LONG READ MAPPING AT SCALE: ALGORITHMS AND APPLICATIONS

A Dissertation
Presented to
The Academic Faculty

By

Chirag Jain

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Computational Science and Engineering

Georgia Institute of Technology

May 2019

LONG READ MAPPING AT SCALE: ALGORITHMS AND APPLICATIONS

Date Approved: March 15, 2019

ACKNOWLEDGEMENTS

I would like to thank the encouragement and support of my advisor Dr. Srinivas Aluru. I appreciate the freedom granted to me to dive into any area of my interest during the beginning years of Ph.D. Among the numerous things which I learnt through him during my graduate research, I particularly admire his push to strive for maintaining mathematical rigor in the research approaches. I would also like to extend my gratitude to my project mentors Dr. Adam Phillippy, Dr. Kostas Konstantinidis and Dr. Sanchit Misra for providing my thesis with interesting new directions and applications. I owe it to them for providing me sufficient background about the relevant biological concepts and latest technologies which prevented me from making unrealistic assumptions. In addition, the discussions with my dissertation committee members has helped me to improve the presentation of this thesis.

This long journey won't have even started without the love and blessings from my parents, who always ensured that I maintain a healthy work-life balance. I learnt a great deal through my senior colleagues and co-authors Tony Pan, Patrick Flick, Oded Green, Miguel Rodriguez at Georgia Tech, and Alexander Dilthey, Sergey Koren at NIH, especially during the initial years of the graduate program. It was also a pleasure to collaborate with my junior colleagues Haowen Zhang and Yu Gao during the latter half of Ph.D. I am especially grateful to my lab-mates for the open discussions, fun, and friendship. The environment in the lab has always been a reservoir of positive energy, with no barriers on seeking any sort of assistance. Finally, I acknowledge the funding agencies as this research is supported in part by the National Science Foundation grants CCF-1816027, IIS-1416259, and Intramural Research Program of the National Human Genome Research Institute, National Institutes of Health.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Capability to sequence DNA has been around for four decades now, providing ample time to explore its myriad applications and the concomitant development of bioinformatics methods to support them. Nevertheless, disruptive technological changes in sequencing often upend prevailing protocols and characteristics of what can be sequenced, necessitating a new direction of development for bioinformatics algorithms and software. We are now at the cusp of the next revolution in sequencing due to the development of long and ultra-long read sequencing technologies by Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT). Long reads are attractive because they narrow the scale gap between sizes of genomes and sizes of sequenced reads, with the promise of avoiding assembly errors and repeat resolution challenges that plague short read assemblers. However, long reads themselves sport error rates in the vicinity of 10-15%, compared to the high accuracy of short reads ($< 1\%$). There is an urgent need to develop bioinformatics methods to fully realize the potential of long-read sequencers.

Mapping and alignment of reads to a reference is typically the first step in genomics applications. Though long read technologies are still evolving, research efforts in bioinformatics have already produced many alignment-based and alignment-free read mapping algorithms. Yet, much work lays ahead in designing provably efficient algorithms, formally characterizing the quality of results, and developing methods that scale to larger input datasets and growing reference databases. While the current model to represent the reference as a collection of linear genomes is still favored due to its simplicity, mapping to graph-based representations, where the graph encodes genetic variations in a human population also becomes imperative.

This dissertation work is focused on provably good and scalable algorithms for mapping long reads to both linear and graph references. We make the following contributions:

1. We develop fast and approximate algorithms for end-to-end and split mapping of

long reads to reference genomes. Our work is the first to demonstrate scaling to the entire NCBI database, the collection of all curated and non-redundant genomes.

2. We generalize the mapping algorithm to accelerate the related problems of computing pairwise whole-genome comparisons. We shed light on two fundamental biological questions concerning genomic duplications and delineating microbial species boundaries.

3. We provide new complexity results for aligning reads to graphs under Hamming and edit distance models to classify the problem variants for which existence of a polynomial time solution is unlikely. In contrast to prior results that assume alphabets as a function of the problem size, we prove that the problem variants that allow edits in graph remain NP-complete for even constant-sized alphabets, thereby resolving computational complexity of the problem for DNA and protein sequence to graph alignments.

4. Finally, we propose a new parallel algorithm to optimally align long reads to large variation graphs derived from human genomes. It demonstrates near linear scaling on multi-core CPUs, resulting in run-time reduction from multiple days to three hours when aligning a long read set to an MHC human variation graph.

CHAPTER 1
INTRODUCTION AND MOTIVATION

1.1 Background

1.1.1 DNA Sequencing

DNA sequencing refers to the process of determining the order of nucleotides in a DNA molecule, each nucleotide of which is an adenine (A), guanine (G), cytosine (C), or thymine (T). Genome sequencing is the process of determining the entire DNA that constitutes the ‘code’ of a living organism for its growth, function, and reproduction. Ability to sequence DNA has been instrumental in numerous discoveries that underpin biomedical advances, including the sequencing and assembly of the first human genome [1, 2]. Over the past four decades, DNA sequencing technologies have evolved from an expensive, laborious and limited throughput process to a low-cost, automatic, and high throughput process [3]. A classic example is the invention of high-throughput short read sequencing just over a decade ago, exemplified by Illumina sequencers that can now provide close to a trillion base pairs per experiment, at costs so low to make sub $1000 human genome sequencing a reality today. While it is desirable to sequence the complete genome from end to end, current technologies can only sequence many small fragments of DNA (called reads), and also make errors in the form of substitutions, insertions or deletions. Due to the challenges posed by such characteristics, biologists and clinicians rely on fast and accurate bioinformatics software for their analyses to answer a wide range of biological and biomedical questions.

1.1.2 Long Read Sequencing

Emerging single-molecule (or long read) sequencing technologies from Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT) represent a breakthrough in genomics, shifting the sequencing paradigm. These technologies offer reads that are orders of magnitude longer than the Illumina-like short reads. Length of the long reads typically averages above 10 Kbp, and the average can exceed 100 Kbp depending upon the sequencing protocol and the quality of input DNA sample [4]. Long reads are attractive because they narrow the scale gap between sizes of genomes and sizes of sequenced reads, with the promise of avoiding assembly errors and repeat resolution challenges that plague short read assemblers. However, long reads themselves sport error rates in the vicinity of $10 - 15\%$, compared to the high accuracy of short reads ($< 1\%$). Multiple sequencing protocols are available that allow making different trade-offs between read lengths, accuracy and throughput (e.g., circular consensus sequencing (CCS) protocol by PacBio and 1D^2 protocol by ONT). Besides their long lengths, other advantages include (i) single-molecule sequencing without an intermediary amplification step, (ii) ability to sequence DNA in real-time (sequence data from nanopore devices is available just minutes after introducing the sample [5]), and (iii) near random distribution of errors within reads allowing the use of redundancy to uncover true sequence. Finally, superior portability and low cost of MinION sequencer from ONT makes long reads particularly suitable for certain applications (e.g., rapid diagnostics for epidemics [6, 7] and forensic genomics).

The properties of long reads in terms of their length and error characteristics exhibit a stark contrast from the output of its predecessor technologies, thus necessitating new computational approaches. Judging by the progress achieved in short read sequencing, one can expect continued developments in long read sequencing in terms of read lengths, error rates, ease of use and lowering costs, thus facilitating their adoption in a wider range of biological applications.

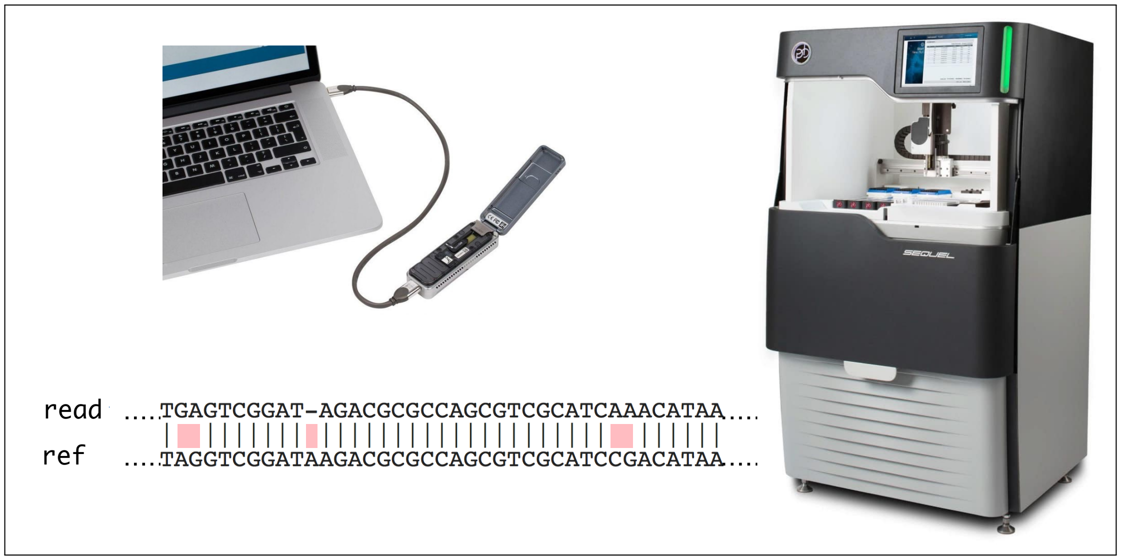

read ....TGAGTCGGAT-AGACGCGCCAGCGTCGCATCAAACATAA....
ref ....TAGGTCGGATAAGACGCGCCAGCGTCGCATCCGACATAA....

Figure 1.1: ONT's MinION (left) and PacBio's Sequel (right) long read sequencing instruments (Image credits: Oxford Nanopore and Pacific Biosciences). Included below is a small snapshot of an E. *coli* nanopore read segment aligned to its reference genome.

1.1.3 Read Mapping and Scalability

Mapping reads onto reference sequences is typically the first computational task during high-throughput sequencing data analyses. From a fundamental mathematical perspective, an exact algorithm for read mapping has long been available. The problem can be solved optimally by designing appropriate variants of the Smith-Waterman alignment algorithm [8]. For each read R and a reference genome G, this takes $O(|R|\cdot|G|)$ time, infeasible in practice when large read sets are mapped to reference genome(s). Hence, mapping algorithms rely on heuristics to identify potential mapping locations, either through seed-and-extend approaches (with exact matches of a fixed length or maximal common substrings as seeds), or through alignment-free approaches. Yet, mapping raw sequences remains a bottleneck for many applications. While the early commercially available long read instruments offered limited throughput, the currently available instruments, e.g., ONT's PromethION, can deliver tera-bases of sequence data in real time. In parallel, reference databases are continually growing in size, with the latest release of non-redundant NCBI RefSeq database already exceeding a tera-base in size.

Desirable characteristics of a read mapper include: (i) efficiency of the algorithm, (ii) avoiding false positive locations, (iii) high sensitivity (i.e., not missing true mapping locations), (iv) scalability to large data sets and reference genome databases, and (v) mathematically characterizing the algorithmic run-time and quality expectations. It is hard to simultaneously achieve all of these objectives; typically, high accuracy comes at the cost of sacrificing run-time efficiency and scalability. Efficient algorithms, combined with nanopore sequencing, could enable real-time genomic analysis of patients, pathogens, cancers, and microbiomes.

1.1.4 Graph-based References

Currently the read mapping problem is commonly perceived with respect to a linear reference genome or a collection of genomes. However, graph-based data structures are heavily relied upon for various bioinformatics applications. Examples include assembly graphs [9, 10] for computing genome assembly, splicing graphs [11] to model splicing variants of genes, partial order graphs [12] to represent a multiple sequence alignment, and more recently, variation graphs [13, 14] for a pan-genome representation. As genome sequencing is becoming ubiquitous, availability of data across multiple individuals and populations is driving the growing importance of graph-based reference representations. This is because graphs provide a natural mechanism for compact representation of related sequences and variations among them. Accordingly, biological applications such as variant calling, genome assembly, read error-correction, and RNA-seq data analysis, all benefit from pattern matching of sequences to a graph-based reference [15, 16, 17, 18].

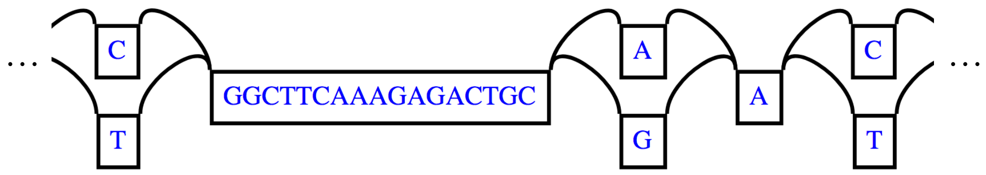


Figure 1.2: Portion of a variation graph built using BRCA1 gene and variant files from 1000 Genomes project [19].

The possibility of an alignment to span through multiple branches in a graph naturally makes read mapping to graph a harder problem than mapping to a linear sequence. Moreover, the nature of the problem changes depending on whether the graphs are cyclic or acyclic [20]. Continued improvements in sequence to sequence alignment algorithms were pivotal to establish it as a fundamental routine for quantifying evolutionary distance in biology [21]. Similarly, fast and accurate sequence to graph aligners are required to fully realize the potential of graph-based reference representations. Typical expectations from an ideal read to graph mapper are similar to the ones described for read to genome(s) mappers, along with the additional requirements of being compatible and robust to real-world genomic graphs, and supporting a clinician-friendly interpretation of the output whenever an alignment spans multiple branches. In contrast to the classic sequence to sequence alignment problem where decades of progress has been made towards designing algorithms and optimized software implementations, research on the sequence to graph alignment problem is still in its infancy.

1.2 Related Works

1.2.1 Long Read Mapping

Read mapping problems can be solved exactly by designing appropriate variants of the Smith-Waterman (SW) alignment algorithm [8]. Although having a guarantee on optimality of the output is attractive, this routine is computationally prohibitive when mapping reads obtained from a high-throughput sequencer to large reference genomes.

Short read sequencing technology has been around for long, and is fairly mature in terms of its read characteristics. Accordingly, efficient mapping algorithms are particularly well studied for it, and popular software such as Burrows-Wheeler Aligner (BWA) [22] and Bowtie [23] are available. Due to high sequencing accuracy (error rate $< 1\%$) in the short reads, it is usually possible to find long maximal exact matches with the reference, which are then used as candidate mapping locations to validate the mapping. However, frequent errors in long reads (error rate 10-15%) make a long exact substring match unlikely. This aspect makes the identification of their true mapping locations a challenging problem, especially when a read is drawn from a repetitive segment of a genome, or a region marked by complex structural variations. Existing tools mitigate this by taking advantage of their long lengths. As such, a common paradigm adopted for long read mapping is to identify the most promising clusters or chains of short exact matches to quickly narrow down the search space. Following is an overview of the heuristics used by existing algorithms.

BLASR [24] and BWA-mem [25] (via option `-x pacbio` or `-x ont2d`) were among the first tools that were leveraged for solving the long read mapping problem, both of which use an FM-index [26] to compute maximal exact matches between the read and the reference. This step is followed by clustering or chaining these matches (or seeds), ranking them by using an appropriate scoring function, and computing a base-to-base alignment of the best scoring clusters. The two tools achieve high accuracy, but are much slower to deal with the high throughput of long read sequencers. One reason for limited scalability is that both the tools identify and evaluate exact matches starting from all the offsets in the read. GraphMap [27] uses spaced-seeds to find the candidate mappings, and obtains better trade-off between computational speed and accuracy.

Even though the reference genome is typically chosen from the same species, it is still from a different organism than the organism which is sequenced by the reads. This factor contributes to additional differences in the reads with respect to the reference genome due to the existence of many single nucleotide and structural variants unique to an individual genome. As lengths of long-read sequences continue to increase, percentage of reads that span structural variants (e.g., long insertions and deletions), and therefore do not map end-to-end to reference, becomes significant. To achieve better mapping accuracy for reads that span such variations, a few tools (e.g., KART [28], NGMLR [29], LAMSA [30]) opt for individually mapping consecutive sub-segments of a read, which are later grouped if they map co-linearly to the reference. NGMLR also uses a convex scoring scheme to effectively differentiate the indels occurring from SVs and sequencing errors.

Alternate similarity metrics such as cosine similarity have been evaluated that better suit high error-rates in long reads. COSINE [31] considers the distributions of short k-mers ($k = 3$–4) to compute the read mappings. Meta-aligner [32] improves mapping accuracy by leveraging reference genome statistics (e.g., repetitiveness) on top of existing mapping algorithms. Typically, the length of exact matches is short in the context of long reads, therefore a few tools (e.g., rHAT [33] and minimap2 [34]) utilize fixed k-mer lookup table for indexing the reference, or a combination of lookup table and an FM-index (e.g., in Kart [28] and lordFAST [35]), to achieve better computational speed. Because memory footprint of indexing all k-mers is much higher than succinct indices, minimap2 [34] uses minimizer-based sampling [36] which substantially speeds up the mapping process while keeping a check on memory usage. Due to its well-engineered chaining score function for long reads, including the fast SIMD-based banded alignment support [37], it is currently the most widely used tool. Further attempts have been made to extend minimap2 for low-memory mobile computing devices for portable sequencing [38], and distributed memory systems to achieve better scalability [39].

One class of algorithms for fast, approximate mapping relies on MinHash-based locality sensitive hashing scheme, originally developed for finding similarities between web documents. Broder [40] proved that an unbiased estimate of the Jaccard similarity coefficient between two sets can be computed efficiently using a subset of hashed elements called a sketch. These ideas have been used to develop alternative mapping for long reads, such as MHAP [41], BALAUR [42] and VATRAM [43].

1.2.2 Whole Genome Comparisons

There exist other problems amenable to solutions developed in the context of long reads, in particular the computation of i) pairwise genome homology map [44], and ii) evolutionary distances [45]. Sequence mapping is also a building block for conducting whole-genome comparisons. While long reads sport high error rates, genome sequences for both prokaryotic and eukaryotic organisms undergo mutations over time. Mapping algorithms designed to tolerate high error rates in long reads can also advance the state-of-the-art for fast alignment-free comparative analyses between genome sequences, while allowing for plenty of divergence due to evolutionary events.

