomes with gray background have manually annotated prophages [90–108].

<table>
<thead>
<tr>
<th>Training Organism</th>
<th>Organism</th>
<th>Ref. Prophage</th>
<th>PhiSpy</th>
<th>Phage finder</th>
<th>Prophinder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bacillus subtilis</td>
<td>Bacillusicheniformis ATCC 14580</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Brucella melitensis</td>
<td>Brucella abortus biotype 1 strain 9-941</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
<td>Escherichia coli APEC O1</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
<td>Escherichia coli K12</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Haemophilus influenzae Rd KW20</td>
<td>Haemophilus influenzae Rd KW20</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Lactococcus lactis subsp. lactis BI-43</td>
<td>Lactococcus lactis subsp. cremoris SK41</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Listeria monocytogenes</td>
<td>Listeria monocytogenes 4037</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Listeria innocua</td>
<td>Listeria innocua</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mycobacterium tuberculosis</td>
<td>Mycobacterium avium subsp. paratuberculosis subsp. subsp. paratuberculosis</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mycobacterium tuberculosis</td>
<td>Mycobacterium avium subsp. paratuberculosis subsp. paratuberculosis</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mycobacterium tuberculosis</td>
<td>Mycobacterium avium subsp. paratuberculosis subsp. paratuberculosis</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mycobacterium tuberculosis</td>
<td>Mycobacterium avium subsp. paratuberculosis subsp. paratuberculosis</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pseudomonas aeruginosa</td>
<td>Pseudomonas aeruginosa</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pseudomonas putida KT2440</td>
<td>Pseudomonas putida KT2440</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Pseudomonas aeruginosa</td>
<td>Pseudomonas aeruginosa</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Pseudomonas aeruginosa</td>
<td>Pseudomonas aeruginosa</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pseudomonas aeruginosa</td>
<td>Pseudomonas aeruginosa</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pseudomonas putida KT2440</td>
<td>Pseudomonas putida KT2440</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pseudomonas putida KT2440</td>
<td>Pseudomonas putida KT2440</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Salmonella enterica subsp. enterica paratyphi A</td>
<td>Salmonella enterica subsp. enterica paratyphi A</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Shewanella oneidensis</td>
<td>Shewanella oneidensis</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Shewanella oneidensis</td>
<td>Shewanella oneidensis</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Shewanella oneidensis</td>
<td>Shewanella oneidensis</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Shewanella oneidensis</td>
<td>Shewanella oneidensis</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus</td>
<td>Staphylococcus aureus subsp. aureus</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus</td>
<td>Staphylococcus aureus subsp. aureus</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus</td>
<td>Staphylococcus aureus subsp. aureus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus</td>
<td>Staphylococcus aureus subsp. aureus</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Streptococcus agalactiae</td>
<td>Streptococcus agalactiae</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS2232</td>
<td>Streptococcus pyogenes MGAS2232</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS2232</td>
<td>Streptococcus pyogenes MGAS2232</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS2232</td>
<td>Streptococcus pyogenes MGAS2232</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS2232</td>
<td>Streptococcus pyogenes MGAS2232</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS2232</td>
<td>Streptococcus pyogenes MGAS2232</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS2232</td>
<td>Streptococcus pyogenes MGAS2232</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS2232</td>
<td>Streptococcus pyogenes MGAS2232</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS2232</td>
<td>Streptococcus pyogenes MGAS2232</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS2232</td>
<td>Streptococcus pyogenes MGAS2232</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Xanthomonas axonopodis pv. citri strain 306</td>
<td>Xanthomonas axonopodis pv. citri strain 306</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Yersinia pestis CO92</td>
<td>Yersinia pestis CO92</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Yersinia pestis CO92</td>
<td>Yersinia pestis CO92</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Total: 141 | 123 | 134 | 16 | 16 | 12.0 | 11

FP% 0.66667 | 1.33333 | 0.60 | 0.60 | 0.10 | 0.11
3.4.6 Performance Analysis

PhiSpy was used to predict prophages in 95 complete bacterial genomes using the training set of closely related genomes and 412 complete bacterial genomes using a universal generic training set (these predicted prophages are accessible at http://www.phantome.org/Downloads/Prophages/PhiSpy/ and http://nar.oxfordjournals.org/content/40/16/e126/suppl/DC1). A detailed flow chart of the performance analysis is described in the supplemental figure 5. In the 95 genomes, 320 potential prophages were identified. Among those, three prophages (in *Streptococcus Agalactiae NEM316*) had no phage-like proteins and were considered as previously undefined prophages (Supplemental Table 2). For performance analysis, the predicted prophages were manually checked (based on the phage subsystems) for 50 genomes (out of the 95 genomes). Most of the genomes with manually annotated prophages [56] were not used in the performance analysis because these prophages were also used to parameterize one of the five criteria (abundance of phage words) developed to identify prophages, and so the result will be biased. We did not test whether phiSpy could detect those prophages used in training the classifier, as that would be a biased assessment. However, to compare with the published data, four genomes were considered which have manually annotated prophages (gray colored genomes in table 3.1). For *Streptococcus pyogenes M1 GAS* and *Streptococcus pyogenes MGAS315*, the identified prophages matched with the published data. In *Escherichia coli O157:H7 EDL933*, 13 prophage regions were found to contain 17 prophages by phiSpy although 18 prophages were found in the original analysis of this genome [90]. Of those 18 published prophages, there were four instances where two adjacent prophages were merged by phiSpy. The prophage that was not identified is a short prophage (8.26 kb) and only has four phage-like proteins. The same reason goes for the unidentified prophage in *Neisseria meningitidis Z2491* [Table 3.1]. The three unidentified prophages in *Pseudomonas*

65
fluorescens Pf-5 are defective prophages [101].

<table>
<thead>
<tr>
<th></th>
<th>Closely related genomes</th>
<th>Distantly related genomes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GC/AT rich genome</td>
<td>Moderate GC% genome</td>
</tr>
<tr>
<td>Transcription Strand</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>Directionality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protein Length</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Customized AT skew</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Customized GC skew</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Abundance of phage</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

1 shows the effectiveness of different characteristics when the training genome and test genome are closely related
2 shows the effectiveness of different characteristics when the training genome and test genome are distantly related

Table 3.2: Effectiveness of different characteristics for identifying prophages in different genomes.

To compare the performance of phiSpy with other phage finding tools, phiSpy, phage_finder, prophinder and the DRAD method were used to predict prophages in 50 genomes (using default settings). For DRAD, no prophages were identified. As shown in table 3.1, phiSpy identified 94% of the prophages with a 6% false negative (FN) rate and a 0.66% false positive (FP) rate, whereas phage_finder predicted 82% of the prophages with a 18% false negative rate and a 1.33% false positive rate and prophinder identified 89% prophages with a 11% false negative rate and a 12% false positive rate (the predicted prophages from phiSpy, phage_finder and prophinder are available at http://www.phantome.org/Downloads/Prophages/PhiSpy/Manually\_Verified/). Therefore, phiSpy can identify more prophages than other phage finding tools with the lowest false positive rate and false negative rate. For prophinder, most of the false positive prophages have a low score. Most of the prophages that were not reported by phiSpy were mainly misclassified in the evaluation step, because there were few proteins annotated as phage proteins in those prophage regions. Also, some comparatively short prophages which were unidentified by
phage_finder were also missed in our classification step but found in our evaluation step.

Figure 3.8: A) Comparative analysis of all prophages identified in 412 complete bacterial genomes by phiSpy, phage_finder and prophinder. B) Comparative analysis of undefined prophages (no phage-like proteins) identified from 412 complete bacterial genomes.

From 412 complete bacterial genomes where no closely related genomes with manually annotated prophages were available to construct training sets, 826 prophages were predicted by phiSpy and 284 of those were considered previously undefined, as they have no known phage genes. Therefore, phiSpy can detect potentially new prophages without relying on a training set. To check whether other phage finding applications can identify the prophages having hypothetical or unknown proteins, phage_finder and prophinder were used to predict prophages in those 412 bacterial genomes [Figure 3.8, http://nar.oxfordjournals.org/content/40/16/e126/suppl/DC1]. Phage_finder identified 378 prophages where 22 of them are undefined prophages and prophinder predicted 339 prophages where 43 of them are undefined prophages [Figure 3.8]. There are 198 known prophages and zero undefined prophages identified in common between phiSpy, phage_phinder and prophinder. There are 52 prophages predicted by both prophinder and phiSpy but not predicted by phage_finder, 120 prophages predicted by both phiSpy and phage_finder but not predicted by prophinder, and only two prophages identified by
both phage_finder and prophinder but not predicted by phiSpy. Hence, phiSpy can predict more known and undefined prophages compared to those tools. The prediction of novel prophages in genomes is only the first step: biological experiments are required to investigate whether these phage-like regions are viable or can be induced out of the chromosome. Recently an approach was described to test prophage viability in *Salmonella enterica* that could be used to test some of these prophages [109].

PhiSpy was used to identify prophages on a large number of bacterial genomes using the infrastructure of kbase (http://kbase.us/). In 4,255 bacterial genomes, phiSpy predicted 12,824 prophages. Among these prophages, there are 9,101 known prophages and 3,723 undefined prophages that have no phage like protein. As shown in table 3.3, *E. coli* contains the maximum number of prophages. There are 218 strains of *E. coli* and they have 2,194 prophages, that means on an average each *E. coli* has about 10 prophages. *Salmonella enterica* contains the second maximum number of prophages and on an average this organism has about 3 prophages.