As high-throughput sequencing costs are continuously plunging, scientific efforts are being made to improve our understanding of genetic evolution in humans and other living organisms using whole-genome sequencing and assembly. There are currently many international genome sequencing efforts such as Genomic Encyclopedia of Bacteria and Archaea (GEBA) led by the U.S. Department of Energy Joint Genome Institute [46], 100,000 Genomes Project [47], among others [48, 49, 50, 51]. As a result of ubiquitous sequencing, the number of genomes available in public databases has expanded tremendously.

Homology Detection

Algorithms for inferring homology between DNA sequences have undergone continuous advances for more than three decades, mainly in the direction of achieving better accuracy to compare distant genomes, as well as better compute efficiency to scale with growing data [52, 53]. Similar to read mapping, majority of the existing genome alignment tools use seed-and-extend heuristics. Exact matches are again computed either using a hash table of k-mers [54, 55, 56, 57], or suffix trees and its variants [58, 59, 60, 61, 62]. A third category includes cross-correlation based algorithms [63]. Homology detection is also useful to identify intra- and inter-chromosomal duplications in the human genome that are known to have implications in genome evolution, its stability and diseases [64, 65, 66]. Accordingly, UCSC genome browser also maintains them as a public database, plus a few specialized tools have been developed to compute such duplications [67, 68, 69, 70].

Whole-genome Distance Estimation

Whole-genome sequencing in microbiology is becoming a new norm for taxonomic classification and identification of prokaryotic organisms, replacing prior tedious laboratory techniques [71, 72]. The whole-genome average nucleotide identity (ANI) is a standard similarity metric in microbiology for taxonomic analysis [71, 73, 74, 75, 76]. In the biological literature, ANI is defined as the average alignment identity of all orthologous genes shared between any two genomes [71]. Existing approaches for computing the ANI score require computing sequence alignments to obtain the alignment identities [73, 75, 77, 78]. In this application, computing alignment consumes majority of the runtime. A recent survey of ANI methods [79] reported speedups of up to 4.7x by using Usearch [80] and MUMmer [81], which is also accompanied by lower accuracy among moderately related genomes in the 75-90% ANI range. Ondov *et al.* [82] provide the first proof-of-concept implementation for fast estimation of ANI using MinHash. It is multiple orders of magnitude faster than alignment-based ANI computation, but a straight-forward adoption of the MinHash technique to the problem of computing ANI has limited utility for incomplete draft genomes [83].

1.2.3 Mapping to Graph-based References

When mapping a read to a directed labeled graph, we seek its best matching path in the graph (Figure 1.3). This problem is relevant in the context of many biological applications, a few of which are listed below:

1. *Genotyping:* A graph-based reference serves as a better alternative for highly polymorphic regions of the human genome such as the major histocompatibility complex (MHC) [15]. Recent biological studies have demonstrated superior genotyping accuracy in variant-rich regions of the human genome by replacing the reference sequence with a variation graph, i.e., a reference sequence augmented with known

genetic variations [16, 84, 85, 86]. Similarly, using graph-based reference to index similar antimicrobial resistance genes has been fruitful in improving resistance typing in microbial environments [87].

2. *Long read error correction:* State-of-the-art hybrid long read error correction tools [88, 89, 18] utilize mapping of raw long read sequences to a de Bruijn graph built using Illumina reads. Here the alignment path in the de Bruijn graph is used as a corrected version of the long read.

3. *Genome assembly:* Hybrid genome assembly tools [17, 90, 91] leverage long reads to provide long range information in de Bruijn graphs, and resolve the branches that cannot be resolved by the short reads alone. This is again executed using approximate pattern matching of long reads to graphs. In addition, long read to graph alignments have been leveraged for haplotype-aware genome assembly [92]. On the contrary, non-hybrid genome assembly pipelines [93, 94] use read alignment to partial order graphs [12], which is a commonly used heuristic to generate a consensus sequence.

4. *RNA-seq analysis:* Splicing graphs provide a compact representation of a transcriptome [11]. Long RNA reads can be sufficiently long to capture full-length cDNA copies of RNA molecules. Alignments of reads to such graphs is useful to identify and quantify novel isoforms [95, 96, 97, 98].

In the context of bioinformatics, pattern matching against a non-linear reference is relatively a new phenomenon, but similar problems have existed since 1960s. A related problem of string pattern matching to a regular expression has been studied extensively for lexical analysis in compiler design [99, 100]. Note that regular expressions can be represented as finite-state automata. The biological application of pattern matching sequences against regular expressions was subsequently explored during 1980s, in the context of locating specific patterns (e.g., repeats, activation sites) using an approximate regular expression search in sequence databases [101, 102, 103].

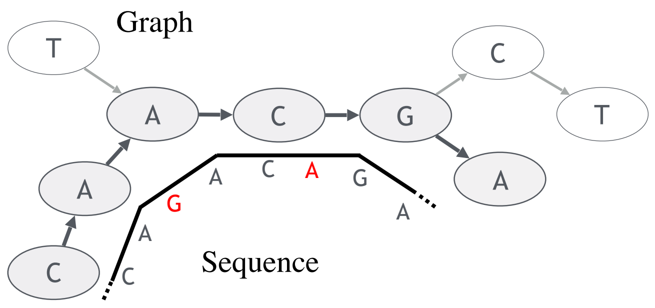


Figure 1.3: An example of an alignment of a sequence along a path in a graph, while accounting for sequence errors (denoted in red).

The formulation of aligning sequences against sequence-labeled directed graphs was studied in a different context about 25 years ago as a search problem in hypertext (synonymous to string labeled graphs) [104], because it was postulated that graphs could represent complex structures (e.g., text documents with hyperlinks). Accordingly, the sequence to graph alignment problem has been studied in string literature in the form of approximate pattern matching to hypertext [104, 105, 106].

For a directed acyclic labeled graph (DAG), existence of a topological order implies that the classic dynamic-programming based alignment algorithms for sequence to sequence alignment can be easily extended. It yields an $O(m(|V| + |E|))$ time algorithm, where m denotes the sequence length, and $G(V, E)$ denotes a character labeled directed graph, respectively [20]. The alignment routine to DAGs is often referred to as partial order alignment in bioinformatics [12].

The problem becomes challenging for general graphs. Amir *et al.* [107] make an interesting observation in this context. An alignment also specifies the set of changes to match the two sequences. Unlike the traditional sequence-to-sequence alignment, the problem variants that permit changes on read or graph or both represent distinct problems. This asymmetry occurs because the input graph can contain cycles, therefore, it is possible that a change in a graph label affects an alignment multiple times, whereas a change in the sequence only affects the alignment at a single position. Amir *et al.* prove that solving the alignment problem while allowing changes on the graph is an $\mathcal{NP}$-complete problem, via proofs that assume alphabets of size $\geq |V|$. In contrast, the problem variant that allows changes to the query sequence alone is polynomially solvable [20, 107]. The best known sequential algorithm to compute optimal alignment score in either acyclic or cyclic graphs requires $O(|V| + m|E|)$ time and $O(|V|)$ space [108, 109].

We next survey the existing tools for long read alignment to variation graphs, a requirement for the genotyping application, using long reads and graph-based references. Several seed-and-extend based heuristic algorithms that have been recently developed for solving the alignment problem to variation graphs [16, 110, 111, 112, 113, 114] have mainly focused on short read mapping. Typically, such algorithms employ an index-based approach to quickly narrow down the search space during the alignment process. In particular, substrings that span all possible alternatives in the graph are indexed using classic string data structures (e.g., FM-index) or hash tables [115, 116]. Due to the exponentially growing potential number of paths as a function of the number of variants, existing heuristics do not translate into a practical solution for dense variant-rich graph regions or aligning long reads [117]. In fact, how to efficiently seed and execute chaining heuristics that are typically used for mapping long reads still remains an open problem in the context of graphs.

A nice feature of the exact quadratic time sequence to graph alignment algorithms is that they guarantee optimal output irrespective of graph topology and error-rate in the input reads. Several prior works [114, 115, 118, 119] assume variation graph is a labeled DAG. Due to the quadratic time complexity of an exact algorithm however, the sequence to DAG alignment problem becomes compute-intensive on real input data sets. The variation graphs associated with some of the most diverse regions (e.g., MHC, LRC segments) in the human genome contain vertices and edges in the order of millions. As a consequence, a naive sequential algorithm would require multiple days or months to align high throughput long read sets. There have been a few attempts recently to accelerate the exact alignment procedure. Graphaligner [120] uses bit-level parallelism to compute edit distance between input reads and DAGs. GSSW (part of vg [16]) extends Farrar's SIMD algorithm [121] to

DAGs. The alignment routine can be used as a stand-alone read mapper, or as part of a heuristic-based mapping algorithm.

1.3 Research Objectives

There is an urgent need to develop and improve bioinformatics methods to fully realize the potential of long-read sequencers, and the research community is seized of this effort. Though the technologies are still in their nascent stages, research efforts have already produced several tools for solving the mapping problems. Yet, simultaneously ensuring practical utility while maintaining theoretical guarantees on output quality remains open.

Exact alignment algorithms that have been known for long do not directly scale to data sets of biological interest. To address this challenge, we develop *new provably good and fast algorithms for mapping long reads*. This is achieved by designing novel formulations to characterize the read mapping targets and leveraging extensive parallelism available in general purpose processors. We not only bridge the gap between theory and practice, but also deliver better performance than heuristics-based algorithms in many scenarios. In what follows, we summarize our contributions in the context of the reviewed literature.

Existing long read to reference genome(s) mappers are based on heuristics, and lack rigorous mathematical characterization of both algorithmic run-times and quality expectations, including in many cases a formal definition of the problem itself. In **Chapter 2**, we develop a new alignment-free algorithm Mashmap for end-to-end mapping of long reads that scales to large reference databases (e.g., Refseq), with sufficient theoretical guarantees on sensitivity and practical validation on the quality of results reported. We formulate the read mapping problem using Jaccard similarity as a proxy, and provide a novel MinHash-based mechanism to solve the mapping problem. In contrast to the existing MinHash-based mapping approaches [41, 42], our Jaccard estimation technique uses MinHash sketch size as a function of read length. This aspect is advantageous because, unlike short-read sequencing, long read technologies can generate highly variable read lengths (e.g., 10^2-10^6

bases). For mapping PacBio reads to the human reference genome, Mashmap is two orders of magnitude faster than BWA-mem [25] and BLASR [24], and delivers competitive performance with respect to state-of-the-art software minimap2 [34].

In **Chapter 3**, we extend the Mashmap algorithm to compute gapped alignments, and refer to it as Mashmap2. Its utility is evaluated in the context of split-read mapping of ultra-long nanopore reads and performing whole-genome comparisons. Even though there are several whole-genome alignment tools, current approaches still remain computationally intensive. For instance, Nucmer [62, 81] and LAST [122], two widely used genome-to-genome aligners, require 10 or more CPU hours to align a human genome assembly to a human reference genome. The alignment-free technique makes Mashmap2 fast and memory-efficient, while providing sensitivity guarantees based on the user-specified minimum alignment length and identity thresholds of the desired local alignments. In practice, Mashmap2 can provide an approximate correspondence between regions of the two input genomes within a few minutes. We demonstrate its utility in identifying all 1 Kbp or longer duplications in the human genome.

In **Chapter 4**, we present the utility of our alignment-free mapping framework to compute whole-genome Average Nucleotide Identity (ANI) among prokaryotic genomes. Existing ANI solvers either rely on time-consuming alignment routines or sacrifice accuracy. Accordingly, it is nearly impossible to calculate ANI values among the available microbial genomes to date, in the order of a hundred thousand, using existing approaches. To remedy this, we develop a scalable ANI solver FastANI, by replacing sequence alignment routine with Mashmap. Our evaluation shows that the proposed method FastANI is up to three orders of magnitude faster than the associated alignment-based methods, without affecting output accuracy. We choose to demonstrate the scalability of FastANI by addressing a fundamental microbiology question of whether prokaryotes form clearly discrete clusters (species), or, a continuum of genetic diversity is observed instead. We provide robust empirical evidence in favor of clear demarcation of species using a large-scale computational analysis of all 90K prokaryotic genomes available in the NCBI database.

The next two chapters address long read mapping to graphs. We provide new complexity results and algorithms that improve fundamental understanding of this problem, including novel ways of solving it. While performing the classic sequence to sequence alignment [8], the decision to allow edits in either one or both of the two sequences does not affect their optimal alignment score. Amir *et al.* [107] show that this symmetry is lost when aligning a read to a directed labeled graph $G(V, E)$. In **Chapter 5**, we prove that the problem variants that allow changes in graphs either standalone or in conjunction with changes in the read are $\mathcal{NP}$-complete under both Hamming and edit distance models for constant-sized alphabets.

In **Chapter 6**, we develop a new practical algorithm for mapping long reads to a variation graph. A sequential $O(m(|V| + |E|))$ time algorithm [20] for sequence alignment to DAGs is known for long, where m denotes the read length, and V and E denote the vertex and edge sets of the graph, respectively. However, the algorithm is too impractical to handle large data sets of biological interest. To resolve this bottleneck, we present the first parallel sequence to graph alignment algorithm PaSGAL that takes full advantage of SIMD-based multi-core architectures. We demonstrate, for the first time, the feasibility of solving the long read to graph mapping problem optimally at the scale of real-world data instances. The results support utility of the algorithm for accurate genotyping using long reads in variation-rich regions of the human genome. **Chapter 7** contains conclusions, and a few interesting open problems for future research.

CHAPTER 2
AN APPROXIMATE ALGORITHM FOR MAPPING LONG READS

In this chapter, we present a new approximate algorithm for mapping long reads that scales to large reference databases. We propose a problem formulation that mathematically characterizes desired mapping targets by linking the Jaccard coefficient between the k-mer spectra of the read and its mapping region to sequence error rate. The problem is solved using an efficient algorithm by estimating the Jaccard coefficient through a combination of MinHash and winnowing techniques. On the quality side, we provide probabilistic bounds on sensitivity. We present techniques for choosing algorithmic parameters as a function of error rate and sequence lengths that guarantees the desired statistical significance, making the algorithm robust to future improvements in the long read sequencing technologies.

While this chapter addresses end-to-end mapping of long reads, its generalization to split-read mapping is discussed later in the next chapter. The proposed algorithm is evaluated using PacBio and ONT data sets, and the scalability is demonstrated by mapping long metagenomic reads to the entire RefSeq database containing 838 Gbp of sequence and $> 60,000$ genomes. We report significant performance gains with respect to BWA-mem [25] and BLASR [24], and much lower memory usage compared to minimap2 [34].

2.1 Preliminaries

Read Error Model: We assume errors occur independently at the read positions, and use a Poisson error model as in previous works [125, 82]. Whilst these assumptions may not reflect the true nature of alignment errors, these are reasonable for designing a practical and provably-good algorithm. Let $\epsilon \in [0, 1]$ be the per-base error rate. The expected number of errors in a k-mer is $k \cdot \epsilon$, and the probability of no errors within each k-mer, assumed

This chapter interpolates material from papers by the author [123, 124] independent, is $e^{-\epsilon k}$. We assume the statement is valid irrespective of error type.

Jaccard Similarity: Assuming $\mathcal{X}, \mathcal{Y}$ are the sets of k-mers in sequences X and Y respectively, their Jaccard similarity [126] is defined as $J(X,Y) = |\mathcal{X} \cap \mathcal{Y}|/|\mathcal{X} \cup \mathcal{Y}|$. The Poisson error model is used to compute the relationship between Jaccard similarity and alignment error rate [82]. We approximate the length of a read alignment to be the read length. Let A be a read derived from B_i, where B_i denotes the length $|A|$ substring of reference B starting at position i. If c and n denote the number of error-free and total k-mers in A, respectively, then the expected value of c/n, termed *k-mer survival probability*, is $e^{-\epsilon k}$. This equation assumes k is large enough such that k-mers in A or B_i are unique. Because $|A| = |B_i|$, $J(A, B_i)$, abbreviated as J, equals $c/(2n - c)$. Using the two equations, we derive the following functions $\mathcal{G}$ and $\mathcal{F}$ to estimate J and ϵ:

$$\mathcal{G}(\epsilon, k) = \frac{1}{2e^{\epsilon k} - 1} \quad \text{and} \quad \mathcal{F}(J, k) = \frac{-1}{k} \times \log\left(\frac{2J}{1+J}\right), \tag{2.1}$$

where $\mathcal{G}(\epsilon, k)$ serves as an estimate of the Jaccard similarity given an error rate, and $\mathcal{F}(J,k)$ estimates the converse. $\mathcal{F}(J,k)$ can be shown as the maximum likelihood estimator (MLE) for error-rate (proved below). Using Jensen's inequality, we also get $\mathbb{E}(J) \geq \mathcal{G}(\epsilon, k)$.

Claim: Given Jaccard similarity J, $\mathcal{F}(J,k) = \frac{-1}{k} \times \log\left(\frac{2J}{1+J}\right)$ is an MLE for error-rate ϵ.

Proof: Denote the probability of a k-mer being error-free as θ, where θ equals $e^{-\epsilon k}$. The count of error-free k-mers follows a binomial distribution with parameters θ, n:

$$P(c\,; \theta, n) = \binom{n}{c} \theta^c (1-\theta)^{(n-c)}$$

Therefore, likelihood function $L(J, n; \theta)$ is given by:

$$L(J, n; \theta) = \binom{n}{c} \theta^c (1-\theta)^{n-c}, \text{ where } c = \frac{2J \cdot n}{1+J}$$

To compute θ_{mle} that maximizes likelihood L, we set $\frac{dL}{d\theta} = 0$, therefore,

$$\begin{aligned}
&\binom{n}{c}\left(c \cdot \theta_{mle}^{c-1}(1-\theta_{mle})^{n-c} - (n-c)\theta_{mle}^{c}(1-\theta_{mle})^{n-c-1}\right) = 0 \\
&\Rightarrow c \cdot (1-\theta_{mle}) - (n-c)\theta_{mle} = 0 \\
&\Rightarrow \theta_{mle} = \frac{c}{n} \\
&\Rightarrow e^{-\epsilon_{mle}k} = \frac{c}{n} \\
&\Rightarrow e^{-\epsilon_{mle}k} = \frac{2J}{1+J} \\
&\Rightarrow \epsilon_{mle} = \frac{-1}{k} \times \log\left(\frac{2J}{1+J}\right)
\end{aligned}$$

MinHash Approximation: The MinHash algorithm is a fast and space-efficient approximation technique to compute an unbiased estimate of Jaccard similarity [40], without explicitly computing the underlying set intersection and union. It relies on a following simple fact: Given a random permutation to the universe of elements, the chance that the smallest items in the two sets under the permutation are same is precisely the Jaccard similarity.

Let s be a fixed parameter. Assuming universe U is the totally ordered set of all possible items, and $\Omega : U \to U$ is a permutation chosen uniformly at random, Broder [40] proved that $P\big(\min_{x\in\mathcal{A}} \Omega(x) = \min_{x\in\mathcal{B}_i} \Omega(x)\big) = J(A, B_i)$, and that

$$|S(\mathcal{A} \cup \mathcal{B}_i) \cap S(\mathcal{A}) \cap S(\mathcal{B}_i)| \;/\; |S(\mathcal{A} \cup \mathcal{B}_i)| \tag{2.2}$$

is an unbiased estimate of $J(A, B_i)$, where $S(\mathcal{A})$ (called the *sketch* of $\mathcal{A}$) is the set of the smallest s hashed items in $\mathcal{A}$, i.e., $S(\mathcal{A}) = \min_s\{\Omega(x) : x \in \mathcal{A}\}$. Typically, the denominator $|S(\mathcal{A} \cup \mathcal{B}_i)|$ is referred as the MinHash sketch size and the numerator as the count of shared sketch elements. This estimate is unbiased provided $S(\mathcal{A})$ is a simple random sample of $\mathcal{A}$. Increasing the sketch size improves the accuracy of the estimate

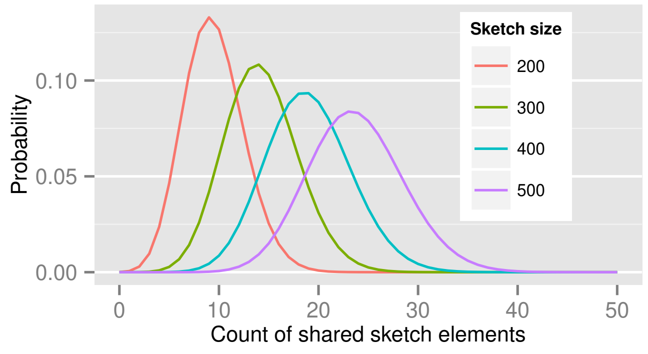


Figure 2.1: Probability distributions of count of shared sketch elements for a read with 15% alignment error ($\epsilon = 0.15$) and k-mer size of 16, with varying sketch sizes. Estimated Jaccard similarity computed using Equation 2.1 is 0.0475.

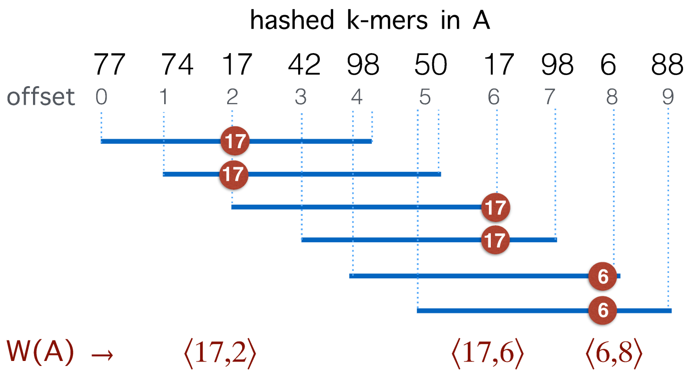


Figure 2.2: Illustration of the winnowing method on a sequence of hashed k-mers in A. $W(A)$ represents the minimizers sampled from the sequence with window size $w = 5$.

because the estimator has an expected error of $\frac{1}{\sqrt{s}}$.

Assuming s is fixed and the true Jaccard similarity $j = J(A, B_i)$ is known, the count of shared sketch elements between $S(\mathcal{A})$ and $S(\mathcal{B}_i)$ follows a hypergeometric distribution. Since s is much smaller than $|\mathcal{A}|$, it can be approximated by the binomial distribution.

$$p\big(\ |S(\mathcal{A} \cup \mathcal{B}_i) \cap S(\mathcal{A}) \cap S(\mathcal{B}_i)| = x | s, j\ \big) = \binom{s}{x} j^x (1-j)^{s-x} \tag{2.3}$$

As an example, Figure 2.1 illustrates this distribution for a read with known Jaccard similarity $j = \mathcal{G}(\epsilon = 0.15, k = 16)$ (using Equation 2.1) and sketch size s varying from 200 to 500.

Winnowing: Winnowing (also known as minimizer sampling [36]) is a local fingerprinting algorithm, proposed to measure similarity between documents by using a subset of hashed words [127]. Unlike MinHash sketching, it bounds the maximum positional gap between any two consecutive selected hashes. It works by sampling the smallest hashed item in every consecutive fixed size sliding window (Figure 2.2). Formal description of this algorithm in the context of genomic sequences follows.

Let A_0 denote the set of all k-mer tuples $\langle k_i, i\rangle$ in sequence A, i denoting the k-mer position. Let w be the window-size used for winnowing, and K_j be the set of w consecutive k-mer tuples starting at position j in A, i.e., $K_j = \{\langle k_i, i\rangle : j \leq i < j+w\}$. Assume Ω is a hash function defined as a random permutation. Then, the set of *minimizers* sampled by the winnowing algorithm in sequence A is $W(A) = \{\min\limits_{\langle k,i\rangle \in K_j} \langle \Omega(k), i\rangle : 0 \leq j \leq |A_0| - w\}$, where

$$\min(\langle k_1, i_1\rangle, \langle k_2, i_2\rangle) = \begin{cases} \langle k_1, i_1\rangle & k_1 < k_2 \text{ or } (k_1 = k_2 \text{ and } i_1 > i_2); \\ \langle k_2, i_2\rangle & \text{otherwise;} \end{cases}$$

Schleimer *et al.* [127] prove that the expected set count of minimizers selected from a random sequence A is $2|A_0|/w$. Moreover, $W(A)$ can be computed efficiently in $O(|A|)$ time and $O(w)$ space using a double-ended queue, as sequence A is read in a streaming fashion [128].

2.2 Problem Formulation

Given a read A and the maximum per base error rate ϵ_{max}, our goal is to identify target positions in reference B where A aligns with $\leq \epsilon_{max}$ per-base error rate. This problem can be exactly solved in $O(|A| \cdot |B|)$ time by designing a suitable quadratic time alignment algorithm. When mapping to a large database of reference sequences, solving this problem exactly is computationally prohibitive. Hence, we define an approximate version of this problem using the Jaccard coefficient as a proxy for the alignment as follows: Let B_i

denote the substring of size $|A|$ in B starting at position i ($0 \leq i \leq |B| - |A|$). For a given k, we seek all mapping positions i in B such that

$$J(A, B_i) \geq \mathcal{G}(\epsilon_{max}, k) - \delta \tag{2.4}$$

Note that if A aligns with B_i with per-base error rate $\leq \epsilon_{max}$, then $\mathbb{E}(J(A, B_i)) \geq \mathcal{G}(\epsilon_{max}, k)$ (using Equation 2.1). As this equation applies only to the expected value of $J(A, B_i)$, we lower this threshold by δ to account for variation in the estimate. The parameter δ is defined as the margin of error in Jaccard estimation using a 90% confidence interval.

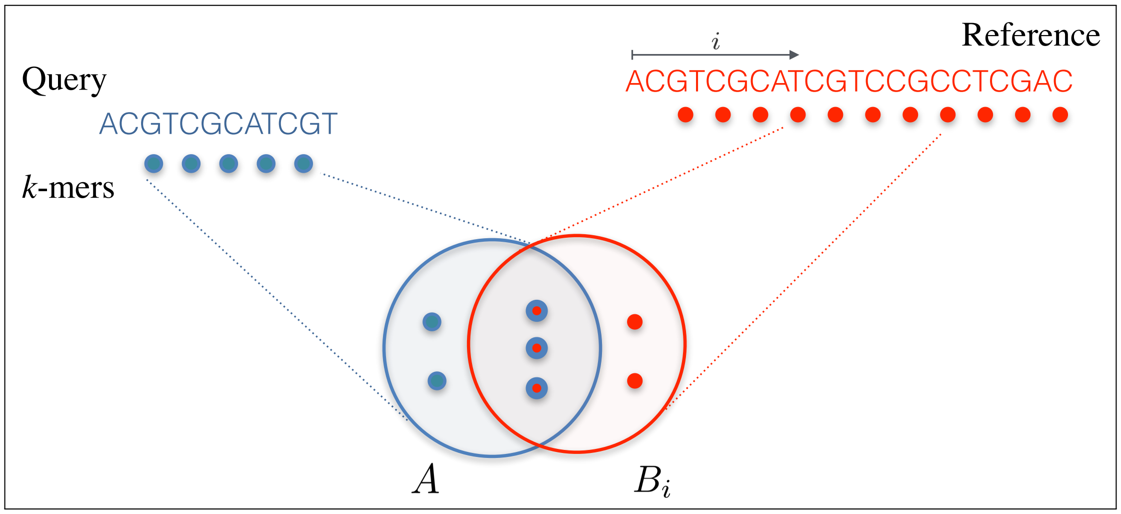


Figure 2.3: Illustrating the use of Jaccard similarity to identify the mapping targets of a long read. Circular dots are used to denote k-mers in the sequences.

2.3 The Proposed Algorithm

Directly computing $J(A, B_i)$ for all positions i is as asymptotically expensive as the alignment algorithm. The rationale for reformulating the problem in terms of Jaccard coefficients is that it enables the design of fast approximate algorithms. We present an algorithm to estimate $J(A, B_i)$ efficiently using a combination of MinHash and winnowing techniques. In addition, we compute an estimate of the alignment error rate ϵ for each mapping reported. Our method relies on an indexing and search strategy we developed to prune the incorrect mapping positions efficiently.

2.3.1 Definitions

Let $W(A)$ be the set of minimizers computed for read A using the winnowing method with window-size w. Assuming s is a fixed parameter, we define $S\big(W(A)\big)$ as the set of the s smallest hashed k-mers that were sampled using winnowing of A, i.e., $S\big(W(A)\big) = \min_s\{h : \langle h, pos\rangle \in W(A)\}$. The set $S\big(W(A)\big)$ is used as a MinHash sketch for the sequence A. To estimate $J(A, B_i)$, we define *winnowed-minhash* estimate $\mathcal{J}(A, B_i)$ for $J(A, B_i)$ as

$$\mathcal{J}(A, B_i) = \frac{|S\big(W(A) \cup W(B_i)\big) \cap S\big(W(A)\big) \cap S\big(W(B_i)\big)|}{|S\big(W(A) \cup W(B_i)\big)|} \tag{2.5}$$

The theoretical properties of the MinHash estimators rely on having a true random permutation which is prohibitive in practice. As such, it is usually implemented using hash functions instead. Our estimator $\mathcal{J}(A, B_i)$, unlike Equation 2.2, uses winnowing to reduce the sampling frame before picking the minimum hash values. We empirically validate in Section 2.7.1 that the quality of the Jaccard estimation using $\mathcal{J}(A, B_i)$ is as good as a typical MinHash implementation. We use $W_h(A)$ to denote the set of hashed k-mers in $W(A)$, i.e., $W_h(A) = \{h : \langle h, pos\rangle \in W(A)\}$.

2.3.2 Indexing the Reference

Retaining the minimizers $W(B_i)$ is sufficient for Jaccard similarity estimation $\mathcal{J}(A, B_i)$ (Equation 2.5). Since $W(B_i) \subseteq W(B)$ (Section 2.1), we compute $W(B)$ from the reference sequence B in order to be able to extract $W(B_i)$ efficiently for any i. The set $W(B)$ can be computed from B in a linear scan in $O(|B|)$ time. We store $W(B)$ as an array $\mathcal{M}$ of tuples $\langle h, pos\rangle$. When created, the set is naturally in ascending sorted order of the positions. Further, to enable $O(1)$ look-up of all the occurrences of a particular minimizer's hashed value h, we also replicate $W(B)$ as a hash table $\mathcal{H}$ with h as the key and an array of its positions $\{pos : \langle h, pos \rangle \in W(B)\}$ as the mapped value. The expected space requirements for $\mathcal{M}$ and $\mathcal{H}$ are $2|B|/w$ (Section 2.1). We postpone our discussion on how to compute an appropriate window-size w to Section 2.4. Besides low memory requirements, a key advantage of this indexing strategy is that a new reference sequence can be incrementally added to the existing data structure in time linear to its length, which is not feasible for suffix array or Burrows-Wheeler transform based indices, typically used in most mapping software.

2.3.3 Searching the Reference

The goal of the search phase is to identify for each read A, positions i such that $J(A, B_i) \geq \mathcal{G}(\epsilon_{max}, k) - \delta$. We instead compute the winnowed-minhash estimate $\mathcal{J}(A, B_i)$. Let $\tau = \mathcal{G}(\epsilon_{max}, k) - \delta$. To avoid directly evaluating $\mathcal{J}(A, B_i)$ for each B_i, we state and prove the following theorem:

Theorem 1. *Assuming sketch size $s \leq |W_h(A)|$,*

$$\mathcal{J}(A, B_i) \geq \tau \;\Rightarrow\; |W_h(A) \cap W_h(B_i)| \geq s \cdot \tau \quad \forall i \; 0 \leq i \leq |B| - |A|.$$

Proof.
$$s \leq |W_h(A)| \implies |S\big(W(A) \cup W(B_i)\big)| = s \tag{6}$$

From Equation 2.5,

$$\begin{aligned}\mathcal{J}(A, B_i) \geq \tau &\implies \frac{|S\big(W(A) \cup W(B_i)\big) \cap S(W(A)) \cap S(W(B_i))|}{|S\big(W(A) \cup W(B_i)\big)|} \geq \tau \\ &\implies \frac{|S\big(W(A) \cup W(B_i)\big) \cap S(W(A)) \cap S(W(B_i))|}{s} \geq \tau \quad \text{(using Equation 6)}\end{aligned}$$

Note that $S\big(W(A) \cup W(B_i)\big) \subseteq S(W(A)) \cup S(W(B_i))$. Therefore,

$$\frac{|\big(S(W(A)) \cup S(W(B_i))\big) \cap S(W(A)) \cap S(W(B_i))|}{s} \geq \tau$$
$$\implies |S(W(A)) \cap S(W(B_i))| \geq s \cdot \tau$$

But, $S(W(A)) \subseteq W_h(A)$ and $S(W(B_i)) \subseteq W_h(B_i)$

Therefore, $|W_h(A) \cap W_h(B_i)| \geq s \cdot \tau$

$\square$

We use the above condition as a filter and only consider positions in B which satisfy $|W_h(A) \cap W_h(B_i)| \geq s \cdot \tau$. To maximize effectiveness of the filter, we set the sketch size $s = |W_h(A)|$. The search proceeds in two successive stages. The first stage identifies candidate positions i using Theorem 1, and the second stage computes $\mathcal{J}(A, B_i)$ at each candidate position i. The position is retained as output if $\mathcal{J}(A, B_i) \geq \tau$, and discarded otherwise.