<table>
<thead>
<tr>
<th>Organism</th>
<th># Prophages</th>
<th># of Genomes</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Escherichia coli</em></td>
<td>2,194</td>
<td>218</td>
</tr>
<tr>
<td><em>Salmonella enterica</em></td>
<td>372</td>
<td>100</td>
</tr>
<tr>
<td><em>Staphylococcus aureus</em></td>
<td>346</td>
<td>91</td>
</tr>
<tr>
<td><em>Bacillus cereus</em></td>
<td>265</td>
<td>51</td>
</tr>
<tr>
<td><em>Enterococcus faecalis</em></td>
<td>197</td>
<td>56</td>
</tr>
<tr>
<td><em>Shigella flexneri</em></td>
<td>177</td>
<td>20</td>
</tr>
<tr>
<td><em>Yersinia pestis</em></td>
<td>166</td>
<td>29</td>
</tr>
<tr>
<td><em>Bacillus thuringiensis</em></td>
<td>164</td>
<td>28</td>
</tr>
<tr>
<td><em>Clostridium difficile</em></td>
<td>142</td>
<td>27</td>
</tr>
<tr>
<td><em>Bacillus anthracis</em></td>
<td>134</td>
<td>26</td>
</tr>
</tbody>
</table>

Table 3.3: Top ten organisms that have the maximum number of prophages
3.5 Discussion

In this report we describe the identification of prophage regions within bacterial genome sequences. We have advanced the current analysis of prophage identification by introducing five distinctive characteristics of prophages that do not depend on sequence similarities. These characteristics were applied for the initial prediction of potential prophages, and each of these predictions was evaluated by identifying the phage insertion point and the similarity of phage proteins. PhiSpy was used to predict prophages in 507 complete bacterial genomes and a total of 1,146 potential prophages were identified, including 287 putative prophages that have no homology to existing phages and may be novel mobile genetic elements. However, the total number of identified prophages might be different than the actually identified prophages for two reasons: (i) if there are several short prophage regions in close proximity then one large phage region might be reported instead of several shorter ones; (ii) if more than one integrase was found in a single predicted prophage, more than one prophage might be reported.

Despite the use of multiple distinctive parameters to classify genes within a genomic region, the current random forest protocol does not allow the accurate determination of the phage start and end. To address this issue, we resorted to the analysis of phage attachment sites by detecting direct or inverted repeats, which are common at the insertion sites of most phages. As insertions are often flanked by several repeated sequences, two criteria were used to consider for all the candidate att sites: (i) proximity to tRNA or integrase genes, as phages can integrate into tRNA/tmRNA genes and the integrase gene is often at the end of the prophage [56]; (ii) inclusion of the greatest number of proteins thought to be included in phage subsystems, to provide further confidence in the prediction of the att sites.

The classification step of phiSpy predicts prophage regions more accurately if it is
trained with genomes that are closely related to the test genome because: (i) differences in the GC% between the training genome and the test genome result in an incorrect weight for the customized AT/GC skew; (ii) differences in the protein length between the training genome and the test genome result in the wrong protein length prediction parameters; (iii) different operon sizes between the training genome and the test genome result in incorrect transcriptional strand orientation parameters; (iv) finally, if the test genome (or a genome closely related to the test genome) was not included in the phage word library, two circumstances might occur: (a) some words in the phage library may match with the test genomes word (b) the genome might have some different prophages whose distinct words are absent in the library, which leads to a bad prediction for the calculation of the abundance of phage words.

The prophages not identified by PhiSpy in the classification step, might fall into one of three categories: (i) if there are few phage genes in a window whose characteristics were dominated by the bacterial genes of that window and those few phage genes were missed; (ii) if several short prophages are located very close together (this often happens in *E. coli* or *Salmonella*), they were identified either as one long prophage or some of them were ignored in the processing of the final rank; (iii) while processing the final rank, sometimes the prediction of prophages was skewed leftwards by its windowing process (which proceeds left to right). The reason for the prophages identified in classification step but misclassified in evolution step is either there are few proteins annotated as phage proteins in the prophage regions or the function of the phage proteins are not yet annotated into phage subsystems.

Some of the characteristics of prophages used here have been described in previous studies, but none of the prophage identification tools applies them for identifying novel prophages. In this paper, we have combined two approaches (similarity-based and composition-based analysis) and come up with an automated application that can identify prophages with or without the homology to known phage genes.
3.6 Acknowledgment

I thank Dr Barbara A. Bailey, for insightful discussion and helpful suggestions on statistical significance analysis.

This research is funded by the Advances in Bioinformatics from the National Science Foundation (http://www.nsf.gov/) [DBI 0850356].

This chapter contains materials from "PhiSpy: a novel algorithm for finding prophages in bacterial genomes that combines similarity and composition-based strategies" by Sajia Akhter, Ramy K. Aziz and Robert A. Edwards, which appears in Nucleic Acids Research, 2012. The dissertation author was the primary investigator and author of this paper. There is a license agreement between Sajia Akhter and Oxford University Press for using this paper in this dissertation and the license number is 3161760987620.
Chapter 4

Applying Shannon's Information Theory to Bacterial and Phage Genomes and Metagenomes

4.1 Information Theory in Genomic Data

The extraordinary advances in speed and throughput of sequencing technologies in the past decade have generated an unprecedented wealth of complete or near complete genome sequences, and have also allowed the emergence of the technology of metagenomics or random community genomics, which aims at sequencing DNA from environmental microbial communities without culturing or isolating individual microbes. Today thousands of fully sequenced genomes and over 7,000 metagenomes have been deposited in public repositories, e.g., GenBank [110], Genomes Online Database (GOLD) [111], the SEED database [112], and the Metagenomics RAST (MG-RAST) server [113]. To be annotated and analyzed, metagenome sequences are compared to genes, proteins, protein domains, protein families, and genomes in known databases. It was shown a few years ago that approximately 19 hours were needed to analyze one megabase of DNA sequence (if linear
compute complexity is assumed), and each data set required about one month of computing time [114]. However, MG-RAST and other public services handle the analysis by using large compute clusters dedicated to sequence searching. Because of the deluge of sequence data, new efficient tools and methods are required for analyzing and comparing sequences, and for prioritizing the sequences to be analyzed when comprehensive analysis is not feasible.

One approach to prioritizing the analysis of unknown genomic or metagenomic sequences is examining the information content of known genes, proteins, and genomes to explore possible patterns or trends that might help in predicting informative sequences, i.e., those sequences likely to encode proteins or to provide new rather than redundant knowledge about the sample to which they belong. In the cell, the information flows from DNA to amino acid sequences, as DNA is transcribed into RNA then translated into amino acids to make proteins. Depending on the different combinations of bases in the deoxyribonucleotides of the DNA sequence, different amino acids are added to the nascent, growing polypeptide chain. Complex proteins consist of different combinations of amino acids and therefore are encoded by various combinations of the four sequence bases. Homopolymeric tracts like AAAAAAAAAC or TTTTTTTCCCCC can only code one or few different amino acids and encode for proteins with amino acid repeats. Therefore, we hypothesize that they are much less likely to encode functional proteins than DNA containing equimolar mixtures of bases (e.g., AGCTAGCTAGCT).

Statistical approaches derived from information theory can quantify the amount of information in a DNA sequence. Several investigators have examined different aspects of information content of genomes, including Shannon’s uncertainty [115–119] and symmetry [11, 120]. For example, Chang and coworkers calculated Shannon’s uncertainty index for all the complete prokaryotic and eukaryotic genomes available in 2005. They found that Shannon’s information in complete genomes is greater than that in matching random
sequences and they described a coarse-grain model for genome growth and evolution that allows a genome to diverge at any stage during its growth [116, 117].

Shannon’s uncertainty [73] was originally designed for encoding and decoding data transmitted and received through a digital communication system. Since sequence data can also be represented as a system where DNA is transformed into amino acids, this theory can be used to calculate the amount of information or uncertainty of a sequence (http://www.ccrnp.ncifcrf.gov/~toms/paper/primer). For each sequence, the uncertainty measurement per base pair generates a score from 0 to 2^n, where n is word length. The greater the uncertainty, the more even is the distribution of each word. For example, the sequence AAAAA can only be read using two letter words as AA regardless of the register and has little uncertainty. In contrast, the sequence ACGT can be read as AC, CG and GT, depending on the register and has more inherent uncertainty and information.

Here, the information content was examined for complete bacterial and phage genomes, and the analysis was extended to the calculation of Shannon’s uncertainty for each sequence within metagenomic libraries. The effects of word size, genome length, and GC% on Shannon’s uncertainty have also been examined. We demonstrate that the information content of sequences from metagenomes correlates with the number of similar sequences that is found by comparison to databases of known sequences. Using this approach may speed up the processing time for analyzing metagenomic data and allow prioritization of computational resources.
4.2 Results

Shannon’s index was calculated for 600 complete phage genomes and 94 complete bacterial genomes (http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3539204/#s1) using word lengths ranging from 1 to 12 nucleotides (nt). Shannon’s indices of phage and bacterial genomes were similar up to word length 7 nt [Figure 4.1], implying an even distribution of all possible sequence words in phage and bacterial genomes. From word length 8 to 12 nt, the rate of increase of Shannon’s index is higher in bacterial genomes than in phage genomes. Moreover, for word lengths greater than 10 nt, Shannon’s index can differentiate bacteria and phages [Figure 4.1].