Stage 1: Algorithm 1 outlines the first stage of our mapping procedure. It calculates all offset positions i in B such that $|W_h(A) \cap W_h(B_i)| \geq \lceil s \cdot \tau \rceil = m$. The output list T is created in the form of one or more tuple ranges $\langle x, y \rangle$, implying that the criterion holds true for all B_i, $x \leq i \leq y$. We begin by computing the minimizer hashed values $W_h(A)$ by winnowing the read A, and compute the positions of their occurrence in the reference (line 4). Accordingly, $L = \{pos : h \in W_h(A) \wedge \langle h, pos \rangle \in W(B)\}$. Next, we sort the array L to process all the positions in ascending order. If B_i satisfies the filtering criterion, there should be at least m entries in L with values between $[i, i + |A|)$. It also implies that m consecutive entries should exist in L with positional difference between the first and m^{th} entry being $< |A|$. This criterion is efficiently evaluated for all B_i using a linear scan on L (lines 6–9). If satisfied, we push the associated candidate range into T. To avoid reporting

B_i more than once, we merge two consecutive overlapping tuple ranges into one.

```
Algorithm 1: Stage 1 of mapping read
Input: read $A$, reference index map $\mathcal{H}$ (hash $k$-mer $\rightarrow pos[]$), s, $\tau$
Output: list $T$ of candidate regions in the reference
1 $m = \lceil s \cdot \tau \rceil$
2 $T = L = []$
3 for $e \in W_h(A)$ do
4     $L$.append($\mathcal{H}(e)$)
5 sort($L$)
6 for $i \leftarrow 0$ to $|L| - m$ do
7     $j \leftarrow i + (m - 1)$
8     if $(L[j] - L[i]) < |A|$ then
9         $T$.append( $\langle L[j] - |A| + 1, L[i] \rangle$ )
```

Stage 2: Evaluation of each tuple $\langle x, y \rangle$ in the Stage 1 output array T requires computing $\mathcal{J}(A, B_i)\ \forall i, x \leq i \leq y$. Accordingly, we compute the minimum s unique sketch elements within $W_h(A) \cup W_h(B_i)$, and count the ones shared between A and B_i. We show the step-by-step procedure in Algorithm 2. We use $\mathcal{L}$ to contain the minimizer hashed values $\{h \in W_h(A) \cup W_h(B_i)\}$. To implement $\mathcal{L}$, we make use of the C++ ordered map data structure that supports logarithmic time insertion, deletion and linear time iteration over unique ordered keys. We keep the hashed value as the map's key, and map it to 1 if it appears in both the reference and the read, and 0 otherwise. For each tuple $\langle x, y \rangle$, we begin by saving the hashed values $W_h(A)$ in read A into map $\mathcal{L}$ (lines 1, 3). Two loops (lines 2, 7) evaluate each tuple $\langle x, y \rangle$ in T, and consider each $B_i,\ x \leq i \leq y$ for Jaccard estimation $\mathcal{J}(A, B_i)$. The function `getMinimizers` gathers the reference minimizer hashes $W_h(B_i)$ by sequentially iterating over $\mathcal{M}$ in the required position range and populating the minimizers associated with each B_i into the map $\mathcal{L}$ (lines 4,8-9). Note that a few incorrect corner minimizers $\{h : \langle h, pos \rangle \in W(B), i \leq pos \leq i + |A|\} \setminus W_h(B_i)$ can appear in $\mathcal{L}$ that were winnowed from windows overlapping with B_i. However, these can be discarded by recomputing the minimum of the first and last window of B_i. Finally,

Algorithm 2: Stage 2 of mapping read

```
Input: index $\mathcal{M}$, Stage 1 output $T$, $s, \tau$
Output: $\mathcal{P}$
1  $\mathcal{L}_0 = \mathcal{L} = \{\}$, $\mathcal{L}_0$.insert$\big(W_h(A)\big)$
2  for $\langle x, y\rangle \in T$ do
3      $i \leftarrow x, j \leftarrow x + |A|, \mathcal{L} \leftarrow \mathcal{L}_0$
4      $\mathcal{L}$.insert$\big($getMinimizers$(i, j)\big)$
5      if $\mathcal{J} =$ solveJaccard$(\mathcal{L}) \geq \tau$ then
6          $\mathcal{P}$.append$\langle i, \mathcal{J}\rangle$
7      while $i \leq y$ do
8          $\mathcal{L}$.delete$\big($getMinimizers$(i, i+1)\big)$
9          $\mathcal{L}$.insert$\big($getMinimizers$(j, j+1)\big)$
10         if $\mathcal{J} =$ solveJaccard$(\mathcal{L}) \geq \tau$ then
11             $\mathcal{P}$.append$\langle i, \mathcal{J}\rangle$
12         $i \leftarrow i + 1, j \leftarrow j + 1$
13 Function getMinimizers $(p, q)$
14     return $\{h : \langle h, pos\rangle \in W(B), p \leq pos < q\}$
15 Function solveJaccard $(\mathcal{L})$
16     $shared\_sketch = \sum_{k=0}^{s-1} \mathcal{L}[k]$
17     return $\mathcal{J} = shared\_sketch/s$
```

function `solveJaccard` computes $|S\big(W(A) \cup W(B_i)\big) \cap S(W(A)) \cap S(W(B_i))|$ by iterating over s minimum unique sketch elements and counting the ones shared between A and B_i. If $\mathcal{J}(A, B_i) \geq \tau$, then the position i and Jaccard estimate $\mathcal{J}(A, B_i)$ are saved into the output $\mathcal{P}$ as pair $\langle i, \mathcal{J}(A, B_i)\rangle$. The corresponding estimate of the alignment error rate ϵ in this case, computed using Equation 2.1, would be $\mathcal{F}\big(\mathcal{J}(A, B_i), k\big)$.

2.4 Selecting Window and Sketch Sizes

The sketch size for Jaccard similarity estimation is inversely proportional to the window size w (Section 2.3.3). A larger window size improves the runtime and space requirement during the search but also negatively affects the statistical significance and accuracy of our estimate. To achieve the right balance, we analyze the p-value of a mapping location being reported under the null hypothesis that both query and reference sequences are random.

For the subsequent analysis, we will assume the sketch size is s, the count of shared sketch elements is a discrete random variable Z, the k-mer size is k, the alphabet set is Σ, and the read and reference sequence sizes are q and r respectively.

Location i is reported if $\mathcal{J}(A, B_i) \geq \tau$, i.e., at least $\lceil s \cdot \tau \rceil$ sketch elements are shared. Following [82], consider two random sequences of length q with k-mer sets X and Y respectively. The probability of a random k-mer α appearing in X or Y, assuming $q \gg k$, is $P(\alpha \in X) = P(\alpha \in Y) = 1 - (1 - |\Sigma|^{-k})^q$. Therefore, the expected Jaccard similarity $J_{null} = J(X, Y)$ is given by

$$J_{null} = \frac{P(\alpha \in X \cap Y)}{P(\alpha \in X \cup Y)} = \frac{P(\alpha \in X) \cdot P(\alpha \in Y)}{P(\alpha \in X) + P(\alpha \in Y) - P(\alpha \in X) \cdot P(\alpha \in Y)}$$

For sketch size s, the probability that x or more sketch elements are shared is $P(Z \geq x | J_{null}, s) = \sum_{j=x}^{s} \binom{s}{j} (J_{null})^j (1 - J_{null})^{s-j}$. Using this equation, we compute the probability of a random sequence of length q mapping to at least one substring in a random reference sequence of size $r \gg q$ as $1 - \big(1 - P(Z \geq x | J_{null}, s)\big)^r$. For a minimum read length l_0 and $x = \lceil s \cdot \tau \rceil$, we wish to ensure that this probability is kept below a user-specified threshold p_{max}. As reported mapping locations i must satisfy $\mathcal{J}(A, B_i) \geq \tau$ and $q \geq l_0$, a mapping with $\mathcal{J}(A, B_i) = \tau, q = l_0$, in general, will have the highest probability of generating a random match. Therefore, we compute the maximum value of w that satisfies the p_{max} constraint for this instance. Sketch size s is set to $|W_h(A)|$, which from Section 2.3.3 is expected to be $q \cdot 2/w$. Since x, s, and w have a circular dependency, we iteratively solve for w, starting from the maximum value l_0, until the probability of a random mapping is $\leq p_{max}$. Influence of different parameters on window size is shown in Figure 2.4. The window size w increases with increasing p_{max} or l_0, but has an inverse relationship with ϵ_{max}. These plots also highlight that as read length and error rate improve, our algorithm automatically adapts to a larger window size, greatly improving efficiency.

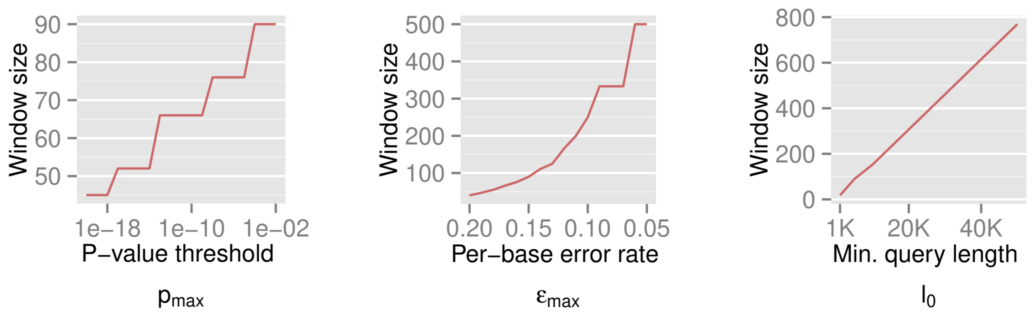


Figure 2.4: Illustration of how w varies with p_{max}, ϵ_{max}, and l_0, respectively. The default values are set as $l_0 = 5000$, $\epsilon_{max} = 0.15$, $p_{max} = 0.001$, $k = 16$, and $r = 10^9$. Steps appear in the first two curves because Z is a discrete variable.

2.5 Proof of Sensitivity

We analyze the sensitivity exhibited by our algorithm in identifying correct mapping locations as a function of the read alignment error rate. Let i be a correct mapping location for read A. If ϵ_{true} is the true error rate in aligning A with B_i, then $J_{true} \approx \mathcal{G}(\epsilon_{true}, k)$. The algorithm reports this mapping location if the Jaccard estimate $\mathcal{J}(A, B_i) \geq \tau$, i.e., the count of shared sketch elements $Z \geq s \cdot \tau$. The associated probability is given by $P(Z \geq s \cdot \tau |\ J_{true}, s) \approx \sum_{x=\lceil s\cdot\tau \rceil}^{s} \binom{s}{x} (J_{true})^x (1 - J_{true})^{s-x}$. We report the corresponding values in Table 2.1 while varying ϵ_{max} and ϵ_{true} from 0.04 to 0.20 error rate, for two sketch sizes $s = 200$ and 500, respectively. In an ideal scenario, a mapping should be reported only if $\epsilon_{true} \leq \epsilon_{max}$, i.e., a perfect algorithm would have “1” in each of the entries at or above the diagonal, and “0” in all other positions. From the table, it is evident that the algorithm achieves close to ideal sensitivity for alignment error rates up to 20%.

2.6 Other Implementation Details

Optimizing for Variable Read Lengths: In contrast to cyclic short-read sequencing, single-molecule technologies can generate highly variable read lengths (e.g. 10^2–10^6 bases). Previously, we discussed how the window size w is determined using the minimum read length l_0 in Section 2.4. From Figure 2.4, notice that we can further reduce the sampling

Table 2.1: Probability of a mapping location being reported by our algorithm for different values of ϵ_{true} and ϵ_{max}. True mapping locations correspond to $\epsilon_{true} \leq \epsilon_{max}$, i.e., entries at or above the diagonal in the tables. Sketch sizes are set to 200 and 500 for the left and right tables, respectively. The k-mer size k is set to 16.

$\epsilon_{\mathbf{true}}$	$\epsilon_{\mathbf{max}}$ 0.04	0.08	0.12	0.16	0.20
0.04	0.951	1	1	1	1
0.08	0	0.937	1	1	1
0.12	0	0.016	0.925	1	1
0.16	0	0	0.184	0.907	0.997
0.20	0	0	0.003	0.403	0.922

$\epsilon_{\mathbf{true}}$	$\epsilon_{\mathbf{max}}$ 0.04	0.08	0.12	0.16	0.20
0.04	0.939	1	1	1	1
0.08	0	0.949	1	1	1
0.12	0	0	0.937	1	1
0.16	0	0	0.013	0.904	1
0.20	0	0	0	0.104	0.896

rate (i.e. use a larger window size) for reads longer than l_0 while still satisfying the p-value constraint. However, to realize this, the sampling scheme for indexing the reference sequence B needs to be consistent with that of query. We propose the idea of *multilevel winnowing* to further optimize the runtime of our algorithm by choosing custom window size for each input read. Suppose $W_w(B)$ denotes the set of winnowed fingerprints in the reference computed using window size w, then $W_{2w}(B) \subseteq W_w(B)$ [127]. We exploit this property to construct a multilevel reference index with multiple window sizes $\{w, 2w, 4w \dots\}$ recursively. This optimization yields us faster mapping time per base pair for reads longer than l_0 as we independently compute the window size for a given read length $l \geq l_0$, and round it to the closest smaller reference window size $\{w, 2w, 4w \dots\}$. The expected time and space complexity to index the reference using multiple levels is unaffected because the expected size of $W_{2^{x+1}w}(B)$ is half of $W_{2^x w}(B)$ and $W_{2^{x+1}w}(B)$ can be determined in linear time from $W_{2^x w}(B)$.

Strand Prediction: To account for the reads sequenced from the reverse strand relative to the reference genome, we compute and store only canonical k-mers, i.e. the lexicographically smaller of the forward and reverse-complemented k-mer. For each k-mer tuple $\langle k, i\rangle$ in $W(A)$ and $W(B)$, we append a strand bit 1 if the forward k-mer is lexicographically smaller and -1 otherwise. While evaluating the read mappings in Stage 2, we compute the

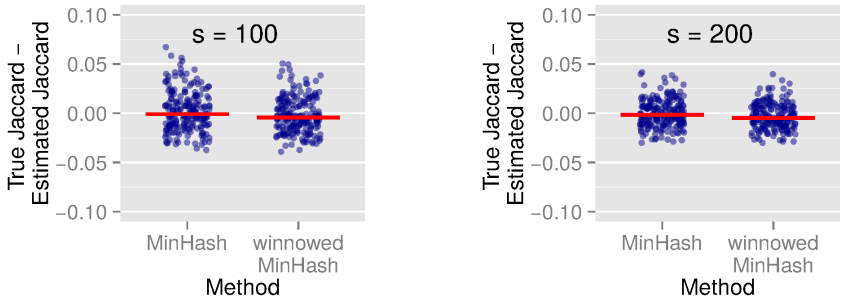


Figure 2.5: Jaccard similarity estimation using MinHash and *winnowed-minhash* estimator $\mathcal{J}(A, B_i)$ over simulated reads, with sketch sizes $s = 100$ and $s = 200$. Red bar indicates the average estimation difference over all reads.

mapping strand of the read through a consensus vote among the shared sketches using sum of pairwise products of the strand bits.

2.7 Experimental Results

2.7.1 Quality of Jaccard Estimation

We first show that the accuracy of the *winnowed-minhash* estimator $\mathcal{J}$ to estimate the Jaccard similarity is as good as the direct MinHash estimation using a hash function. We construct a random sequence of length 5 kbp with each character having equal probability of being either A,C,G or T. We generate reads while introducing substitution errors at each position with probability 0.15. Note that both substitutions and indels have a similar effect of altering the k-mers containing them, and a uniform distribution of errors alters more k-mers than a clustering of errors. Figure 2.5 shows the estimation difference against the true Jaccard similarity using MinHash and our estimator for two different sketch sizes $s = 100$ and $s = 200$. Based on these results, we conclude that the bias in our estimation is practically negligible as the mean error by our method in estimating Jaccard similarity is < 0.003 for both sketch sizes. As expected, the magnitude of estimation error reduces with increasing sketch size.

2.7.2 Mapping MinION and PacBio Reads

We refer the C++ implementation of our algorithm as Mashmap and compare its runtime performance and memory usage against alignment based long-read mappers BWA-MEM v0.7.15-r114 [25], BLASR vSMRTportal 2.3.0 [24], minimap v0.2 [129], and minimap2 v2.15 [34]. We also perform a comparison of the approximate mapping targets generated by Mashmap, minimap and minimap2. Like Mashmap, both minimap and minimap2 use winnowing to index the reference, but do not use the MinHash approximation to estimate Jaccard similarity or nucleotide identity. Instead, minimap seeks clusters of minimizer matches to identify regions of local similarity, whereas minimap2 uses a chaining heuristic to identify co-linear matches. minimap2 provides an option to compute base-to-base alignment after identifying the mapping targets, but we turn off the feature for a fair measurement of its runtime. Importantly, minimap and minimap2 approximate a local alignment process, which is useful for split-read mapping. However, because Mashmap (v1.0) is designed to find complete read mappings, we only consider this case for the following comparisons.

Datasets and Methodology: We evaluated the algorithms by mapping long read datasets generated using single-molecule sequencers from Pacific Biosciences and Oxford Nanopore, and report single-threaded execution timings on an Intel® Haswell CPU (Xeon E5-2698) with 512 GB RAM. We use two datasets, labeled N1 and P1 respectively, both containing reads of length $\geq$ 5 kbp. Dataset N1 is a random sample of 30,000 reads from the MinION (R9/1D) sequencing dataset of the *Escherichia coli* K12 genome [130]. Dataset P1 contains 18,000 reads generated through a single SMRT cell from PacBio's (P6/C4) sequencing of the CHM1 human genome [131]. We map N1 to *E. coli* K12 (4.6 Mbp) and P1 to the human reference (3.2 Gbp). For Mashmap, we use the following parameters: $l_0 = 5000$, $\epsilon_{max} = 0.15$, and $p_{max} = 0.001$. When a read maps to multiple locations, Mashmap only

minimap2 was developed post publication of this work

Table 2.2: Runtime and memory usage comparison of Mashmap against minimap, minimap2, BWA-MEM and BLASR for N1, P1 datasets. BWA-MEM was executed with long read mapping parameters `-x pacbio/ont2d`. Similarly, minimap2 was executed using parameters `-x map-pb/map-ont`.

Method	N1 (MinION-K12)			P1 (Pacbio-CHM1)		
	Index	Map	Memory (MB)	Index	Map	Memory (GB)
Mashmap	0.4s	41s	13	4m 12s	1m 57s	3.7
minimap	0.6s	20s	232	2m 58s	1m 32s	6.8
minimap2	0.6s	29s	459	2m 56s	1m 41s	9.7
BWA-MEM	2.0s	5h 20m	1283	1h 24m	5h 27m	5.8
BLASR	1.3s	10h 17m	697	40m 36s	20h 40m	17.6

reports locations where mapping error rate is no more than 1% above the minimum of error rate over all such locations.

Run-Time Performance: Run-times for the index building and mapping stages, and memory used, for the four methods are compared in Table 2.2. As both BWA-MEM and BLASR are sensitive alignment based methods, we expect their run-times to be significantly higher. Indeed, they take several hours in comparison to seconds (N1) or a few minutes (P1) taken by alignment-free methods. The principal challenge is whether the latter methods can retain the quality obtainable through alignment based methods. We note that Mashmap has the lowest memory footprint for both datasets, and its run-time is similar to minimap and minimap2. The ability to compute the sampling rate at runtime gives Mashmap its edge in terms of memory usage.

Quality of Mapping: As there is no standard benchmark using real datasets, we assess sensitivity/recall using BWA-MEM's starting read mapping positions, and precision by computing Smith-Waterman (SW) alignments of the reported mappings (Table 2.3). Since minimap, minimap2 and BWA-MEM also report split-read alignments, we post-filter their results to only keep alignments with $\geq 80\%$ read coverage. Recall is measured against BWA-MEM alignments which satisfy the $\epsilon_{max} = 0.15$ cutoff ($\geq 85\%$ identity). As min-

Table 2.3: Precision and recall statistics of Mashmap, minimap and minimap2 using datasets N1 and P1.

Id	Recall statistics				Precision statistics		
	Mashmap	minimap	minimap2	#BWA mappings	Mashmap	minimap	minimap2
N1	100.0%	99.87%	99.97%	10,823	94.39%	94.32%	93.89%
P1	96.56%	97.83%	97.81%	10,115	84.59%	30.34%	97.08%

imap, minimap2 and Mashmap estimate mapping positions, the reported mapping is assumed equivalent to BWA-MEM if they overlap, and the strand matches. Precision was directly validated using Smith-Waterman (SW) alignment (with scoring matrix: $match = 1$, $mismatch = -1$, $gap_{open} = -2$, $gap_{extend} = -1$). For the evaluation, we allow SW-identity $\geq 75\%$ and query coverage $\geq 80\%$. Results in Table 2.3 show that the three algorithms have close to ideal sensitivity/recall, demonstrating their ability to uncover the right target locations.

Minimap2 and Mashmap achieve high precision, avoiding too many false positives. The three algorithms consistently achieve about 94% precision, indicating that error rate in a small fraction of ONT reads may be higher than 25%. Minimap's low precision on human is largely driven by false-positive mappings to repetitive sequence. Mashmap false positives here are dominated by reported mappings with a SW query coverage less than 80% of the read length. It may be possible to avoid such mappings by considering the positional distribution of shared sketch elements during the second stage filter, or by adopting a local alignment reporting strategy like minimap2.

We compare our identity estimates $(1 - \epsilon) \times 100$ against the SW alignment identities in Figure 2.6. For the PacBio reads, we observe that most of the points are aligned close to $y = x$. However, for the nanopore reads, our approach overestimates the identity. This is because PacBio sequencing produces mostly random errors, whereas current nanopore errors are more clustered and systematic [132]. Although clustering of sequencing errors distorts the identity estimates, it actually improves the mapping sensitivity because the

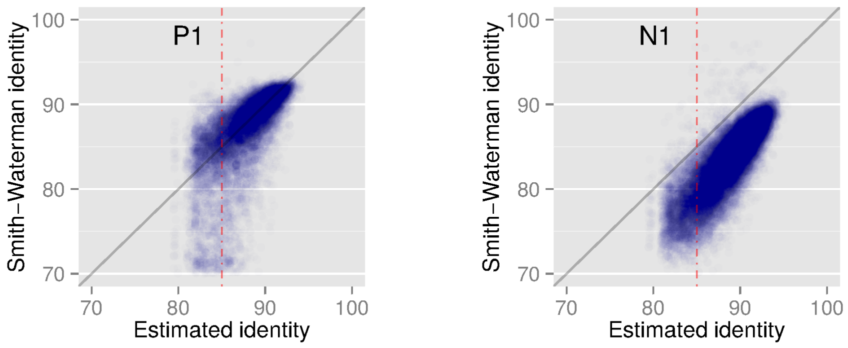


Figure 2.6: Correlation between Smith-Waterman identity and the identity estimated by Mashmap using datasets P1 (PacBio) and N1 (MinION). Red dotted line corresponds to the error cut-off $\epsilon_{max} = 0.15$.

count of minimizer matches increases.

Performance Gain from Future Improvements in Long-read Sequencing Technologies: We discussed how Mashmap adjusts its k-mer sampling rate for estimating the Jaccard similarity based on the provided error-rate (ϵ_{max}) and minimum length (l_0) cutoffs in Section 2.4. Here we show that improvement in read lengths and sequencing error-rate can boost the performance of Mashmap without affecting its output accuracy. For dataset P1, memory usage by Mashmap drops significantly with decreasing per-base error rate threshold ϵ_{max} from 0.20 to 0.10 (Figure 2.7a), or increasing minimum length threshold l_0 from 5 kbp to 30 kbp (Figure 2.7b). Note that the reduced k-mer sampling rate from reference and query sequences would also translate to faster mapping time. All this is achieved while maintaining high recall scores (>90%) against the BWA-mem mappings that satisfy the input thresholds (Figures 2.7c, 2.7d).

2.7.3 Mapping to RefSeq

We perform mapping of a publicly available PacBio read set consisting of 127,565 reads (each $\geq$ 5 kbp) sequenced from a mock microbial community containing 20 strains [133]. To demonstrate the scalability of our algorithm, we map these reads against the complete

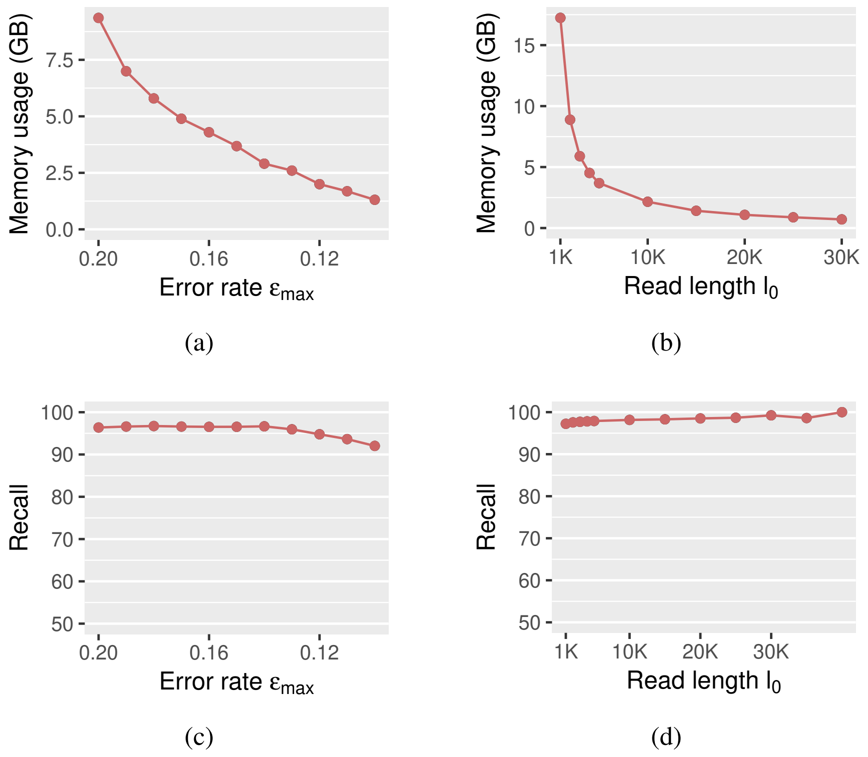


Figure 2.7: **(a)** Drop in memory-usage of Mashmap with varying values of maximum per-base error rate threshold ϵ_{max} from 0.20 to 0.10. Here l_0 is fixed to 5 kbp. **(b)** Drop in memory-usage of Mashmap with varying values of read length cutoff l_0 from 5 kbp to 40 kbp. Here ϵ_{max} is fixed to 0.15. **(c)** Recall scores against BWA-mem mappings which satisfy input cutoffs with varying ϵ_{max}. These values are consistently above 90%. Note that recall score is relatively higher at $\epsilon_{max} = 0.20$ because a significant fraction of PacBio reads in dataset P1 have error rates much less than 20%. With decreasing ϵ_{max} threshold, fraction of borderline cases increases. **(d)** Recall scores against BWA-mem mappings with varying l_0 parameter. These scores are consistently above 97%.

NCBI RefSeq database (838 Gbp) containing sequences from 60,892 organisms. This experiment was executed using default parameters ($l_0 = 5000, \epsilon_{max} = 0.15, p_{max} = 0.001$) on an Intel Xeon CPU E7-8837 with 1 TB memory. The other software could not index the entire RefSeq database at once with the memory limitation. Mashmap took 29 CPU hours to index the reference and 16 CPU hours for mapping, with a peak memory usage of 660 GB. Note that the same index can be repeatedly used for mapping sequences, conferring our method the ability to process data in real-time. To check the accuracy of our results, we ran BWA-MEM against the 20 known genomes of the mock community. The recall of Mashmap against BWA-MEM mappings ranged from 97.7% to 99.1% for all the 20 genomes in the mock community.

2.8 Summary

We have presented a fast approximate algorithm for mapping long reads to large reference genomes. Instead of reporting base-level alignments, Mashmap reports all reference intervals with sufficient Jaccard similarity compared to the k-mer spectrum of the read. In contrast to earlier techniques based on MinHash and winnowing, we provide a formal characterization of the mappings the algorithm is intended to uncover, and provide a provably good algorithm for computing them. Mashmap is designed such that it automatically adapts to different minimum read length and error-rate thresholds, and provides both positional and identity estimates for each mapping reported. As a result, our algorithm has been integrated as a mapping engine to address a wider set of biological applications, including metagenomics long read classification [134], whole-genome homology prediction [44], phylogenetic distance estimation [45], and detecting segmental duplications in the human genome [68, 64].

CHAPTER 3
GAPPED ALIGNMENT FOR SPLIT-READ MAPPING AND WHOLE-GENOME COMPARISONS

In this chapter, we augment new algorithmic strategies to extend our mapping framework for computing local alignment boundaries, useful for both split-read and whole-genome mapping applications. Given minimum identity and length requirements for local alignments, we formulate the characteristics of homologies we intend to compute. The new algorithm internally makes use of the Mashmap end-to-end read mapping framework by applying it to non-overlapping substrings of the query sequence. We mathematically show that all valid local alignment boundaries, which satisfy the user-specified alignment identity and length thresholds, are reported with high probability. The new problem formulation provides a convenient handle for users to account for the divergence between the query and reference sequences, without having to do any parameter tuning. Further, we formulate a heuristic to prioritize mappings with higher scores. We leverage the classic plane-sweep technique from computational geometry to develop an $O(n \log n)$ algorithm to solve the filtering problem, with n being the count of total mappings.

We demonstrate the practical utility of the algorithm by evaluating accuracy and computational performance using real data instances, which include mapping mammalian genome assemblies and ultra-long nanopore reads to the reference genomes, and sensitive self-alignment analysis of the human genome. We compared its performance against Minimap2 [34] and the widely used alignment-based method Nucmer [81, 62]. Our algorithm operates in about a minute and 4 GB memory, including both indexing and mapping stages, to map human genome assembly to a reference when given minimum alignment identity and length requirements of 95% and 10 Kbp respectively. This makes it one of the most

This chapter interpolates material from a paper by the author [44]

resource-efficient software for genome-to-genome mapping, especially with respect to the memory-usage. This performance is achieved while maintaining output sensitivity percentage in the high 90s. Finally, we leverage the computational efficiency of our algorithm, and perform a sensitive self-alignment of the human genome to compute all duplications of length $\geq$ 1 Kbp and $\geq$ 90% identity. The reported output achieves good recall and covers twice the number of bases than the current UCSC browser's segmental duplication annotation.

3.1 The Mashmap2 Algorithm

We designed Mashmap2 to enable fast computation of homology maps between two sequences or a sequence and itself. It consists of two algorithmic components. The first computes approximate boundaries and alignment scores for all pairs of substrings that exceed a user specified length and identity threshold. The second applies a novel filtering algorithm to optionally weed out redundant, paralogous mappings.

3.1.1 Computing Local Alignment Boundaries

Consider all local mappings of the form $Q[i..j]$ between sequences Q (query) and R (reference) of length l_0 or more, such that $Q[i..j]$ aligns with a substring of R with per-base error-rate $\leq \epsilon_{max}$ and $|j - i + 1| \geq l_0$. Alignment algorithms have quadratic time complexity, therefore an exact evaluation of the local mappings between all possible substring combinations will require at least $\Omega(|Q||R|)$ time. As such, solving this problem exactly is computationally prohibitive for typical sizes of real datasets. Instead of explicitly computing all such structures, we seek at least one seed mapping of length $l_0/2$ along the path of each optimal alignment. Doing so, while maintaining high sensitivity and sufficient specificity will allow computation of the local alignments efficiently using an appropriate alignment algorithm.

In our approach, we leverage our previous alignment-free end-to-end read mapping algorithm, designed for mapping noisy long reads (Chapter 2). This allows us to benefit from its attractive properties including probabilistic guarantees on quality, and algorithmic and space efficiency. We continue to assume the same error model that was used in this work, also restated here. We assume that alignment errors, i.e, substitutions and indels in a valid alignment occur independently and follow a Poisson distribution. We also simplify by assuming that k-mers are independent entities in sequences. For a given per-base error rate threshold ϵ_{max}, the read-mapping algorithm reports all target mapping coordinates and identity estimates of a read in the reference, where it aligns end-to-end with $\leq \epsilon_{max}$ per-base error rate, with high probability. This is achieved by linking Jaccard coefficient between the k-mer spectra of the read and its mapping region to the alignment-error rate, under the assumed error distribution model.

Proposed Algorithm

We first split the query sequence Q into $l_0/2$ sized non-overlapping fragments. If a substring of Q, say Q_{sub}, of length $\geq l_0$ aligns against a substring of R with $\epsilon \leq \epsilon_{max}$ per-base error rate, then following statements hold true:

– There is at least one $l_0/2$ sized query fragment that maps end-to-end along the optimal alignment path. This is because at least $\lfloor (|Q_{sub}| - l_0/2 + 1) / (l_0/2) \rfloor \geq 1$ fragments completely span Q_{sub} (see Figure 3.1).

– Under the assumed error distribution, the expected count of errors in a sub-interval is proportional to its length. Therefore, the above $l_0/2$ sized fragment should map along the optimal alignment path with $\epsilon \cdot l_0/2$ expected errors.

Accordingly, the read mapping routine in Mashmap can be used to map each fragment with ϵ_{max} error-rate threshold. Let p be the probability that a fragment is mapped to the desired target position on the reference, computed as described in Section 2.5. Probability of reporting at least one seed mapping along the optimal alignment is given by

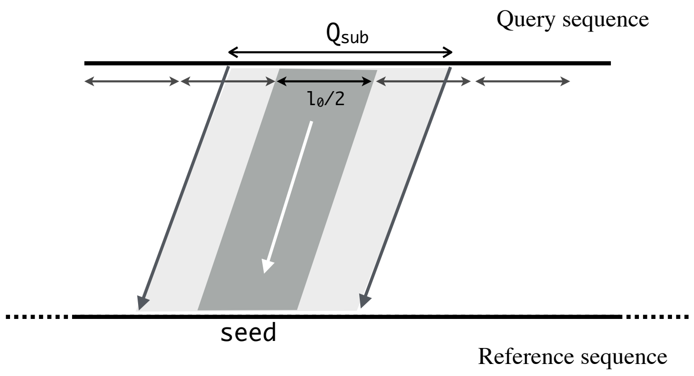


Figure 3.1: A local alignment depicting the inclusion of a length $l_0/2$ fragment of the query sequence.

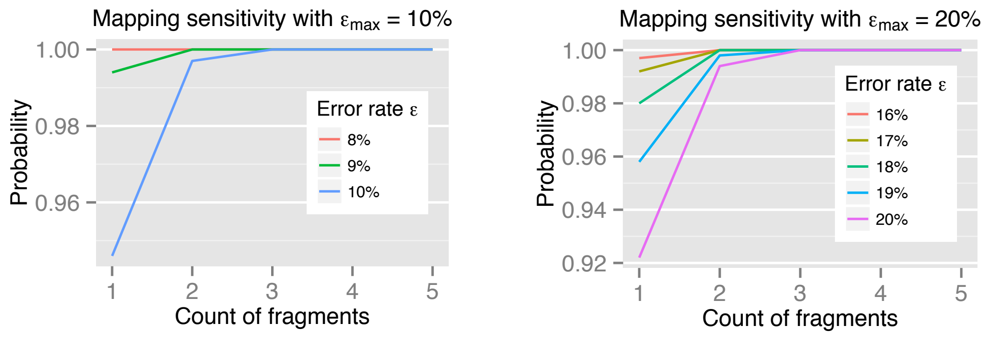


Figure 3.2: Probability of mapping at least one seed fragment for two different error-rate thresholds $\epsilon_{max} = 10\%, 20\%$. As true error rate ϵ decreases, the probability values accordingly improve as expected. Similarly, longer alignments spanning more fragments are more likely to be reported. Most importantly, all the sensitivity scores are consistently above 90%. To compute the probability values, sketch size for Minhash based Jaccard estimation was assumed as 200, and the k-mer size was set to 16. These parameters are internally set by Mashmap (Section 2.4).

$1 - (1 - p)^{\lfloor (|Q_{sub}| - l_0/2\ +1)/(l_0/2) \rfloor}$. We show that these probability scores are sufficiently high, between 0.92 and 1.00 for alignment error rate thresholds ϵ_{max} 10% and 20% respectively (Figure 3.2).