4.2.1 Factors Influencing Differences in Shannon’s Uncertainty

The difference between Shannon’s indices of phage and bacterial genomes for word length greater than 8 nt suggested that either word length, genome size or a combination of both might influence this uncertainty value.

**Shannon’s index vs. word length:** Word length is reportedly an important factor influencing the value of Shannon’s index [116, 117]. A high Shannon’s index (close to the maximum possible index, i.e., for word length \( n \), the maximum index will be \( 2^n \)) depends on the presence of all possible combinations of words in the genome. Consequently, the longer the genome the higher the probability of having different word variations. For a given word length of \( n \), there are \( 4^n \) possible word combinations for DNA sequences. The length of most phage genomes (585 out of 600) ranges from \( 4^7 \) bp to less than \( 4^9 \) bp [Figure 4.2]. Therefore, for word size greater than 8 nt, many words will only be represented zero or one times, which will result in a lower Shannon’s index for most of these genomes. In contrast, the average length of the 94 bacterial genomes used in this analysis is about 3 million bp (between \( 2 \times 4^{10} \) and \( 4^{11} \) bp). Therefore, bacterial genomes have a higher
Figure 4.1: Shannon’s indices of 600 complete phage genomes and 94 complete bacterial genomes. Blue crosses represent phage genomes and red circles represent bacterial genomes. As the word length increases, Shannon’s index is more discriminatory between phage and bacterial genomes.

Shannon’s index than phage genomes using word lengths smaller than 12 nt. For word lengths 11 nt or 12 nt, Shannon’s index can distinguish phage and bacterial genomes [Figure 4.1] although this is likely because phage genomes are too short to generate sufficiently high Shannon’s indices for words of this size.

**Shannon’s index vs. genome length:** Shannon’s indices for all phage genomes have been plotted against their lengths. For word length 12 nt, Shannon’s index highly correlates with the logarithm of the genome length [Figure 4.3]. For word length 9 nt, there is still a significant correlation; however, for shorter word length, no significant correlation was observed between genome length and Shannon’s index. For shorter word lengths, most of
the genomes have almost all combination of words in their genomes, so there is no strong correlation between Shannon’s index and genome length. In contrast, for longer words, the bigger genomes have more combinations of words than the smaller genomes; so Shannon’s index correlates with genome length.

Calculations with irrelevant word lengths may give the wrong impression and create false differences between genomes [Figure 4.1]. To compare genomes based on Shannon’s index, the word length \( n \) should be chosen in a way that allows the possibility of having all combination of words \( (4^n) \) in all the genomes. Therefore, for a given genome of length \( L \), the possible word length \( n \) to calculate Shannon’s index should be

\[
4^n < L \Rightarrow n \leq \left\lfloor \frac{1}{2} \log_2 L \right\rfloor
\]
Figure 4.3: Shannon’s index vs. length for 600 complete phage genomes using word length 6, 9 and 12.
Figure 4.4: Shannon’s index vs. GC% for 600 complete phage genomes using word length 1 nt to 7 nt.
Figure 4.5: The relationship between Shannon’s index and $|\text{GC\%} - 0.5|$ for 600 complete phage genomes using word length 1 nt to 7 nt.
Shannon’s index vs. GC%: For most phage genomes, the maximum word length that should be used to calculate Shannon’s index is 7 nt. When word lengths from 1 to 7 nt were used to calculate Shannon’s index, GC-rich and GC-poor genomes were found to have lower Shannon’s index since these genomes tend to have less diverse word combinations than genomes with 50% GC content [Figure 4.4]. The strong relationship between Shannon’s index and |GC% - 0.5| for word length 1 to 5 nt suggests that Shannon’s index is strongly influenced by the GC composition of the DNA sequence [Figure 4.5]. For word lengths above 6 nt, the relationship is not strongly supported. Different sequences may have the same GC%, but Shannon’s index depends on the distribution of the different word combinations. Therefore two different sequences having the same GC% may have different Shannon’s indices, and the probability of this happening increases with the word length. Thus, as word length is increased, the correlation between Shannon’s index and GC content becomes weaker [Figure 4.5].

4.2.2 Shannon’s Uncertainty in Metagenomes

Shannon’s uncertainty was calculated for different metagenomic data sets. The maximum uncertainty equates to a sequence that has equal frequencies of each word (e.g. A, G, C, T for word length one) and the majority of reads in a metagenome have an uncertainty greater than 1.8 per nt [Figure 4.6] suggesting an even distribution of bases in the reads, although the relative information content of the reads varies by sample.

To investigate whether information content correlates with functional content, we compared the frequencies with which each sequence matched an entry in the known databases. The similarities between the metagenomic sequences and the SEED non-redundant protein database had been pre-calculated using BLASTX [11, 43] as a part of the annotation and analysis procedure performed by the MG-RAST server [113]. For a set of reads with a given uncertainty, the fraction of reads that were similar to sequences in the SEED non-
Figure 4.6: Cumulative comparison of the uncertainty (for word length 1) in DNA sequences in metagenome samples. Eight samples representative of the 24 used in this study are shown here: Soudan Mine Black Stuff (pink [121]), Line islands Kingman reef phage (light green [122]), Line islands Tabuaren phage (light blue [122]), Marine phages from the Gulf of Mexico (blue [123]), Marine samples supplemented with DMSP (magenta [124]), Line islands Palmyra Phage (dark green [122]), Line islands Christmas Reef phage (red [122]), Marine samples supplemented with vanillate (green [124])). The uncertainty is greater than 1.7 for 85% to 90% sequences of all samples.

A redundant database was extracted from these pre-calculated similarities [Figure 4.7]. A read with more information (higher uncertainty) was more likely to be similar to sequences in the database than a read with less information. Different metagenomes varied in the fraction of reads that are similar to known sequences, but this likely reflects the sampling limitations that have thus far limited the breadth of the known sequences [125].

4.3 Discussion

Since the publication of the first complete genome sequences, genome composition has been appealing to mathematicians, statisticians, and computer scientists. Base distribution
Figure 4.7: Comparison of Shannon’s uncertainty and the observed similarity to known sequences. Shannon’s uncertainty (H) was calculated for word length one, and is compared with similarity to the SEED no-redundant protein database. Samples are coloured the same as in Fig. 5. Word lengths up to 11 letters were also used to calculate (H) and all cases confer same results (data not shown).

statistics, skews and biases [71, 126–129], sequence symmetries [73, 120], and information content have all been examined in the hope of deciphering hidden codes within the genomes [73] and better understanding genome growth and evolution [117, 130–134].

Among the mathematical methods used, Shannon’s uncertainty has previously been considered as a genome analysis strategy [115–118]. In the work of Chang and colleagues [116, 117], Shannon’s uncertainty was calculated for complete prokaryotic and eukaryotic genomes available at that time, and it was found that genomes belonged to a universality class that could be mathematically represented by a simple formula, yet Plasmodium genomes stood out as an intriguing exception, still unexplained. Additionally, the variation of Shannon’s index with sequence word length and genome length was examined [116, 117]. Here, our findings confirmed and advanced that study by establishing the relationship between word size and genome length for calculating Shannon’s index.

We also found that at a certain word lengths, Shannon’s index can be used to differ-
entiate phage and bacterial sequences. Although this differentiation is sensitive to genome length, with some modification, this observation can help find phage genes embedded in bacterial genomes. As an application, we calculated Shannon’s index for a group of DNA sequences using a word size of 12 nt (four consecutive amino acids) and we were able to use this group of sequences to detect prophages in bacterial genomes [135].

Finally, our findings show that the information content of metagenomic sequences varies from sample to sample, but about 85% of those sequences have high levels of uncertainty, suggesting that they are comprised of approximately equal numbers of each of the four bases [Figure 4.6, 4.7]. In addition, the information content in metagenomic sequences was found to correlate with the likelihood that the sequence would be similar to a previously characterized sequence (in the non-redundant database). This suggests that the large numbers of metagenomic sequences could be rapidly sorted based on their information content to prioritize similarity searches and other common computations. It is to be noted, however, that those metagenomic sequences have to be preprocessed and cleared of potential repeats or homopolymeric runs, sometimes introduced by sequencing methods (e.g., the introduction of runs of nucleotides during high-throughput sequencing). For this purpose, tools such as PRINSEQ [135] MG-RAST can be used prior to sequence analysis of metagenomic data sets. Moreover, the correlation between information content and similarity may provide a rapid mechanism to screen for either false positive matches (sequences matching the database that should not) or false negative matches (sequences with no match in the database, but that should). Of course, the extremely large numbers of sequences with high uncertainty but no similarity in the databases might be influenced by the lack of sampling in the known databases [136].
4.4 Methods

4.4.1 Retrieval of Genomic and Metagenomic Data

All genomes used in this analysis were retrieved from the SEED database and servers [112] (http://servers.theseed.org), where they have been consistently annotated and classified into subsystems [137, 138] in the RAST server (http://rast.nmpdr.org). Likewise, metagenomic sequence data sets were retrieved from the MG-RAST server (http://metagenomics.theseed.org).

For the calculation and analysis of Shannon’s uncertainty, a subset of 24 metagenomes was selected from the previously studied SCUMS data set [121], most of which were created by pyrosequencing. The metagenomes were chosen to represent the range of data sets available from sequences sampled in simple and well-characterized environments to more complex environments with multiple species present. The raw data were used without assembly, and the samples included in the data set cover both viral and microbial metagenomes, sampled from such diverse biomes as mines, marine environment, soils, and animals [121–124, 139]. The shortest sequence in the data set was 31 bp and the longest was 362 bp.

4.4.2 Calculation of Shannon’s Uncertainty

Shannon’s uncertainty was calculated using the following equation [73],

$$ - \sum p_i \log(p_i) $$

where $p_i$ is the frequency of the i-th word in a sequence. For example, for word length one, $p_i$ is calculated from the frequencies of the nucleotides A, G, C, T. If each word is equally frequent, $p_i = 0.25$. In general, for all words of length $n$ being equally likely, $p_i$ is $1/4n$. 

85
4.5 Acknowledgement

This work was supported by grant NSF DBI-0850356 from the NSF Division of Biological Infrastructure to RAE (PhAnToMe project). This chapter is a reprint of: Sajia Akhter; Barbara A. Bailey; Peter Salamon; Ramy K. Aziz; Robert A. Edwards. Applying Shannon’s information theory to bacterial and phage genomes and metagenomes. Nature Sci Rep. 2013; 3: 1033.
Chapter 5

Kullback Leibler Divergence in Complete Phage and Bacterial Genomes

The central dogma of molecular biology describes the irreversible flow of information in biological systems from nucleic acids to amino acids, whose combinations make up the main cellular components—proteins. In principle, such flow of information is no different from other data storage and communication systems, and can thus be studied by information theory [73]. Indeed, information theory has been often applied to study different aspects of prokaryotic and eukaryotic genomes, including, for example, genome composition [126,130], architecture [132,140,141], coding potential [114], order and entropy [142], and symmetry [119, 120]. Previous studies used Shannon’s index [73] to classify informative DNA sequences [115–117, 143] and find prophages in bacterial genomes [134].

In an attempt to prioritize analysis efforts for high-throughput sequencing data sets, notably metagenomic data sets, we calculated the Shannon’ index of a large sample of bacterial and phage genomes, and showed that the information content of a genome’s nucleotide sequence largely depends on the genomes size and its GC content. Subsequently, we were able to predict which sequences within a metagenomic library are more likely to match sequences already deposited in public databases [143]. Here, we continue to ex-
plore the information theory by expanding our analysis to the coding potential of a genome, focusing on amino acids rather than nucleotide content. To that end, we used the Kullback-Leibler divergence (KLD) value [144] to examine biases in the amino acid composition of the potentially translated gene products (predicted proteins) encoded by a genome.

5.1 Kullback Leibler Divergence

Kullback and Leibler generalized Shannon’s approach to support statistical comparisons between populations [144]. The KLD measures the deviation of one distribution from another distribution. In this study, we hypothesized that KLD might be a good measure of the diversity of an encoded proteome. We demonstrate that KLD correlates well with an organism’s phylogeny, lifestyle (whether it is free-living or host-dependent), and amino acid utilization profile, in addition to correlating with the GC content of bacterial genomes.

5.2 Results

5.2.1 Divergence of Amino Acid Composition in Genomes

KLD was calculated for all predicted proteins encoded by 372 bacterial genomes and 835 phage genomes. The skew in the KLD distribution, for all genomes combined, ranged from 0.002 to 0.22 [Figure 5.1]. We found that both the most skewed bacterial genome, *Wigglesworthia glossinidia* (KLD = 0.224), and the most skewed phage genome, *Spiroplasma kunkelii virus SkV1.CR2-3x* (KLD = 0.222), had a low GC content of about 22%. We also found that phage genomes, in general, tended to have lower KLD values than bacterial genomes. This finding suggests that bacteria might have more biased amino acid utilization than phages.
Figure 5.1: Trends in amino acid composition divergence. A. The 372 complete bacterial genomes (black) and 835 complete phage genomes (blue) analyzed are ranked according to their composition. B. Box plots showing the amino acid composition divergence for bacteria (gray) and phages (blue).
Figure 5.2: Amino acid divergence varies for each phylogenetic taxon of bacteria and phage bacterial hosts. The divergence of amino acid composition for each phylogenetic group from the mean of all bacteria and phages is shown. Error bars represent the standard error of the mean. The numbers represent the number of genomes considered for each class.

The phylogeny and lifestyle of the ten bacterial species and ten phages with the most and skewed amino acid composition (as measured by their KLD values) are shown in Table 5.1 and Table 5.2 respectively. Consistently, the bacterial species whose genomes have the most skewed amino acid compositions are obligate intracellular parasites, with a very limited ecological niche range and restricted lifestyle (e.g., Wigglesworthia glossinidios, an endosymbiont of the tsetse fly, Table 5.1). Likewise, the bacterial species that are hosts for the phage genomes with the most skewed amino acid compositions are intracellular parasites (e.g., Spiroplasma kunkelii, a parasite that causes Corn Stunt Disease, Table 5.2).

There are several possible explanations for these deviations in amino acid distribution. For example, the variation could be phylogenetically biased or determined by the lifestyle of the organism. Alternatively, a physical parameter such as the DNA composition, osmotic, or thermodynamic stability might control the variation in production of amino acids.
<table>
<thead>
<tr>
<th>Genus and Species</th>
<th>KLD of amino acid composition from the mean</th>
<th>%GC</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wigglesworthia glossinidia endosymbiont of Glossina brevipalpis</td>
<td>0.224</td>
<td>22.5</td>
<td>Wigglesworthia are obligate intracellular bacteria and endosymbionts of the tsetse fly</td>
</tr>
<tr>
<td>Mycoplasma mobile 163K</td>
<td>0.162</td>
<td>25</td>
<td>Fish pathogen</td>
</tr>
<tr>
<td>Mycoplasma mycoides subsp. mycoides SC str. PG1</td>
<td>0.162</td>
<td>24</td>
<td>Cattle pathogen</td>
</tr>
<tr>
<td>Borrelia burgdorferi B31</td>
<td>0.162</td>
<td>28.2</td>
<td>A human pathogen that lives in rodents, and can be transferred to humans via tick bites</td>
</tr>
<tr>
<td>Borrelia garinii PBi</td>
<td>0.162</td>
<td>28.1</td>
<td>A human pathogen that lives in rodents, and can be transferred to humans via tick bites</td>
</tr>
<tr>
<td>Mycoplasma hyopneumoniae 232</td>
<td>0.156</td>
<td>28.6</td>
<td>Pig pathogen responsible for porcine pneumonia</td>
</tr>
<tr>
<td>Ureaplasma parvum serovar 3 ATCC 700970</td>
<td>0.155</td>
<td>25.5</td>
<td>Mucosal pathogen of humans</td>
</tr>
<tr>
<td>Mycoplasma hyopneumoniae 7448</td>
<td>0.154</td>
<td>28.5</td>
<td>Pig pathogen responsible for porcine pneumonia</td>
</tr>
<tr>
<td>Mycoplasma pulmonis UAB CTIP</td>
<td>0.154</td>
<td>26.6</td>
<td>Mouse pathogen causing murine pneumonia</td>
</tr>
<tr>
<td>Mycoplasma hyopneumoniae J</td>
<td>0.153</td>
<td>28.5</td>
<td>Pig pathogen causing porcine pneumonia</td>
</tr>
</tbody>
</table>

Table 5.1: The most skewed bacterial genomes.

and composition of the proteins. The availability of amino acids, their precursors, or enzymatic limits on the interconversion of amino acids may also affect these skews.

To investigate the impact of phylogeny on KLD deviations, we calculated the mean KLD for each phylogenetic group and compared it to the mean KLD of all proteins. The variation in amino acid composition provides a signature profile for each phylogenetic group (for both phage and bacterial genomes, figure 5.2), which might be predictive for metagenomic sequences.
Table 5.2: The most skewed phage genomes [145].

### Table 5.2: The most skewed phage genomes [145].

<table>
<thead>
<tr>
<th>Virus type</th>
<th>KLD of amino acid composition from the mean</th>
<th>%GC</th>
<th>Comments on the host</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Spiroplasma kunkelii virus SKV1.CR2-3x</em></td>
<td>0.222</td>
<td>22.2</td>
<td>Parasitic lifestyle Causative agent of corn Stunt Disease</td>
</tr>
<tr>
<td><em>Spiroplasma phage SVTS2</em></td>
<td>0.211</td>
<td>22.7</td>
<td>Parasitic lifestyle</td>
</tr>
<tr>
<td><em>Spiroplasma phage I-C74</em></td>
<td>0.199</td>
<td>23.1</td>
<td>Parasitic lifestyle</td>
</tr>
<tr>
<td><em>Propionibacterium phage B5</em></td>
<td>0.192</td>
<td>64.3</td>
<td>A parasite and commensal of humans and other animals; Lives in and around the sweat glands, sebaceous glands and other areas of the skin</td>
</tr>
<tr>
<td><em>Spiroplasma phage I-R8A2B</em></td>
<td>0.186</td>
<td>22.9</td>
<td>Parasitic lifestyle</td>
</tr>
<tr>
<td><em>Acholeplasma phage MV-L1</em></td>
<td>0.183</td>
<td>33.3</td>
<td>N/A</td>
</tr>
<tr>
<td><em>Mycoplasma phage phiMFV1</em></td>
<td>0.167</td>
<td>25.1</td>
<td>Parasitic lifestyle</td>
</tr>
<tr>
<td><em>Clostridium phage D-I873 CLG.Contig168</em></td>
<td>0.153</td>
<td>25.3</td>
<td>N/A</td>
</tr>
<tr>
<td><em>Mycoplasma phage P1</em></td>
<td>0.152</td>
<td>26.8</td>
<td>Parasitic lifestyle</td>
</tr>
<tr>
<td><em>Clostridium phage c-st</em></td>
<td>0.133</td>
<td>26.3</td>
<td>N/A</td>
</tr>
</tbody>
</table>