The above seed matches and their alignment identity estimates are further processed to compute approximate local boundaries and their scores. After computing all seed matches, matches which involve consecutive query sequence fragments are merged together if they are mapped closely in the same order on the reference sequence. Suppose mappings from the consecutive query fragments $q_i, q_{i+1}, \ldots, q_j$ are mapped to reference positions with be-

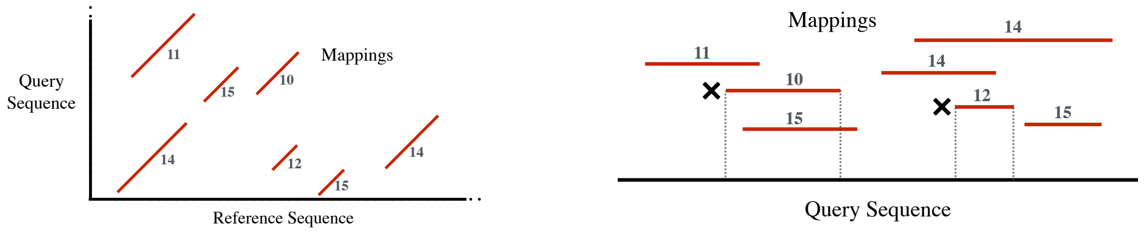


Figure 3.3: Left figure is a toy example to illustrate line segments corresponding to multiple local alignments obtained between query and reference sequence, similar to a dot-plot. Each alignment segment is labeled with an alignment score. Now, suppose we wish to filter best mappings for the query sequence. These segments can be considered as weighted intervals over the query sequence (right figure). In the above case, two intervals marked with a cross are completely subsumed by higher scoring intervals, and therefore, will be labeled as redundant by our filtering heuristic.

gin positions $p_0, p_1, \ldots, p_{j-i}$ respectively, then they are grouped together as a local alignment segment if $p_0 \leq p_1 \leq \ldots \leq p_{j-i}$, and $p_{k+1} - pk \leq l_0$, $[0 \leq k < j - i]$. The alignment boundaries are estimated as the first and last mapping offsets of the group. The corresponding alignment scores are estimated as their average identity estimate multiplied by the sum of the fragment lengths. We use these alignment boundaries and the scores as input to a subsequent filtering algorithm.

3.1.2 A Geometric Algorithm for Filtering Alignments

Large mammalian genomes and plant genomes have abundant repetitive sequences. As a consequence, a large fraction of inferior mappings are reported due to paralogous genomic segments or false positive mappings resulting from simple sequence repeats. Furthermore, from a biological perspective, closely examining all alternative mappings may not be feasible. Therefore, different strategies are adopted to identify biologically relevant outputs. We formulate a filtering heuristic for our mapping application, and develop an optimal $O(n \log n)$ algorithm to solve it. We also prove that $\Omega(n \log n)$ runtime is necessary to solve this problem. The effectiveness of this algorithm on real genomic data is demonstrated later, in the Results section.

Problem Formulation

Suppose all output mappings of a query sequence are laid out as weighted segment intervals, with the alignment scores used as weights (Figure 3.3). We propose the following filtering heuristic: a segment is termed redundant if and only if it is subsumed by higher scoring segments at all of its positions. Therefore, the objective is to identify all *good* (non-redundant) segments. In practice, there can be multiple alignments with equal scores. Therefore, segment scores are allowed to be non-unique.

A sub-optimal $O(n^2)$ algorithm for solving the above problem can be readily developed by doing an all to all comparison among the segments. However, it would lead to practically slow implementation for typical input sizes. The formulated filtering problem bears resemblance to the line segment intersection test problem for which Shamos and Hoey gave a classic $O(n \log n)$ algorithm using plane-sweep technique [135]. Accordingly, we summarize their algorithm next, and subsequently describe the modifications made to solve the filtering problem.

The Shamos-Hoey Algorithm

Similar to the filtering problem, the problem of detecting whether n segments have an intersecting pair has a trivial $O(n^2)$ solution. Shamos and Hoey solved this problem using a plane-sweep based $O(n \log n)$ algorithm. The algorithm defines an ordering between segments in the 2D plane. The main loop of the algorithm conceptually *sweeps* a vertical line from left to right, and while doing so, the sweep-line status data-structure $\mathscr{L}$ dynamically holds segments which intersect the sweep-line. The sweep-line halts at $2n$ endpoints of the input segments, and the order of segments in $\mathscr{L}$ is evaluated to detect any intersection. For efficiency, this algorithm chooses a balanced tree to implement the sweep-line status $\mathscr{L}$. As such, it spends $O(\log n)$ time at each halting point, and therefore, the total runtime is bounded by $O(n \log n)$. This algorithm is popular not only for its theoretical and practical efficiency, but also for ease of implementation.

In our problem as well, evaluating segments which intersect the vertical sweep-line at $2n$ endpoints is sufficient to identify all *good* segments. However, evaluating all intersecting segments at each endpoint is inefficient, and again leads to a quadratic algorithm. Therefore, we devise a new ordering scheme among segments which will enable us to evaluate only a subset of intersecting segments at each endpoint.

Proposed Algorithm for Alignment Filtering

We define an order between segments as following: Between two segments, the segment with higher score is considered as greater, but if the scores are equal, then the segment with the larger starting position is considered as greater. This particular ordering helps avoid redundant computations, and will be crucial for bounding the runtime later.

Similar to the Shamos-Hoey algrithm, we also use a height-balanced Binary Search Tree (BST) as the data-structure for the sweep-line status $\mathscr{L}$, which tracks the segments that intersect the vertical sweep line. $\mathscr{L}$ is required to support the following operations in our algorithm:

1. *insert*(s). Insert segment s into $\mathscr{L}$.

2. *delete*(s). Delete segment s from $\mathscr{L}$.

3. *mark_good*(). Mark all segments with highest score as *good* in $\mathscr{L}$.

Note that the *insert* and *delete* operations are naturally supported in $O(\log n)$ time in BSTs, whereas the *mark_good* function can be realized as a sequence of *maximum* and *predecessor* operations. If there are k segments with equal and highest scores in $\mathscr{L}$, the function *mark_good* uses $O(k \log n)$ time. With the data-structures and the operations defined above, we give an outline of the complete filtering procedure in Algorithm 3. The main loop of the algorithm iterates over the $2n$ segment endpoints, which is analogous to the sweep line moving from left to right, halting at the $2n$ points. In each iteration, we update the sweep-line status $\mathscr{L}$ so that it holds the segments which intersect the sweep line, and mark the highest-scoring segments as *good* using the *mark_good* function.

Lemma 1. *Algorithm 3 solves the filtering problem correctly.*

Proof. Consider a function $S : \mathbb{N} \rightarrow \{0, 1\}^n$ from positions in the query sequence to subsets of segments $\{1, 2, \ldots, n\}$. A segment $s_i \in \{S(pos)\}$ if and only if it is among the highest scoring segments which overlap with the query sequence at position pos. Clearly, a union of all subsets in the domain of function S equals the set of *good* segments. If we perform a linear scan on the domain, from begin to end position of the query sequence, then value of S can change only at the $2n$ endpoints of the segments. Therefore, the highest scoring segments overlapping at the $2n$ endpoints constitute the set of *good* segments, which is precisely what Algorithm 3 computes. □

Algorithm 3: Plane-sweep based alignment filtering algorithm

```
   Input: segments $\{1 \ldots n\}$
1  Sort the $2n$ segment endpoints and place them in the array $\mathscr{E}$
2  Initialize the sweep-line status structure $\mathscr{L}$
3  Initially mark all the segments as redundant
4  for $i \leftarrow 1$ to $2n$ do
5      p = $\mathscr{E}[i]$
6      $set\_beg$ = set of segments of which $p$ is a left endpoint
7      for $s \in set\_beg$ do
8          $\mathscr{L}$.insert($s$)
9      $set\_end$ = set of segments of which $p$ is a right endpoint
10     for $s \in set\_end$ do
11         $\mathscr{L}$.delete($s$)
12     $\mathscr{L}$.mark_good()
13     $i = i + |set\_beg| + |set\_end|$
```

We make an additional modification to the above algorithm for efficiency, specifically in the *mark_good* function. In this function, we mark the highest scoring segments in the tree $\mathscr{L}$ as *good*. We execute this by traversing the segments in decreasing order in $\mathscr{L}$, starting from the maximum. However, we terminate the traversal if a segment is observed as marked *good* already. This helps to avoid redundant computations, and the algorithm still remains correct due to the following property:

Lemma 2. *Consider all the segments with equal and highest scores in $\mathscr{L}$: $s_1, s_2, \ldots, s_j, \ldots, s_k$, ordered in non-increasing manner. Suppose segment s_j has been marked good in one of the previous iterations of the algorithm, then the segments $s_{j+1}, s_{j+2}, \ldots s_k$ must have already been marked good as well.*

Proof. The aforementioned property is satisfied by default during the first iteration of the algorithm because there cannot be any previously marked segments. Suppose this property remains true till iteration i, and we are currently executing iteration $i + 1$. Segments $s_1, s_2, \ldots s_k \in \mathscr{L}$, so we know that the sweep line intersects these segments. Also, the ordering of the segments is maintained based on their scores and begin positions, and since the scores of these segments are equal, therefore $begin\_pos(s_1) \geq begin\_pos(s_2) \geq \ldots \geq begin\_pos(s_k)$. Now consider the iteration when segment s_j was marked *good*. Then, the sweep line must have intersected the segments $s_{j+1}, s_{j+2}, \ldots s_k$ as well. Therefore, if the segment s_j was marked, then the segments $s_{j+1}, s_{j+2}, \ldots s_k$ must have been marked within or before the same iteration. □

The total cost of sorting, *insert* and *delete* operations in Algorithm 3 is clearly $O(n \log n)$. Because the revised *mark_good* function marks at most n segments throughout the algorithm, its runtime is also bounded by $O(n \log n)$. Thus, we conclude that the runtime complexity of our alignment filtering algorithm is bounded by $O(n \log n)$.

Theorem 2. *Given n alignment segments, Algorithm 3 solves the alignment filtering problem in $O(n \log n)$ time.*

Theorem 3. *The above proposed filtering algorithm is optimal given the objective function.*

Proof. The INTEGER ELEMENT UNIQUENESS problem (given n integers, decide whether they are all unique) is known to have a lower bound of $\Omega(n \log n)$ assuming the algebraic decision-tree model [136]. A simple transformation can be designed to show that

$$\text{INTEGER ELEMENT UNIQUENESS} \propto_n \text{ALIGNMENT FILTERING}$$

Let $\{x_1, x_2, \ldots, x_n\}$ be a set of n integer elements. For each element x_i, construct a segment with begin position, end position, and score as x_i, x_i, and i respectively. Because each segment is assigned a unique score, all the n elements are unique if and only if the filtering algorithm reports all the segments as *good*. □

3.1.3 Related Work for Filtering Alignments

There can be many alternative formulations of the filtering criteria. For instance, BLAST [54] filters out alignments if they are fully contained in $\geq K$ alignments of higher scores [137]. Berman *et al.* [137] also discussed a weaker alternative filtering condition where a match is filtered out if each position in a segment is covered by $\geq K$ segments of higher score. Note that our filtering formulation is its special case with $K = 1$. They discussed a different $O(n \log n)$ algorithm to solve the problem based on interval-tree of all input segments. Although a direct performance comparison is not possible due to unavailability of their implementation, note that the tree size in our plane-sweep based algorithm is limited by the number of overlapping segments which intersect the vertical sweep-line, which can be (and typically is) orders of magnitude smaller than the total count for large datasets. As such, even with the same theoretical complexity, we expect our algorithm to perform faster with less memory usage in practice.

3.1.4 Execution for Mapping Applications

The above filtering criteria is useful to identify the promising alignments between query and reference genomes. For the genome-to-genome mapping application, we execute the filtering algorithm twice, once to filter best alignments for query sequence, followed by filtering best alignments for reference sequence. Mappings which pass both filters constitute the orthologous matches, required for building a one-to-one homology map. For read mapping however, filtering on just the query sequence is appropriate. Accordingly, Mashmap2 provides two filtering modes: `one-to-one` and `map` for the two applications respectively.

3.2 Results

We assess the performance of Mashmap2 for genome-to-genome and split-read mapping in comparison to recent versions of state-of-the-art software Minimap2 [34] and Nucmer [62]. Results indicate that Mashmap2 provides output of comparable quality, and yields significant gains in memory-usage. Subsequently, we demonstrate the utility of Mashmap2 in accurately computing all 1 Kbp long duplications in the human genome.

3.2.1 Genome-to-Genome Mapping

Datasets

To evaluate and compare Mashmap2 for mapping genomes, we used six datasets D1-D6 listed in Table 3.1. Dataset D1 includes comparison between microbial genomes E. *coli* O157:H7 and E. *coli* K12. The two instances D2 and D3 require mapping of NA12878 human reference genome assemblies to the hg38 human reference genome. Query genome assemblies in both instances D2 and D3 are the recently published assemblies computed using Canu [138] using ultra-long Oxford Nanopore Technologies (ONT) reads [4]. Dataset D3 includes a long-read only Canu assembly whereas assembly in dataset D2 is also error-corrected using Illumina reads. The next two datasets D4, D5 involve inter-species genome comparisons- human v/s gorilla and chimp v/s gorilla, respectively. Finally, to evaluate Mashmap2 for the split-read mapping task, D6 includes raw ultra-long human ONT reads, generated using a single flowcell [4]. We restrict our benchmarking to real data instances because simulations typically fail to capture the full complexity of mutational processes.

Table 3.1: List of datasets used for evaluation. Datasets D1-D5 are included to evaluate Mashmap2 for genome-to-genome mapping application, and D6 for long read mapping application. We discarded a small fraction of contigs and reads with length $<$10 Kbp.

Id	Query sequences ($\geq$ 10 Kbp)			Reference genome
	Source	# Sequences	N50 (bp)	
D1	E. *coli* O157 genome	2	5.5M	E. *coli* K12 MG1655
D2	human genome assembly (ONT+Illumina)	2,269	7.7M	human (hg38)
D3	human genome assembly (ONT)	2,263	7.4M	human (hg38)
D4	human (hg38) genome	365	145M	gorilla (gorGor5)
D5	chimp (panTro5) genome	3,086	137M	gorilla (gorGor5)
D6	Ultra-long human ONT reads	7,656	129K	human (hg38)

Defining Baseline and Methodology

We used MUMmer package (v4.0.0.beta2), which includes the Nucmer4 alignment program for comparing DNA sequences [62]. Nucmer4 is sensitive enough to report alignments for both assembly and read mapping tasks, therefore we considered its output as truth while evaluating accuracy. In addition, UCSC genome browser [139] hosts high-quality pairwise syntenic alignment sets between popular mammalian genomes. Therefore, for evaluating the inter-species genome comparisons (D4, D5), we could use these as our truth sets. These alignments were originally computed using BLASTZ [56] with careful parameter tuning, and are more reliable for this purpose. We also used Minimap2 (v2.7-r659) [34] as a baseline for various performance metrics. Minimap2 executes chaining algorithm on fixed-length exact matches to compute alignment boundaries. To our knowledge, it is among the fastest tools available to map DNA sequences in an alignment-free fashion.

Each software, including ours exposes many parameters (e.g., k-mer or seed length).

Default k-mer size in Mashmap2 is 16. We mostly conform to default parameters with all software tested, except as noted below. Mashmap2 mainly requires a minimum length and identity for the desired local alignments. In this test, we targeted long alignments, and accordingly fixed the minimum alignment length requirement as 10 Kbp. We set the minimum alignment identity requirement for all the datasets based on their input characteristics as $\{$D1-D2: 95%, D3-D5: 90%, D6: $80\%\}$. Accordingly, we tested Mashmap2 for reporting the alignment boundaries as per the provided requirements. Filtering modes were set to `one-to-one` and `map` for datasets D1-D5 and D6 respectively. Nucmer4 was run with default parameters, followed by running *delta-filter*, both components of the MUMmer package. Following its user documentation, *delta-filter* was executed with `-1` parameter to construct one-to-one alignment map in datasets D1-D5 and `-q` parameter for read mapping in D6. Finally, Minimap2 supports genome-to-genome mapping mode using `-x asm5` flag, and nanopore read mapping mode using `-x map-ont`. We executed all three software in multi-threaded mode using 8 CPU threads. All comparisons were done on an Intel Xeon E5-2680 platform with 28 physical cores and 256 GB RAM.

Runtime and Memory Usage

The wall-clock runtime and memory-usage of Mashmap2, Minimap2 and Nucmer4 using datasets D1-D6 are shown in Table 3.2. The runtimes represent end-to-end time, from reading input sequences to generating the final output. Minimap2 can report base-to-base alignments but does not by default. Thus, the final output of Mashmap2 and Minimap2 are alignment boundaries and scores, whereas Nucmer4 outputs base-to-base alignments. Both alignment-free methods Mashmap2 and Minimap2 map most of the query bases to unique positions in all datasets (shown later), therefore base-to-base alignments can be computed quickly for the final output using chaining heuristics and vectorization techniques [37, 34].

From Table 3.2, we observe that Mashmap2 uses significantly less memory when compared to Minimap2. It improves memory-usage by 5.3x, 4.9x, 4.4x, 3.0x, 3.3x, 1.04x for

Table 3.2: Total execution time and memory usage comparison of Mashmap2 against Minimap2 and alignment-based tool Nucmer4. All software were run in parallel using 8 CPU threads.

Id	Mashmap2		Minimap2		Nucmer4	
	Time	Memory	Time	Memory	Time	Memory
D1	0.5s	16M	0.4s	85M	5.2s	138M
D2	1m 26s	3.5G	3m 3s	17.3G	5h 1m	53G
D3	6m 33s	3.6G	3m 11s	15.9G	2h 10m	53G
D4	27m 33s	9.0G	15m 6s	26.7G	33h 4m	57G
D5	25m 40s	7.7G	5m 54s	25.7G	24h 58m	56G
D6	13m 6s	10.0G	3m 10s	10.4G	25m 2s	53G

the six datasets respectively. The performance gap against Nucmer4 is much wider with speedups of 10.4x, 210x, 19.8x, 72.0x, 58.4x, 1.9x and memory-usage improvements by 8.6x, 15.1x, 14.7x, 6.3x, 7.3x, 5.3x on the datasets D1-D6 respectively. Low memory requirements in Mashmap2 can allow for larger comparisons (e.g. a genome against a big, in-memory reference database).

Mashmap2 and Minimap2 follow the same initial step of sampling k-mers using minimizers [127, 36], followed by computing their exact matches in the reference genome. However, they differ in using these exact matches to compute the mappings. Mashmap2 includes an efficient MinHash-based mechanism to estimate Jaccard similarity and auto-tunes the internal parameters (e.g., k-mer sampling rate, Jaccard similarity threshold), conforming to the local alignment identity and length requirements provided by user. Auto-tuning can help achieve faster runtime and reduce memory-usage with increasing identity and length thresholds (Figure 3.4). It is important to maintain high accuracy while being fast, therefore we next evaluate the quality of output.