N/A: No data available

### 5.2.2 GC Content and Amino Acid Variations among Genomes

The comparisons of amino acid deviation between different phages and bacteria is shown in figure 5.1 and figure 5.2. To inspect the functional significance of those differences, we compared the composition of proteins involved in different aspects of metabolism. In this comparison, the null hypothesis was that the compositional bias was randomly distributed among all protein metabolic functional classes, and the alternative hypothesis was that the bias was limited to one or a few functional groups that might contain critically skewed amino acid compositions in some genome. To address this potential source of bias, we used SEED subsystems [136], collections of genes in pathways or functional associations manually curated by teams of annotators in the SEED database [112, 138]. Different subsystems are arranged in a hierarchy of groups.
Figure 5.3: Comparison of the divergence of amino acid composition for the most divergent bacterial genomes (top panel) and the genomes of five bacteria chosen at random (bottom panel). In (a) the first five bars are *Wigglesworthia glossinidia*, *Borrelia garinii*, *Mycoplasma mycoides*, *Ureaplasma parvum serovar* and *Buchnera aphidicola* (see Table 5.1). In (b) the first five bars are *Bifidobacterium adolescentis*, *Bacillus B-14905*, *Nostoc sp. PCC 7120*, *Salmonella bongori 12149*, and *Chlamydophila pneumoniae CWL029*. In both panel the sixth bar is for the mean of amino acid utilization for each subsystem. (Note the difference in y-axis scale between two panels).
Figure 5.4: Frequency of each of the twenty amino acids in the three domains of life and the most skewed genomes. The first three bars are the average frequency of amino acid for the three domains Archea, Bacteria and Eukaryota. The next five bars are for *Ureaplasma parvum* serovar, *Wigglesworthia glossinidia*, *Borrelia garinii*, *Buchnera aphidicola*, and *Mycoplasma mycoides*. Arrows indicate amino acid frequency smaller or larger than the mean for these five bacteria.

**Bacterial genomes:** At the time this study was performed there were 31 top-level classifications for protein functions, 229 first-second level classifications (the second level is not unique, but the combination of first and second level is), and 1,078 third level classifications (the subsystems themselves). To investigate whether the amino acid skews in protein composition are dependent on protein function, we calculated KLD for each subsystem first level hierarchy in ten bacterial genomes. Five were chosen from the most extremely skewed organisms (Table 5.1), and five were chosen at random from the remaining genomes. KLD values of the five bacterial species with most skewed amino acid composition greatly differed from the mean for all subsystems, as expected from their overall bias. However, those differences were not restricted to one or a few metabolic process, but were rather consistent across all subsystems tested, so the null hypothesis that the distribution of skewed amino acids is random across the genome and is independent of functional categories could not be rejected. The five control bacterial species chosen at random exhibited
much less variation in amino acid composition.

Figure 5.5: Comparison of KLD and GC-content for all bacterial genomes and for individual groups of subsystems. The GC content of each genome is plotted on the x-axis, and the variation in amino acid composition is shown on the y-axis.
To examine whether the compositional skew of bacterial protein sequences was only restricted to one or a few amino acids, we calculated the frequency of occurrence of each amino acid for the five bacterial genomes that have the most skewed amino acid composition [Figure 5.4]. The null hypothesis was that there would be random changes in the amino acid compositions in these genomes. However, the initial hypothesis was rejected: all five bacterial genomes have significantly reduced their utilization of the amino acids alanine (A), glycine (G), proline (P), and arginine (R), compared to the mean amino acid utilization calculated from all bacteria. This utilization bias appears to have been compensated by an increase in the utilization of the amino acids isoleucine (I), lysine (K) and asparagine (N).

This switch in amino acid utilization has a considerable biological impact because these amino acids are discriminatory in the standard genetic table. A genome consisting entirely of guanosine and cytosine could only encode for alanine (GCC or GCG), glycine (GGC or GGG), proline (CCC or CCG), or arginine (CGC or CGG). In contrast, a genome that contains only adenosine and thymidine could only encode for asparagine (AAT), isoleucine (ATT or ATA), leucine (TTA), lysine (AAA), phenylalanine (TTT), or tyrosine (TAT). Thus, the skew in amino acid composition seems to have been driven by the GC content of the DNA sequence more than the lifestyle, phylogeny, or other characteristics associated with the genome.

The correlation between the percent of the sequences that are either guanosine or cytosine (GC%) and the KLD of the amino acid composition to the mean was calculated [Figure 5.5]. The relationship between GC% and amino acid divergence is given by the equation $y = 2(x - 0.5)^2$, where $x$ is the GC% and $y$ is the divergence of amino acid composition (with a correlation coefficient of 0.84). To test whether the correlation is similar for all areas of metabolism, the relationship between GC% and KLD was calculated for the different subsystems shown in figure 5.5. Most subsystems had similar parabolas suggesting that the
DNA content and amino acid composition were related. However, the relationship did not hold for the secondary metabolism subsystems (the correlation coefficient fell to 0.119, figure 5.5). This suggests that the amino acid profiles of proteins involved in secondary metabolism subsystems are not related to the GC content of the genome. We hypothesize that this may be because genomic subsystems involved in secondary metabolism are more frequently horizontally transferred than those involved in core metabolism, which are usually highly conserved, and we may be observing the skew of the donor organism rather than the current host.

**Phage genomes:** To examine the amino acid utilization behavior for the most skewed phage genomes, we analyzed three GC-rich and three GC-poor phage genomes in more detail. Similar to bacterial genomes, the amino acid composition also seems to be driven by the GC content for the most skewed phages [Figure 5.6]. For example, for amino acids lysine (AAA, AAG) and isoleucine (ATT, ATC, ATA), phage genomes with poor GC content have higher frequency but the GC-rich phage genomes have relatively lower frequency compared to the average amino acid utilization among 835 phage genomes.

![Figure 5.6: Frequency of each of the twenty amino acids for phage genomes. The first three bars are for Spiroplasma kunkelii virus SkV1_Cr2-3x (GC = 22%), Mycoplasma phage phiMFV1 (GC = 25%) and Sulfolobus islandicus rod-shaped virus 1 (GC = 25%). These three genomes are GC poor genomes. The forth bar represents the average frequency of amino acid for 835 phage genomes. The last three bars are for Propionibacterium phage B5 (GC = 64%), Thermus phage P23-77 (GC = 67%) and Streptomyces phage VWB (GC = 71%), which are GC rich genomes.](image-url)
Figure 5.7: Comparison of KLD and GC-content for all phage genomes (A), and for individual groups of subsystems (B-H). The GC content of each genome is plotted on the x-axis, and the variation in amino acid composition is shown on the y-axis. The correlation equation for panel A - complete phage genomes is $y = 1.7x^2 - 1.7x + 0.44$ and the correlation coefficient $R^2 = 0.63$. In panel B, the skews are shown only for those proteins in the subsystems involved in replication and the equation is $y = 2.2x^2 - 2.3x + 0.6$, and the correlation coefficient is 0.3. In panel C, the skews are shown only for those proteins involved in the capsid subsystems. In panel D, the skews are shown only for those proteins in the subsystems involved in phage head. In panel E, the skews are shown only for those proteins in the subsystems involved in phage lysis. In panel F, the skews are shown only for those proteins in the subsystems involved in phage experimental. In panel G, the skews are shown only for those proteins in the subsystems involved in phage tail. In panel H, the skews are shown only for those proteins in the subsystems involved in phage tail fiber.
Like with bacterial genomes, deviation of the amino acid composition (KLD) in phage genomes strongly correlates with the genomes GC% [Figure 5.7]. The relationship is $y = 1.7x^2 - 1.7x + 0.44$, where $x$ is the GC% and $y$ is the KLD (with a correlation coefficient of 0.63). The relationship between KLD and GC-content is statistically different for bacteria and phages ($p$-value $< 10^{-6}$) [details in supplemental materials]. To check whether the variation of amino acid utilization is restricted to one or a few subsystems, KLD was calculated for several phage subsystems in all phage genomes. However, no strong correlation was observed between functional category and GC%, with the exception of the phage replication subsystem (correlation coefficient, $R^2 = 0.3$). This lack of correlation can be explained by the highly diverse nature of phages, which have different mutational and gene transfer dynamics than bacteria.

**GC content and amino acid frequency within genomes:** As the variation of amino acid in a genome (for both bacteria and phages) has a strong correlation with the genome’s GC content, the frequency of amino acid utilization was calculated and plotted against GC% for 446 bacterial genomes and 835 phage genomes (supplemental figure 1 and 2). The correlation between each amino acid and GC% for both phages and bacterial genome follows the similar pattern, although, for phages, there is almost no correlation between the amino acid deviation and GC% for most of the subsystems.

### 5.3 Discussion

A complete genome, unlike random sequences, represents an evolutionary successful set of nucleotides, whose combination encodes a functioning organism that survived selection pressure through the evolutionary times and that is still evolvable [146]. The accrual of complete genome sequences provides an invaluable resource for exploring the different means by which the combination of four nucleotides (A, G, C, and T/U) encodes life
forms able to survive the different environments of our planet. Because the genetic code is relatively simple-yet redundant, studying the information content inherent to complete genome sequences is expected to enable the discovery of various properties of a genomes architecture (e.g., gene order and density, and genome symmetry), compositional bias (e.g., GC content and skews), coding potential (i.e., all possible amino acid combinations it can encode), and codon usage preferences. Such properties can be correlated with functional aspects encoded by the genome and can shed the light on its natural history allowing the study of the organisms evolution [147,148]. Additionally, they can be exploited to develop better strategies of genome interpretation. For example, the information theory, compositional statistics, and genome topography have been extensively used in gene prediction, genome assembly, RNA finding [149, 150], and the prediction of horizontal gene transfer [151–155].

In this study, we explored the possibility of exploiting the coding potential and amino acid distribution biases within complete bacterial and phage genomes for better interpreting metagenomic sequences and predicting which sequence reads within large data sets are likely to encode proteins. To this end, we calculated KLD, a measure of information divergence, for a set of bacterial and phage genomes, and compared the distribution of amino acids in different protein-coding sequences in an attempt to use this metric as a measure of how much those sequences deviate from the standard—the standard being defined by the combined amino acid distribution in all genomes.

We found a significant difference in amino acid utilization between phylogenetic groups of bacteria and phages. In addition, we found that the most skewed amino acid utilization profiles belong to intracellular endosymbiotic or pathogenic bacteria, or to phages that infect them. Whereas amino acid skews did not seem to be restricted to a particular functional subsystem they strongly correlated with the GC content of bacterial genomes. Many studies have shown that the GC content of a genome influences the frequencies of oligonucleotides
and thus amino acid composition of its encoded proteome, which reflect the lifestyle of the organism (e.g., [156–159]).

In this work, we describe how the GC content is driving the divergence of amino acid composition in bacterial genomes away from the mean composition. All five bacterial genomes with the highest amino acid compositional skew have low GC content (ranging from 22% to 28%), and consequently fewer alanine, glycine, proline and arginine residues in their encoded proteins. Their relative inability to encode these amino acids, and their substitution of them with isoleucine, lysine, and asparagine explains the significant skews seen in the protein sequences [Figure 5.3].

Conversely, GC-rich bacteria have fewer codons for phenylalanine, isoleucine, lysine, asparagine, and tyrosine, but compensate with alanine, glycine, proline, and arginine. Therefore, both GC-rich and GC-poor bacteria have the most divergent amino acid compositions, while bacteria with an average GC content have an average amino acid composition [Figure 5.4]. The correlation coefficient ($r^2 = 0.84$) suggests a strong relationship between GC% and KLD. However, as the relationship is not linear, we propose that this relationship gives a better understanding of the correlation between GC content and the variation of amino acid utilization.

The divergence of amino acid composition is not restricted to one or a few functional categories, but is common across all subsystems. For almost all subsystems involved in primary metabolism, the relationship closely follows similar quadretic equations with high correlation coefficients. In contrast, subsystems involved in secondary metabolism appear to have a poor correlation between GC content and amino acid composition. Two possible reasons for this are a high level of horizontal gene transfer in genes within these subsystems, ameliorating the amino acid utilization, or the poorer quality of annotation of secondary metabolism in diverse organisms. Only 167 bacteria have an annotated secondary metabolic subsystem, and most of those have GC content between 40% and 60%.
Some differences were noted in the trends of KLD variation between bacterial and phage genomes. Phages have slightly lower KLD than bacterial genomes, which suggests that bacterial genomes have more homogeneous amino acid frequencies than phage genomes. This probably indicates that bacterial genomes are more conserved than those of phages, which could be the result of strong negative selection pressure exerted on core metabolic and information transfer subsystems in bacteria, as opposed to the lack of core sets of genes among known phages. Moreover, phage population dynamics, their mode of replication, and their rapid turnover result in highly variable, mosaic genomes.

It is worth noting that phages with most skewed amino acid composition are those infecting bacterial endosymbionts or obligate parasites. This observation is consistent with our overall hypothesis that KLD values reflect genome conservation, a phenomenon exaggerated in genomes with limited environment and poor exchange with other sources of DNA. Endosymbionts and intracellular parasites are confined to closed environments, and thus their genomes have the highest variation from the average amino acid distribution. On the other hand, genomes of bacteria (or phages that infect them) living in open environments, which are continuously changing have less variability from the average distribution.

Additional evidence for this correlation between KLD and sequence conservation comes from the observation that only within the phage replication subsystem, the correlation between KLD and GC content is strong: Phage replication genes are among the very few genes that are conserved across most phage types.

Interestingly, *Mycoplasma* species were the only bacterial species with the most skewed KLD values (Table 5.1) and which host phage genomes with the highest KLD skew as well (Table 5.2). In a recent study, where the KLD was calculated for tetranucleotides in bacterial genomes [142], *Mycoplasma sp*. was also considered as the most skewed bacterial genome.
5.4 Methods

5.4.1 Retrieval of Sequence Data

All genomic data, including gene annotations and functional classification, were obtained from the SEED database (http://pubseed.theseed.org [112, 160]). Metagenomic data sets were collected from the MG-RAST server (http://metagenomics.theseed.org [113]). All complete phages were obtained from Phantome database (http://phantome.org/).

5.4.2 Calculation of Kullback-Leibler Divergence

The Kullback-Leibler divergence (KLD) was initially calculated for 372 whole bacterial genomes and 835 complete phage genomes according to the following equation [144].

\[ D_{KL}(P || Q) = \sum_{i} P(i) \log \frac{P(i)}{Q(i)} \]

As used here, \( P(i) \) is the frequency of the \( i \)th amino acid in a given genome \( X \), and \( Q(i) \) is the average frequency of this amino acid calculated from all complete genomes, i.e., all bacterial genomes were used for calculating \( Q(i) \) when the given genome \( X \) is a bacterial genome, all phage genomes were used for calculating \( Q(i) \) when \( X \) is a phage.

The same strategy was followed for the calculation of KLD for a specific subsystem [136]; the subsystem analysis was conducted on subsystems covering 446 bacterial genomes.

The KLD for each phylogenetic class was calculated by the following equation, where \( n \) is the number of genomes in each class.

\[ \frac{1}{n} \sum_{j=1}^{n} \sum_{i} P(i) \log \frac{P(i)}{Q(i)} \]
5.5 Acknowledgement

I thank Dr Barbara A. Bailey for her helpful suggestions on statistical analysis. I also thank Dr Ramy K. Aziz for his valuable comments and suggestions.
Chapter 6

Summary

This dissertation presents several novel techniques that exploit NGS data to address several biological problems in two different fields. First, for efficient data analysis, we have designed a novel sequence aligner called qudaich, which is a powerful tool for aligning the overwhelmingly large volume of NGS datasets. Second, we explored different theories to understand the inherent genomic characteristics in NGS datasets and have developed a novel approach, PhiSpy for predicting prophages in bacterial genomes.

Qudaich introduces a unique way for pairwise local sequence alignment, which outperforms the traditional approaches in this area. Qudaich finds the best (or close to the best) possible alignment for each query sequence by organizing both database and query sequences together. Based on this organization, the optimal alignment can be generated using Smith-Waterman-Gotoh algorithm by identifying the candidate database sequences for each query sequence. Here, we propose two heuristics that reduce the search space while finding the candidate database sequences. In most of the cases, the alignments with the candidate database sequences are the best possible alignment against the whole database. Qudaich not only takes less time to execute, but also finds more useful alignments than contemporary aligners do.

Another key contribution of this dissertation is phiSpy, a novel way for finding prophages
in bacterial genomes. PhiSpy combines homology based and composition based strategy, and is capable of finding the largest number of prophages including novel and unique ones. Unlike other homology based prophage finding applications, phiSpy depends on five distinctive genomic characteristics that can identify prophages without using any sequence similarity with known phage genes. These characteristics are protein length, transcription strand directionality, customized AT and GC skew, and the abundance of unique phage words. Some of these genomic characteristics have been described briefly in previous studies, but phiSpy is the first tool to leverage these characteristics to predict prophages. After predicting the prophages based on these characteristics, phiSpy evaluates the prediction by finding phage insertion points and the similarity of phage proteins. Using this approach, phiSpy successfully reports a total of 1,146 potential prophages in 507 complete bacterial genomes, including 287 putative prophages [134]. PhiSpy was used to predict prophages on a large number of bacterial genomes, using the infrastructure of kbase. In 4,335 bacterial genomes, 12,824 prophages were identified where 3,723 was considered as undefined prophages that have no homology to existing phages. These putative prophages may be novel mobile genetic elements.

This dissertation also presents a detailed study of the information content in genomic data (both DNA and protein level) by applying Shannons information theory and Kullback-Leibler divergence. Our study shows that Shannons index correlates with genomic characteristics and can be effectively applied for finding informative sequences. We demonstrated that Kullback-Leibler divergence can be used to compare the deviation of amino acid utilization. This provides several useful observations. We found that the bacteria with the most skewed amino acid utilization profiles, as well as the bacteria that host phages with the most skewed profiles, are endosymbionts or intracellular pathogens. We discussed that the divergence of amino acid composition is not restricted to certain metabolic processes but is common across all bacterial genomic subsystems. Finally, we showed that the amino
acid utilization profiles strongly correlate with the genomes GC content in bacteria but very weakly correlate with the genomes GC content in phages.
Appendix A

Supplementary Material

A.1 PhiSpy

Supplimental Table 1: List of 41 bacterial genomes, which have manually annotated prophages.