Accuracy

Accuracy evaluation of Mashmap2 and Minimap2 in comparison to the assumed truth sets is shown in Table 3.3. As stated before in Section 3.2.1, recall was measured against the assumed true alignments, i.e., Nucmer4 alignments for intra-species comparisons (D1-D3,

Table 3.3: Accuracy evaluation of Mashmap2 and Minimap2 to do an alignment-free computation of mapping boundaries. Recall was measured against the truth sets assumed (Section 3.2.1).

Id	Recall scores			Fraction of query bases mapped uniquely		Precision*
	Mashmap2	Minimap2	#True Alignments	Mashmap2	Minimap2	Mashmap2
D1	100%	100%	144	74.0%	78.9%	72.0%
D2	97.5%	98.3%	35,186	96.8%	96.3%	50.0%
D3	97.1%	98.1%	37,807	96.9%	96.0%	55.2%
D4	97.0%	97.7%	63,908	87.5%	91.3%	75.9%
D5	97.5%	98.0%	65,289	89.8%	93.2%	57.3%
D6	99.3%	99.5%	4,349	89.9%	84.6%	34.8%

*Fraction of mappings which satisfied alignment thresholds in Mashmap2

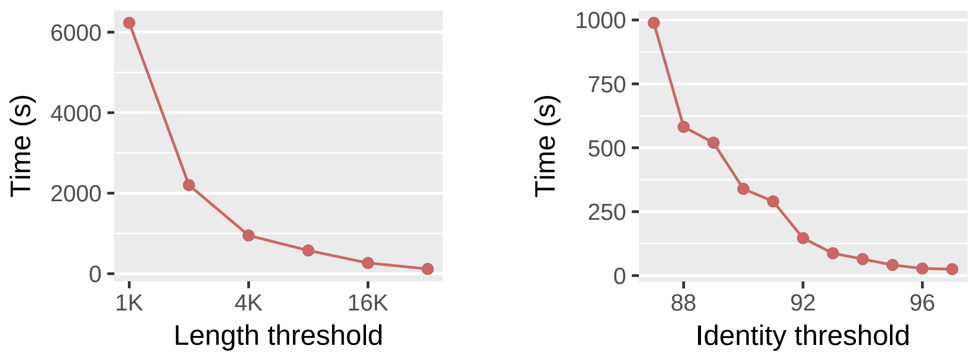


Figure 3.4: Wall time of Mashmap2 decreases with increasing length or identity thresholds using dataset D3 and 8 CPU threads. In this experiment, identity and length thresholds were fixed to 90% and 10 Kbp while varying the other parameter. Memory-usage also follows a similar trend (data not shown).

D6) and UCSC browser pairwise alignments for inter-species comparisons (D4, D5) which satisfy the alignment requirements in terms of minimum length and identity provided to Mashmap2. We also expected Minimap2 to report these alignments because it is designed to compute matches in these identity ranges.

A reported local alignment boundary estimate by Mashmap2 or Minimap2 was assumed to recall a true alignment if it overlapped with the alignment on both query and reference sequences, and if the mapping strand matched. From Table 3.3, we observe that both Mashmap2 and Minimap2 consistently achieved high recall scores $\geq 97\%$, with Minimap2

Table 3.4: Effectiveness of the filtering algorithm in Mashmap2. A large fraction of mappings were filtered out by the algorithm, while the recall scores against the assumed truth sets remained largely unaffected. Last column in this table is copied from Table 3.3 for convenience.

Id	Count of output mappings			Recall scores	
	Without filter	With filter	Ratio (without/with)	Without filter	With filter
D1	145	82	1.77	100.0%	100.0%
D2	6,541,930	3,985	1,642	99.9%	97.5%
D3	53,331,538	3,137	17,001	99.7%	97.1%
D4	152,536,106	4,756	32,072	100.0%	97.0%
D5	152,266,777	13,834	11,007	100.0%	97.5%
D6	18,604,261	12,930	1,439	99.9%	99.3%

performing slightly better. Obtaining high recall scores by itself is not sufficient, because it can be achieved by mapping a query sequence to all possible positions. In parallel to achieving high recall scores, both Mashmap2 and Minimap2 mapped a large fraction of query genome assemblies to unique mapping positions in the reference genomes. To show this, we computed the fraction of base-pairs of the query sequence that are mapped to a single position on the reference genome (Table 3.3).

Next, we evaluated the precision, i.e. what fraction of Mashmap2 mappings yield one or more alignments above the specified length and identity thresholds. We used LAST [122] to compute the alignments. The precision score of Mashmap2 averages to 57.5% across all the datasets, varying from 34.8% (in D6) to 75.9% (in D4). The corresponding scores for Minimap2 using the same threshold values are much lower (average = 15%), but Minimap2 follows different design principles and lacks similar guarantees on the characteristics of its output. In the current context of tasks that require such guarantees, Mashmap2 provides better precision on all datasets.

Efficacy of The Filtering Algorithm

Eukaryotic genomes tend to contain a lot of repetitive sequences, therefore, the motivation behind our plane-sweep based filtering heuristic is to discard noisy mappings, and compute

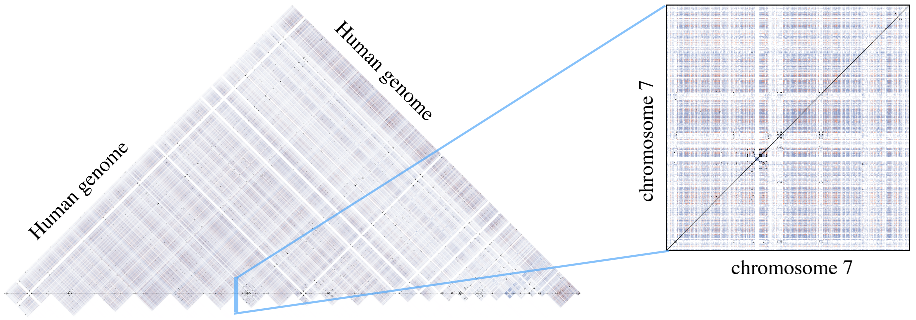


Figure 3.5: Visualization of ≥ 1 Kbp duplications in the human genome computed using Mashmap2. Alignments are colored based on their lengths: blue 1-5 Kbp, red 5-10 Kbp, black >10 Kbp. Majority of blue and red mappings occur due to SINEs and LINEs repeats respectively. Right plot is a magnification of ≥ 1 Kbp duplications within chromosome 7. Chromosome 7 is known to be one of the most duplicated human chromosomes. Large clustered duplications in red circle are associated with Williams-Beuren syndrome [140].

promising matches between the query and reference genomes. We show the importance and effectiveness of our filtering strategy in Table 3.4. Note that a large fraction of mappings was pruned out by the filter. While doing so, high recall scores against the assumed true sets were maintained (see Table 3.4). Although we do not present the contribution of this phase to the total runtime, the plane-sweep algorithm is fast in practice; it used an insignificant fraction of the total runtime.

3.2.2 Computing Duplications in the Human Genome

Soon after the publication of the human genome, it was realized that the genome is replete with repetitive sequences [141]. Intra- and inter-chromosomal duplications have been found to play a vital role in genome evolution, its stability, and diseases [66], and knowing the location of such repeats can be important for many genomic analyses. Yet, fully annotating all repeats in a genome can be computationally challenging. To demonstrate the scalability of Mashmap2, we computed all ≥ 1 Kbp duplications in the hg38 human genome [142] with $\geq 90\%$ alignment identity. The importance of these duplications has been known for a long time [66, 69]; accordingly the UCSC genome browser also maintains them as a public database (named as segmental duplications) for the human genome.

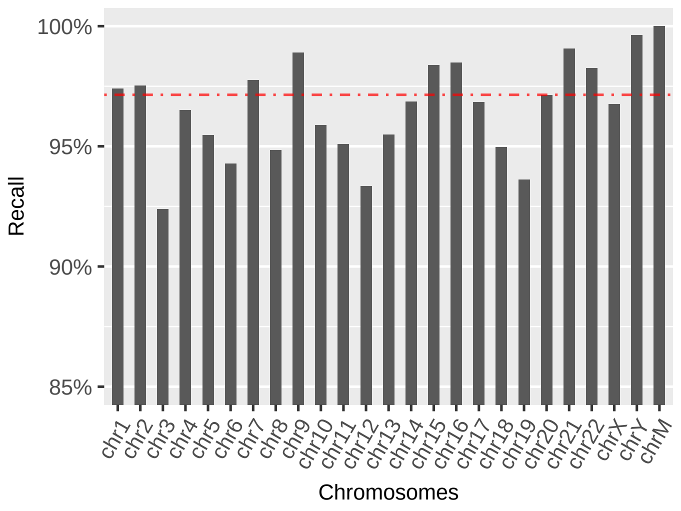


Figure 3.6: Recall scores of duplications computed using Mashmap2 against the UCSC segmental duplication database. Above 90% recall scores are achieved on each chromosome consistently. The red dotted line shows the aggregate recall score of 97.15% for the complete genome.

The goal of our experiment is to recover as many duplications as possible. Due to the probabilistic guarantees provided by our algorithm (Section 3.1.1), we expect it to compute such duplications with a high recall value. Typical genome-to-genome aligners including Minimap2, Nucmer4 and BLASTZ do not provide such guarantees, and typically require extensive parameter tuning as well as preprocessing of input to perform this task [e.g., 70, 69]. We summarize our method below and contrast our output with the UCSC database.

Methodology

We used 24 chromosome sequences (1-22, X,Y) and mitochondrial DNA from the hg38 version of the human genome as our input sequence set. To compute all ≥1 Kbp, ≥90% identity duplications, we directly used Mashmap2 with the same length and identity requirements, with filtering disabled. From its output, we discarded short $\leq$ 500bp mappings with $< 90\%$ estimated identity, plus the trivial duplications (i.e. regions matching with themselves), and were left with 2.1 billion candidate mappings. The count of reported mappings is high due to several high-copy repeat families in the genome, not all of which

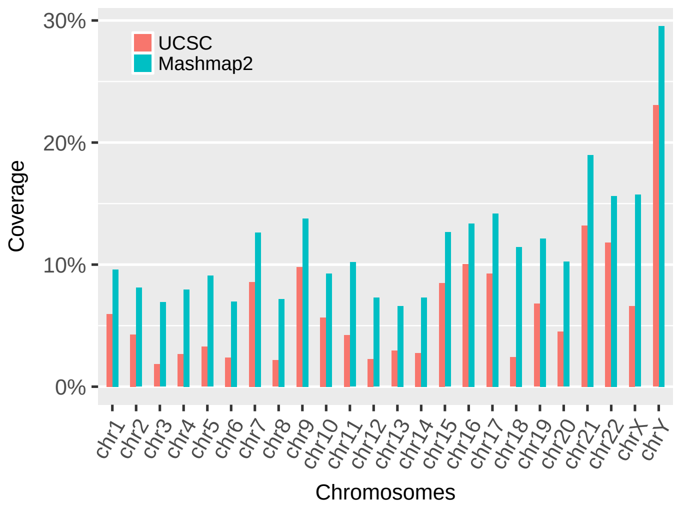


Figure 3.7: Comparison of genomic coverage between the UCSC Segmental Duplication database and Mashmap2 output alignments. Both methods reported equal coverage 83% on mitochondrial chromosome (not shown above to keep the plot legible). Coverage of duplications computed using our method is significantly higher, owing to its exhaustive search of all repeats with $\geq$1 Kbp length and $\geq$90% identity without repeat masking.

exceed our minimum thresholds. To remove the shorter or lower identity mappings, each of the approximate alignments was processed using LAST to compute a base-level alignment. This resulted in 210 million validated alignments with $\geq$1 Kbp length and $\geq$90% identity. We note that a large fraction of the candidate mappings failed to satisfy the specified cutoffs here. This is because Mashmap2 looks at the Jaccard similarity of k-mer sets to evaluate the mappings, but does not consider the distribution of k-mer match positions [143]. As a result, frequently occurring exact repeats of length < 1 Kbp in the human genome can also qualify as a match in the output. It may be be possible to improve the specificity by further considering the distribution of k-mer matches. This experiment took 120 CPU hours for executing Mashmap2 and $24,000$ CPU hours for validating all reported mappings using LAST. We show a dot-plot visualization of the reported alignments in Figure 3.5, which appears dense due to extensive duplications in the human genome. Finally, we converted the alignments into BED format to compare against the UCSC database using Bedtools [144]; the accuracy results are discussed next.

Accuracy Evaluation and Insights

The UCSC Segmental Duplications database for the hg38 human genome was computed using a standard pipeline proposed by [70], and was last updated in 2014. It is important to note that prior to computing genomic duplications, their method removed high-copy repeat elements (e.g., LINEs, *Alu*s) from the genome. Therefore, this database is not an exhaustive set of all $\geq$1 Kbp, $\geq$90% identity duplications in the genome, but a significant fraction of them. Nonetheless, low-copy repeat annotations have a higher likelihood of being missed by a mapper. Therefore, checking the recall against this database serves as an appropriate test to evaluate Mashmap2 in computing all homologous mappings of specified characteristics.

To measure recall on each chromosome, we computed coverage of those UCSC duplication annotations that have overlap with Mashmap2 duplications, and divided it by the coverage of all UCSC duplication annotations. Therefore, a 100% recall score would imply that all base-pairs which are annotated as segmental duplication in the UCSC database are part of one or more Mashmap2 alignments. We show these recall scores for each chromosome as well for the complete genome in Figure 3.6. Recall is consistently observed to be above 90% for each chromosome, and the aggregate recall for the complete genome is 97.15%. Among the 2.85% missed alignments, a large fraction of alignments were not recalled because difference in the alignment parameters can affect alignment identity and length. As a result, same regions can yield slightly different alignments using LAST and BLAST. If we relax the alignment identity and length cutoff in LAST to 88% and 950 bp respectively, the recall score improves to 98.28%. High recall scores achieved here, as well as in our prior experiments, demonstrate high sensitivity of our algorithm for any specified alignment characteristics by user, which is consistent with the theory in Section 3.1.1.

Finally, we compared the coverage of our alignments versus the UCSC database. Since our method did an exhaustive search of all duplications with $\geq$ 1 Kbp length and $\geq$ 90% identity without masking any genomic repeats, we observe that our algorithm attains either equal or higher coverage on each chromosome (Figure 3.7). For the complete genome, coverage of our alignments is 10.3%; 5% higher than the coverage of UCSC annotations. We further examined the subset of our duplications which do not overlap with UCSC segmental duplications. Indeed a large coverage fraction (82%) comprises of high-copy repeats (i.e. coverage depth > 50), potentially due to common repeat elements, which explains the wide gap in the coverage observed. The remaining 18% coverage fraction, however, is composed of low-copy repeats, with coverage depth ≤ 50 indicating the potential to uncover novel segmental duplications. Validating this possibility requires a more careful inspection of the output, and will be our future work. Mashmap2 alignments are available online at `https://gembox.cbcb.umd.edu/mashmap/index.html`.

3.3 Summary

In this chapter, we presented a fast algorithm for computing homology maps between whole genomes. We have given both theoretical and experimental evidence of the sensitivity provided, in terms of computing local alignment boundaries based on the minimum alignment length and identity parameters. To the best of our knowledge, this is the first practical and scalable algorithm to provide such guarantees. This formulation grants a convenient mechanism for users to execute this algorithm based on the underlying applications, which can be (but not limited to) mapping genome assembly of variable quality, aligning long reads to reference genomes, or computing segmental duplications in large genomes. Additionally, we formulated a filtering heuristic, and proposed an optimal plane-sweep based filtering algorithm for prioritizing alignments based on their scores and locations. The filtering algorithm is practically fast, accurate, and easy to implement in a few lines of code by using standard libraries. When mapping a human genome assembly to the human reference genome, Mashmap2 takes only about a minute from reading input sequences to generating the final alignment boundaries, identity estimates, and a dot-plot for visualization. Because of the underlying auto-tuning mechanism in Mashmap2, its runtime and memory-use depend on the sensitivity requirements provided to the algorithm.

CHAPTER 4
GENOME DISTANCE ESTIMATION FOR HIGH-THROUGHPUT ANI ANALYSIS

Whole-genome Average Nucleotide Identity (ANI) is recognized as the standard genome sequence similarity metric for microbial species classification. Existing ANI solvers either rely on time-consuming alignment routines, or sacrifice the output accuracy. In this chapter, we describe how to accelerate Average Nucleotide Identity (ANI) computation by leveraging our alignment-free mapping technique. Our new algorithm FastANI provides ANI values that are essentially identical to the alignment-based ANI values for both complete and draft quality genomes, while being two to three orders of magnitude faster. Accordingly, FastANI enables accurate estimation of pairwise ANI values for large cohorts of genomes.

Even though the term 'species' was conceived by Aristotle about 2,000 years ago to design a classification system for the organisms observed in nature, existence of such discrete biological units species in microbes is still debated [75]. By applying our algorithm to a collection of 90,000 prokaryotic genomes, we take a data-driven approach to address the following fundamental question: "do well-defined clusters of genomes (species) exist?"