<table>
<thead>
<tr>
<th>Genome Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bacillus halodurans C-125</td>
</tr>
<tr>
<td>Bacillus subtilis subsp. subtilis str. 168</td>
</tr>
<tr>
<td>Bifidobacterium longum NCC2705</td>
</tr>
<tr>
<td>Brucella melitensis 16M</td>
</tr>
<tr>
<td>Caulobacter crescentus CB15</td>
</tr>
<tr>
<td>Clostridium perfringens str. 13</td>
</tr>
<tr>
<td>Clostridium tetani E88</td>
</tr>
<tr>
<td>Deinococcus radiodurans R1</td>
</tr>
<tr>
<td>Escherichia coli CFT073</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
</tr>
<tr>
<td>Escherichia coli O157:H7</td>
</tr>
<tr>
<td>Escherichia coli O157:H7 EDL933</td>
</tr>
<tr>
<td>Haemophilus influenzae Rd KW20</td>
</tr>
<tr>
<td>Lactococcus lactis subsp. lactis II1403</td>
</tr>
<tr>
<td>Listeria innocua Clip11262</td>
</tr>
<tr>
<td>Listeria monocytogenes EGD-e</td>
</tr>
<tr>
<td>Mesorhizobium lot MAFF303099</td>
</tr>
<tr>
<td>Mycobacterium tuberculosis CDC1551</td>
</tr>
<tr>
<td>Mycobacterium tuberculosis H37Rv</td>
</tr>
<tr>
<td>Neisseria meningitidis MC58</td>
</tr>
<tr>
<td>Neisseria meningitidis Z2491</td>
</tr>
<tr>
<td>Pasteurella multocida subsp. multocida str. Pm70</td>
</tr>
<tr>
<td>Pseudomonas aeruginosa PAO1</td>
</tr>
<tr>
<td>Pseudomonas putida KT2440</td>
</tr>
<tr>
<td>Ralstonia solanacearum GMI1000</td>
</tr>
<tr>
<td>Salmonella enterica subsp. enterica serovar Typhi str. CT18</td>
</tr>
<tr>
<td>Shewanella oneidensis MR-1</td>
</tr>
<tr>
<td>Shigella flexneri 2a str. 301</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus Mu50</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus MW2</td>
</tr>
<tr>
<td>Streptococcus agalactiae 2603V/R</td>
</tr>
<tr>
<td>Streptococcus pyogenes M1 GAS</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS315</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS8232</td>
</tr>
<tr>
<td>Streptomyces coelicolor A3(2)</td>
</tr>
<tr>
<td>Vibrio cholerae O1 biovar eltor str. N16961</td>
</tr>
<tr>
<td>Xanthomonas axonopodis pv. citri str. 306</td>
</tr>
<tr>
<td>Xylella fastidiosa 9a5c</td>
</tr>
<tr>
<td>Xylella fastidiosa Temecula1</td>
</tr>
<tr>
<td>Yersinia pestis CO92</td>
</tr>
<tr>
<td>Yersinia pestis KIM</td>
</tr>
</tbody>
</table>
**Supplemental Table 2**: Prophage prediction in 45 complete bacterial genomes, which has a closely related training organism

<table>
<thead>
<tr>
<th>Training Organism</th>
<th>Organism</th>
<th>Known Prophage</th>
<th>Probable Prophage</th>
<th>Undefined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brucella melitensis</td>
<td>Brucella suis 1330</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caulobacter crescentus</td>
<td>Caulobacter sp. K31</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
<td>Escherichia coli ATCC 8739</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
<td>Escherichia coli CFT073</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
<td>Escherichia coli E24377A</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
<td>Escherichia coli O157:H7</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
<td>Escherichia coli W3110</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Listeria innocua</td>
<td>Listeria monocytogenes EGD-e</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mycobacterium tuberculosis H37/Rv</td>
<td>Mycobacterium tuberculosis CDC1551</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mycobacterium tuberculosis H37/Rv</td>
<td>Mycobacterium tuberculosis H37Ra</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pseudomonas putida KT2440</td>
<td>Pseudomonas putida W619</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pseudomonas putida KT2440</td>
<td>Pseudomonas syringae pv. phaseolicola 1448A</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pseudomonas putida KT2440</td>
<td>Pseudomonas syringae pv. tomato str. DC3000</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
<td>Salmonella enterica subsp. enterica serovar Choleraesuis str. SC-B67</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
<td>Salmonella enterica subsp. enterica serovar Typhi str. CT18</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Shewanella oneidensis</td>
<td>Shewanella baltica OS185</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Shewanella oneidensis</td>
<td>Shewanella baltica OS195</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Shewanella oneidensis</td>
<td>Shewanella sp.ANA-3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
<td>Shigella flexneri 2a str. 24577</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
<td>Shigella flexneri 2a str. 301</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus Mu50</td>
<td>Staphylococcus aureus RF122</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus Mu50</td>
<td>Staphylococcus aureus subsp. aureus COL</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus Mu50</td>
<td>Staphylococcus aureus subsp. aureus MRSA252</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus Mu50</td>
<td>Staphylococcus aureus subsp. aureus Mu50</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus Mu50</td>
<td>Staphylococcus aureus subsp. aureus NCTC 8325</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus Mu50</td>
<td>Staphylococcus aureus subsp. aureus USA300</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus Mu50</td>
<td>Staphylococcus epidermidis ATCC 12228</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus Mu50</td>
<td>Staphylococcus haemolyticus JCSC1435</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Staphylococcus aureus subsp. aureus Mu50</td>
<td>Staphylococcus saprophyticus subsp. saprophyticus ATCC 15305</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Xanthomonas axonopodis</td>
<td>Stenotrophomonas maltophilia K279a</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Streptococcus agalactiae 2603V/R</td>
<td>Streptococcus agalactiae NEM316</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS2232</td>
<td>Streptococcus pyogenes MGAS10394</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Streptococcus pyogenes MGAS2232</td>
<td>Streptococcus pyogenes MGAS0232</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Streptomycyes caelicolor</td>
<td>Streptomycyes avermilius MA-4680</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Xanthomonas axonopodis</td>
<td>Xanthomonas campestris pv. campestris ATCC 33913</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Xanthomonas axonopodis</td>
<td>Xanthomonas campestris pv. campestris ATCC 8004</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Xanthomonas axonopodis</td>
<td>Xanthomonas campestris pv. vesicatoria str. 85-10</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Xylella fastidiosa 2a str. 301</td>
<td>Xylella fastidiosa M12</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Xylella fastidiosa 2a str. 301</td>
<td>Xylella fastidiosa Temecula1</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Yersinia pestis CO92</td>
<td>Yersinia enterocolitica 8081</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Yersinia pestis CO92</td>
<td>Yersinia pestis biovar Medievalis str. 91001</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Yersinia pestis CO92</td>
<td>Yersinia pestis KIM</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Yersinia pestis CO92</td>
<td>Yersinia pseudotuberculosis IP 32953</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Yersinia pestis CO92</td>
<td>Yersinia pseudotuberculosis VP3</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Total: 166, 10, 3
Supplemental Figures:

Supplemental Figure 1: Flowchart of \textit{phiSpy}.

1. Input a bacterial genome
2. Calculate five characteristics using a sliding window of \( n \) genes
3. Is there a closely related training genome?
   - Yes: Execute Random Forest using closely related training genome. Consider all five parameters.
   - No: Execute Random Forest using generic training set. Ignore the parameter ‘abundance of phage words’
4. Random forest produces a rank for each sliding window
5. Produces a rank (0 for bacterial gene; 1 for phage like gene) for each gene by taking the average rank of the window in which the gene participated
6. For each predicted prophage, find the \textit{att} sites by identifying a repeated short DNA sequence which has minimum distance from integrase, tRNA/tmRNA
7. Verifying the \textit{att} sites
8. Verifying the predicted prophage region. A region is considered as prophage if
   1. there are \( >5 \) unknown/phage like protein;
   2. \# of phage like/unknown proteins >= \( \frac{3}{5} \) of the total \# proteins in the predicted region
9. Traverse the whole genome. If there is a group of phage like genes, which was not considered in the initial prediction, then consider the region as a potential prophage
10. A list of potential prophages
Supplemental Figure 2: An example of how to calculate the parameter *transcriptional strand orientation*.

Maximum consecutive genes in the same direction

Total # of genes = 17
# of adjacent genes on opposite strands = 8
% of adjacent genes on opposite strands = 8/17 %
% of the maximum consecutive genes in the same direction = 4/17 %
Supplemental Figure 3: Permutation distribution for four different test statistics. The blue line indicates the observed difference of the mean/median of the two distribution of skew. The permutation Achieved Significant Level (ASL<sub>perm</sub>) leads to rejection of the null hypothesis for all four statistics. (A) the distribution for customized AT skew and the observed differences of mean. (B) the distribution for customized AT skew and the observed differences of median. (C) the distribution for customized GC skew and the observed differences of mean. (D) the distribution for customized AT skew and the observed differences of median.
Supplemental Figure 4: Median protein length difference for bacteria (■) and phages (☐). For bacteria, the difference is the median length of all proteins in the genome and the median of all bacterial proteins in the genome. For phages, the difference is the median length of all proteins in the genome and the median of all phage proteins in the genome. The median difference is higher for phage proteins than bacterial proteins.
Supplemental Figure 5: Flowchart of performance analysis.

115
Permutation test for customized AT/GC skew:

For permutation test, two samples (F and G) were created. To create sample F, 190 prophages in 41 complete bacterial genomes were considered. Sample F consists of the absolute difference between the customized AT/GC skew of prophage genes and the customized AT/GC skew of prophages’ flanking genes (same length of corresponding prophage). The size of sample F is 190.

To make the sample G, 800 different bacterial regions were randomly selected from 41 bacterial genomes. The absolute differences of the customized AT/GC skew of these regions and the customized AT/GC skew of the flanking genes of these regions was calculated for sample G. The sample size of G is 800.

Null hypothesis, \( H_0 \): \( F = G \)

To test the null hypothesis we did the permutation test using both the difference of mean (mean of F – mean of G) and the difference of median (median of F – median of G). The test was done as follows:

1. The difference in means/medians between the two samples was calculated, which was the observed value of the test statistic.
2. Sample F and G were combined and randomly divided them into two groups (A and B) of size 190 and 800.
3. The difference in means/medians of group A and B was calculated and recorded.
4. Step 2 and 3 were repeated for 100,000 times.

**Customized AT skew (mean)**

Sampled permutation size, \( s = 100,000 \)

Mean of Sample F = 0.06627
Mean of Sample G = 0.0555

The difference in mean between sample F and G, (say T) = 0.01076
The sampled permutation values where the difference in means is greater than T = 294
P value = 294/100000 = 0.00294 < 0.01
So we can reject the null hypothesis.

**Customized AT skew (median)**

Sampled permutation size, \( s = 100,000 \)

Median of Sample F = 0.0539
Median of Sample G = 0.04408

The difference in median between sample F and G, (say T) = 0.0099
The sampled permutation values where the difference in medians is greater than T = 742
P value = 742/100000 = 0.00742 < 0.01
So we can reject the null hypothesis.

**Customized GC skew (mean)**

Sampled permutation size, \( s = 100,000 \)
Mean of Sample F = 0.05445
Mean of Sample G = 0.04537
The difference in mean between sample F and G, (say T) = 0.00908
The sampled permutation values where the difference in means is greater than T = 830
P value = 830/100000 = 0.00830 < 0.01
So we can reject the null hypothesis.

Customized GC skew (median)
Sampled permutation size, s = 100,000
Median of Sample F = 0.04099
Median of Sample G = 0.03304
The difference in median between sample F and G, (say T) = 0.00794
The sampled permutation values where the difference in medians is greater than T = 1441
P value = 1441/100000 = 0.01441 < 0.05
So we can reject the null hypothesis.
**T-test for the slope of the model of Shannon’s index and the frequency of phage words:**

There are two samples: bacteria and phages.  
The sample size of bacteria, \( m = 401 \)  
The linear model of bacterial sample: \( F = 5.85 \ H + 0.014 \) \( \ldots \) \( (1) \)  
The sample size of phages, \( n = 600 \)  
The linear model of phage sample: \( F = 8.57 \ H + 0.047 \) \( \ldots \) \( (2) \)

We want to test whether the slope of these two equations are significantly different or not.

\( H_0: \beta_b = \beta_p \)  
\( H_A: \beta_b \neq \beta_p \)

where, \( \beta_b \) is the slope of bacterial sample and \( \beta_p \) is the slope of phage sample.

\[ t = \frac{\hat{\beta}_p - \hat{\beta}_b}{\sqrt{SE(\hat{\beta}_p)^2 + SE(\hat{\beta}_b)^2}} \]

where, SE is the standard error.

\[ t = \frac{8.9288 - 5.708284}{\sqrt{0.0224^2 + 0.02371^2}} = 9.87 \]

Degree of freedom = \( m - 2 + n - 2 = 997 \)

In t table, for degree of freedom 1000 and \( p = 0.001 \), the value is 3.3. Our \( t = 9.87 > 3.3 \). So we can reject the null hypothesis.

That means the slope of the bacterial and phage samples are different.
A.2 Kullback Leibler Divergence in Complete Phage and Bacterial Genomes

This appendix contains supporting information for chapter 5.

T-test for testing the relationship between GC content and amino acid variations for bacteria and phage samples:

There are two samples: bacteria and phages.  
The sample size of bacteria, \(m = 372\)  
The linear model of bacterial sample: \(y = 2x^2 - 2x + 0.5\) \(\ldots\) (1)  
The sample size of phages, \(n = 835\)  
The linear model of phage sample: \(y = 1.7x^2 - 1.7x + 0.44\) \(\ldots\) (2)

The general form of these equation is: \(y = \beta_0 + \beta_1 x + \beta_2 x^2\).  
We want to test whether the coefficients (both \(\beta_1\) and \(\beta_2\)) of these two equations are significantly different or not.

\(H_0: \beta_1^1 = \beta_1^2\) and \(\beta_2^1 = \beta_2^2\)  
\(H_A: \beta_1^1 \neq \beta_1^2\) and \(\beta_2^1 \neq \beta_2^2\)

where, \(\beta_1^1\) is the \(\beta_1\) coefficient of equation 1 (bacterial sample) and \(\beta_1^2\) is the \(\beta_1\) coefficient of equation 2 (phage sample).

T test for two independent unequal sample sizes:

\[ t = \frac{\hat{\beta}_p - \hat{\beta}_n}{\sqrt{SE(\hat{\beta}_p)^2 + SE(\hat{\beta}_n)^2}} \]  
where, SE is the standard error.

Here, degree of freedom, \(df = m-2 + n-2 - 372 - 3 + 835 - 3 = 1201\)

For coefficient \(\beta_1\), \(p\) value is \(1.058481e-07\), and for coefficient \(\beta_2\), \(p\) value is \(1.631291e-06\). So we can reject the null hypothesis.

That means equation 1 and equation 2 are significantly different.

R code:

```r
phage = read.table("/Users/sajiaakhter/Documents/2013/KLD_paper/t-test/phage.txt", sep = \'\t\')
bact = read.table("/Users/sajiaakhter/Documents/2013/KLD_paper/t-test/bact.txt", sep = \'\t\')
bfit = lm(V2~I(V1^2)+V1,data = bact)
pfit = lm(V2~I(V1^2)+V1,data = phage)
bsummary = summary(bfit)$coefficients
psummary = summary(pfit)$coefficients
df = bfit$sdf$residual + pfit$sdf$residual # degree of freedom
```

119
\[
\begin{align*}
\text{beta1\_num} &= \text{bsummary}[3,1] - \text{psummary}[3,1] \\
\text{beta1\_dnum} &= \sqrt{\text{bsummary}[3,2]^2 + \text{psummary}[3,2]^2} \\
\text{beta1\_tstat} &= \frac{\text{beta1\_num}}{\text{beta1\_dnum}} \\
\text{beta1\_p\_value} &= 2 \times \text{pt}(-\text{abs} (\text{beta1\_tstat}), \text{df})
\end{align*}
\]

\[
\begin{align*}
\text{beta2\_num} &= \text{bsummary}[2,1] - \text{psummary}[2,1] \\
\text{beta2\_dnum} &= \sqrt{\text{bsummary}[2,2]^2 + \text{psummary}[2,2]^2} \\
\text{beta2\_tstat} &= \frac{\text{beta2\_num}}{\text{beta2\_dnum}} \\
\text{beta2\_p\_value} &= 2 \times \text{pt}(-\text{abs} (\text{beta2\_tstat}), \text{df})
\end{align*}
\]
Supplemental Figure 1: Amino acid frequency of 446 bacterial genomes.
Supplemental Figure 2: Amino acid frequency of 835 complete phage genomes.
Appendix B

Vita

2007 Bachelor of Science in Computer Science
    American International University-Bangladesh, Bangladesh

2013 Doctor of Philosophy in Computational Science
    Claremont Graduate University and San Diego State University

Publications


M Hedin, J Starrett, S Akhter, AL Schnhofer, JW Shultz. Phylogenomic Resolution of Pale-


Bibliography


[10] Heng Li and Nils Homer. A survey of sequence alignment algorithms for next-


[15] Stefan Kurtz, Adam Phillippy, Arthur L. Delcher, Michael Smoot, Martin Shumway,
Corina Antonescu, and Steven L. Salzberg. Versatile and open software for compar-

[16] Arthur L Delcher, Adam Phillippy, Jane Carlton, and Steven L Salzberg. Fast algo-
rithms for large-scale genome alignment and comparison. *Nucleic acids research*,


[18] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows-
2009. PMID: 19451168.

[19] Heng Li and Richard Durbin. Fast and accurate long-read alignment with Bur-


128


[101] Mark W Silby, Ana M Cerdeo-Trraga, Georgios S Vernikos, Stephen R Giddens, Robert W Jackson, Gail M Preston, Xue-Xian Zhang, Christina D Moon, Stefanie M Gehrig, Scott A C Godfrey, Christopher G Knight, Jacob G Malone, Zena Robinson,


[134] Sajia Akhter, Ramy K Aziz, and Robert A Edwards. PhiSpy: a novel algorithm for finding prophages in bacterial genomes that combines similarity- and composition-


[139] Elizabeth A Dinsdale, Robert A Edwards, Dana Hall, Florent Angly, Mya Breitbart, Jennifer M Brulc, Mike Furlan, Christelle Desnues, Matthew Haynes, Linlin Li, Lauren McDaniel, Mary Ann Moran, Karen E Nelson, Christina Nilsson, Robert Olson, John Paul, Beltran Rodriguez Brito, Yijun Ruan, Brandon K Swan, Rick Stevens, David L Valentine, Rebecca Vega Thurber, Linda Wegley, Bryan A White,