4.1 Biological Relevance

Large collections of prokaryotic genomes with varied ecologic and evolutionary histories are now publicly available. This deluge of genomic data provides the opportunity to more robustly evaluate important questions in microbial ecology and evolution, as well as underscores the need to advance existing bioinformatics approaches for the analysis of such big genomic data. One such question is whether bacteria (and other microbes) form dis-

This chapter interpolates material from a paper by the author [44]

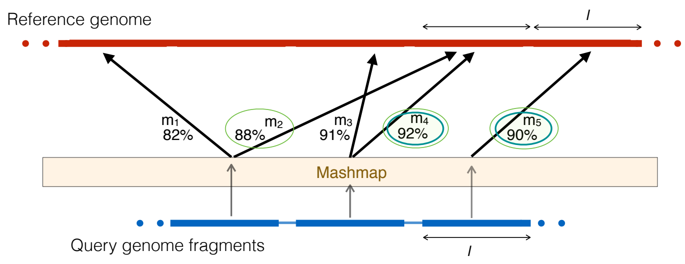
crete clusters (species), or, due to high frequency of horizontal gene transfer (HGT) and slow decay kinetics, a continuum of genetic diversity is observed instead. Studies based on a small number of closely related genomes have shown that genetic continuum may prevail [145]. On the other hand, other studies have argued that HGT may not be frequent enough to distort species boundaries, or that organisms within species exchange DNA more frequently compared to organisms across species, thus maintaining distinct clusters [146]. An important criticism of all these studies is that they have typically been performed with isolated genomes in the laboratory that may not adequately represent natural diversity due to cultivation biases, or were based on a small number of available genomes from a few phylogenetic lineages, which does not allow for robust conclusions to emerge. Therefore, it is still unclear if well-defined clusters of genomes are evident among prokaryotes and how to recognize them. Defining species is not only an important academic exercise but also has major practical consequences. For instance, the diagnosis of disease agents, the regulation of which organisms can be transported across countries and which organisms should be under quarantine, or the communication about which organisms or mixtures of organisms are beneficial to human, animals or plants, are all deeply-rooted on how species are defined.

One fundamental task in assessing species boundaries is the estimation of the genetic relatedness between two genomes. In recent years, the whole-genome average nucleotide identity (ANI) has emerged as a robust method for this task, with organisms belonging to the same species typically showing $\geq$95% ANI among themselves [73, 71]. ANI represents the average nucleotide identity of all orthologous genes shared between any two genomes and offers robust resolution between strains of the same or closely related species (i.e., showing 80-100% ANI). The ANI measure does not strictly represent core genome evolutionary relatedness, as orthologous genes can vary widely between pairs of genomes compared. Nevertheless, it closely reflects the traditional microbiological concept of DNA-DNA hybridization relatedness for defining species [73], as it takes into account the fluid nature of the bacterial gene pool and hence implicitly considers shared function.

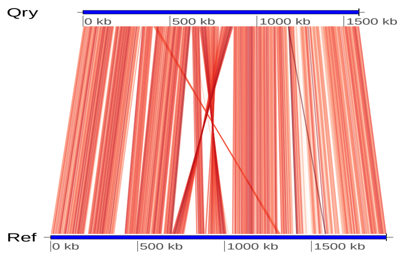
Sequencing of 16S rRNA genes is another highly popular, alternative traditional method for defining species and assessing their evolutionary uniqueness. However, methods based on single [147] or a set of universally conserved genes [148], such as 16S rRNA and ribosomal protein-encoding genes are often not applicable to incomplete genomes (e.g., the genes are not assembled), and these genes typically show higher sequence conservation than the genome average. Consequently, analysis of universal genes does not provide sufficient resolution at the species level [149], and has frequently resulted in lack of clear genetic discontinuities among closely related taxa [148]. ANI offers several important advantages such as higher resolution among closely related genomes. Finally, ANI can be estimated among draft (incomplete) genome sequences recovered from the environment using metagenomic or singe-cell techniques that do not encode universally conserved genes but encode at least a few hundred shared genes, greatly expanding the number of sequences that can be studied and classified compared to a universal gene-based approach. Accordingly, ANI has been recognized internationally for its potential for replacing DNA-DNA hybridization as the standard measure of relatedness, as it is easier to estimate and represents portable and reproducible data [74, 150]. Despite these strengths, to date ANI-based methods could not be applied for a large number of genomes due to their reliance on alignment-based searches [e.g., BLAST [54]], which are computationally expensive due to the quadratic time complexity of alignment algorithms [151].

4.2 The FastANI Algorithm

FastANI uses Mashmap as its underlying mapping framework. Analogous to the sequencing errors, here we assume that genome mutations occur independently and follow Poisson distribution. As a result, we can avoid direct alignments, but instead relate alignment identity between the sequences to Jaccard similarity of constituent k-mers (Section 2.1).



(a)



(b)

Figure 4.1: **a.** Graphical illustration of FastANI's work-flow for computing ANI between a query genome and a reference genome. Five mappings are obtained from three query fragments using Mashmap. $M_{forward}$ saves the maximum identity mapping for each query fragment. In this example, $M_{forward} = \{m_2, m_4, m_5\}$. From this set, $M_{reciprocal}$ picks m_4 and m_5 as the maximum identity mapping for each reference bin. Mapping identities of orthologous mappings, thus found in $M_{reciprocal}$, are finally averaged to compute ANI. **b.** FastANI supports visualization of the orthologous mappings $M_{reciprocal}$ that are used to estimate the ANI value using genoPlotR [152]. In this figure, ANI is computed between *Bartonella quintana* strain (NC_018533.1) as query and *Bartonella henselae* strain (NC_005956.1) as reference. Red line segments denote the orthologous mappings computed by FastANI for ANI estimation.

4.2.1 Extending Mashmap to Compute ANI

Previously established and widely used implementations of ANI begin by either identifying the protein coding genomic fragments [75] or extracting approximately 1 Kbp long overlapping fragments [73] from the query genome. These fragments are then mapped to the reference genome using BLASTn [54] or MUMmer [81], and the best match for each fragment is saved. This is followed by a reverse search, i.e., swapping the reference and query genomes. Mean identity of the reciprocal best matches computed through forward and reverse searches yields the ANI value. Rationale for this bi-directional approach is to bound the ANI computation to orthologous genes and discard the paralogs. In designing FastANI, we followed a similar approach while avoiding the alignment step.

FastANI first fragments the given query genome (A) into non-overlapping fragments of size l. These l-sized fragments are then mapped to the reference genome (B) using Mashmap. Mashmap first indexes the reference genome and subsequently computes mappings as well as alignment identity estimates for each query fragment, one at a time. At the end of the Mashmap run, all the query fragments $f_1, f_2 \ldots f_{\lfloor |A|/l \rfloor}$ are mapped to B. The results are saved in a set M containing triplets of the form $\langle f, i, p \rangle$, where f is the fragment id, i is the identity estimate, and p is the starting position where f is mapped to B. The subset of M (say $M_{forward}$) corresponding to the maximum identity mapping for each query fragment is then extracted. To further identify the reciprocal matches, each triplet $\langle f, i, p \rangle$ in $M_{forward}$ is 'binned' based on its mapping position in the reference, with its value updated to $\langle f, i, bin \rangle = \langle f, i, \lfloor p/l \rfloor \rangle$. Through this step, fragments which are mapped to the same or nearby positions on the reference genome are likely to get equal bin value. Next, $M_{reciprocal}$ filters the maximum identity mapping for each bin. Finally, FastANI reports the mean identity of all the triplets in $M_{reciprocal}$ (See Figure 4.1 for an example and visualization).

We define τ as an input parameter to FastANI to indicate a minimum count of reciprocal mappings for the resulting ANI value to be trusted. It is important to appropriately choose the parameters (l, τ and I_0).

4.2.2 Algorithmic Parameter Settings

FastANI is targeted to estimate ANI in the 80%-100% identity range. Therefore, it calls Mashmap mapping routine with an identity cutoff $I_0 = 80\%$, which enables it to compute mappings with alignment identity close to 80% or higher.

Choosing an appropriate value of query fragment l requires an evaluation of the trade-off between FastANI's computation efficiency and ANI's estimation accuracy. Higher value of l implies less number of non-overlapping query fragments, thus reducing the overall runtime. However, if l is much longer than the average gene length, a fragment could span more than one conserved segment, especially if the genome is highly recombinant. We empirically evaluated different values of l and set it to 3 Kbp (Table A.1). Last, we set τ to 50 to avoid incorrect ANI estimation from just a few matching fragments between genomes that are too divergent (e.g., showing $< 80\%$ ANI). With $l = 3$ Kbp, $\tau = 50$ implies that we require at least 150 Kbp homologous genome sequence between two genomes to make a reliable ANI estimate, which is a reasonable assumption for both complete and incomplete genome assemblies based on our previous study [153].

Software and Data Availability FastANI can be downloaded at `https://github.com/ParBLiSS/FastANI`. All the datasets used in this study are available at `http://enve-omics.ce.gatech.edu/data/fastani`.

4.3 Results

We first demonstrate FastANI yields accuracy on par with the widely accepted BLASTn based approaches, and then leverage its computational efficiency to analyze genomic relatedness within and across species.

4.3.1 Benchmark Datasets

To test accuracy and speed, we evaluated FastANI on both high-quality closed genomes from NCBI RefSeq database as well as publicly available draft genome assemblies. We first removed poor quality genome assemblies with low N50 length (< 10 Kbp). In total, five datasets were used, D1 through D5 (see Table 4.1). Dataset D1 is the set of closed prokaryotic genomes downloaded from RefSeq database. Datasets D2, D3, and D4 include draft genome assemblies of isolates of *Bacillus cereus s.l.*, *Escherichia coli*, and *Bacillus anthracis*, respectively, downloaded from the prokaryote section of the NCBI Genome database. Dataset D5 includes a recently published large collection of metagenome-assembled genomes (MAGs) [154]. These sizable datasets represent genomes showing different levels of identity among themselves and varying values of completeness and assembly quality (Figure 4.2). For each dataset, one genome was selected as the query genome and its ANI was computed with every genome in the complete dataset. In all cases except in D1, query genome strains were selected randomly.

Table 4.1: Datasets used for testing accuracy and speed of FastANI.

Id	Reference clade	No. of Genomes	Median N50 (Mbp)	Query Genome
D1	NCBI RefSeq	1,675	3.14	*E. coli* K-12 MG1655
D2	*Bacillus cereus s.l.*	570	1.16	*B. anthracis* 52-G
D3	*Escherichia coli*	4,271	0.15	*E. coli* 0.1288
D4	*Bacillus anthracis*	464	0.59	*B. anthracis* 2000031001
D5	MAGs [154]	7,897	0.04	*Acinetobacter* sp. UBA6007

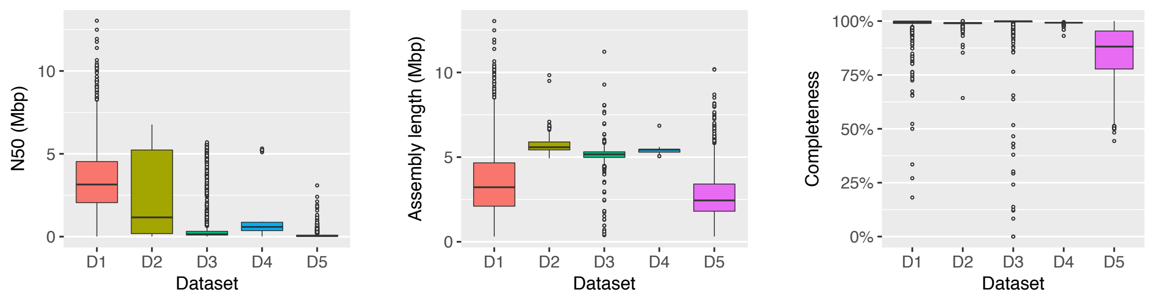


Figure 4.2: N50, genome-length and completeness distribution for the five datasets D1-D5 is shown using boxplots. Genome completeness was estimated using the presence of marker genes in CheckM (v1.0.3) [155]. All five datasets exhibit different assembly N50 and length characteristics. As expected, majority of genomes in datasets D1-D4 (isolates) are complete, whereas metagenome-assembled genomes (MAGs) in the D5 dataset have low completeness.

4.3.2 Accuracy Evaluation

We evaluated FastANI against the BLASTn based method [77] of computing ANI, henceforth referred as ANI_b, and the ANI values predicted by the Mash [82] (v1.1) tool. User documentation for Mash recommends using larger sketch size (i.e., k-mer sample) than the default to obtain higher accuracy [82]. Accordingly, we ran Mash with both the default sketch size of $1K$ as well as increase it up to $100K$.

FastANI achieves near perfect linear correlation with ANI_b on all datasets D1-D5 (Figure 4.3, Table 4.2). Mash results improve with increasing sketch size, particularly for D1. However, even when executed with the largest sketch size of $100K$, Mash results diverge from ANI_b values on datasets D1, D3 and D4. For D1, this primarily appears to be caused by divergent genomes (e.g., showing $< 90\%$ ANI). For D3, Mash diverges on closely related genomes due to fragmented and incomplete genome assemblies of the draft genomes. Dataset D4 is challenging because its constituent genomes are closely related strains of *Bacillus anthracis*, with $\text{ANI}_\text{b} > 99.9$ for all the pairs. FastANI provides much better precision than Mash in D4 dataset, and therefore, can be used to discriminate between very closely related microbial strains such as those of different epidemic outbreaks. However, for two genomes out of the 464, FastANI estimates are diverging from ANI_b. To investigate further, we visualized gene synteny pattern using Mauve [156] and found that these two genome sequences have many re-arrangements with respect to the query genome (Figure A.1). Given that *B. anthracis* strains typically show high genome synteny [157], these results indicate that the two genomes were poorly assembled. Incorrect data will yield unpredictable results not only with FastANI but using any method that assesses genetic relatedness, including phylogeny-based methods. If the two incorrect *B. anthracis* assemblies are removed, FastANI's correlation with ANI_b improves to 0.944 in D4.

These correlation results demonstrate that FastANI provides significant quality improvement over Mash (see Table 4.2), and can be a reasonable substitute for ANI_b. Although this experiment was conducted with single query genome per dataset, increasing

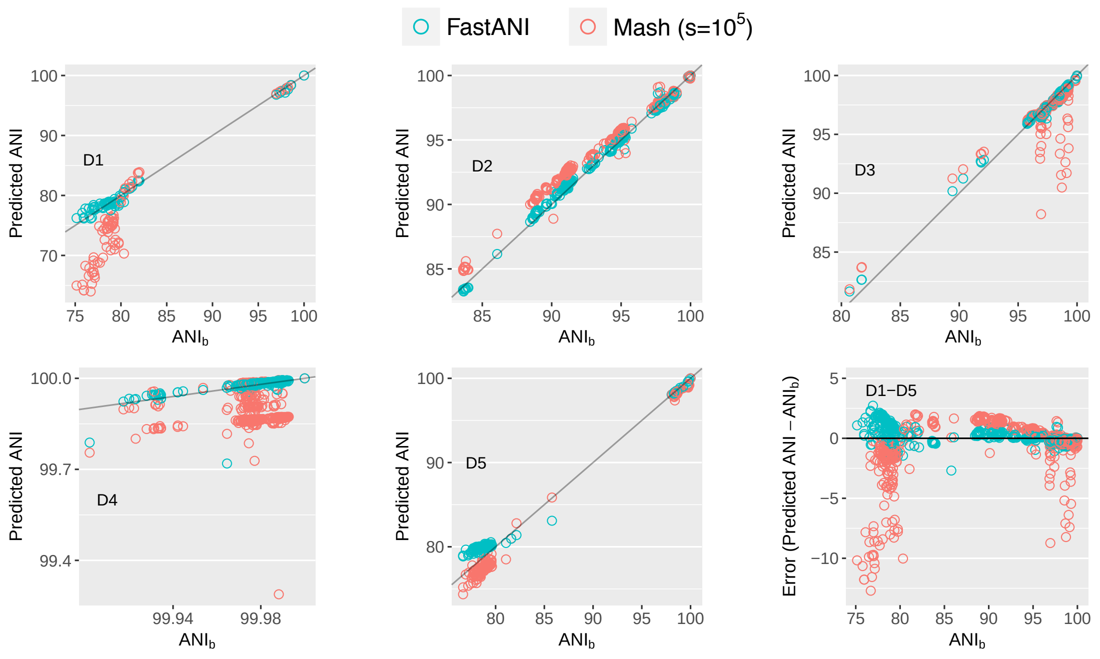


Figure 4.3: Plots showing how FastANI and Mash-based ANI (sketch size $= 10^5$) output correlate with ANI_b values for datasets D1-D5. Because FastANI assumes a probabilistic identity cutoff that is set to 80% by default, it reports 76, 570, 4,271, 464 and 130 genome matches for the individual queries in datasets D1-D5 respectively. To enable a direct quality comparison against FastANI, Mash is executed for only those pairs that are reported by FastANI. Notice that each dataset encompass a different nucleotide identity range (x-axes). Gray line represents a straight line $y = x$ plot for reference. Pearson correlation coefficients corresponding to these plots are listed separately in Table 4.2. Last plot shows error of these methods *w.r.t.* ANI_b using all five datasets.

the count of query genomes did not affect our conclusions (Figure A.2). Further, FastANI estimates were accurate for draft genomes, in the range of $20\% - 100\%$ completeness (Figure 4.4).

We also highlight the effect of genome completeness and contamination on FastANI's accuracy using simulated datasets (Figure 4.5). The completeness and contamination were simulated by manipulating the read composition of E. *coli* str. O157-H7 prior to its assembly, i.e., under-sampling the reads at various scales to induce incompleteness and adding reads of *Pseudomonads aeruginosa* str. PAO1 to induce contamination. Degrees of completeness and contamination were measured using the gene composition of assembled genome. Results show that the contamination had almost no effect at all on FastANI's

Table 4.2: Comparison of FastANI and Mash-based ANI accuracy by measuring their Pearson correlation coefficients with ANI_b values. Mash is executed with sketch sizes (-s): 1,000 (default), 10,000 and 100,000. FastANI achieves > 0.99 correlation with ANI_b in all cases but D4. Its correlation value on D4 improves from 0.681 to 0.944 if the two poor assemblies present in D4 are not taken into account.

Dataset	FastANI	Mash		
		-s 10^3	-s 10^4	-s 10^5
D1	**0.995**	0.594	0.932	0.935
D2	**0.999**	0.996	0.997	0.997
D3	**0.995**	0.944	0.944	0.944
D4	**0.681**	-0.040	0.003	0.010
D5	0.998	0.634	0.997	**0.999**

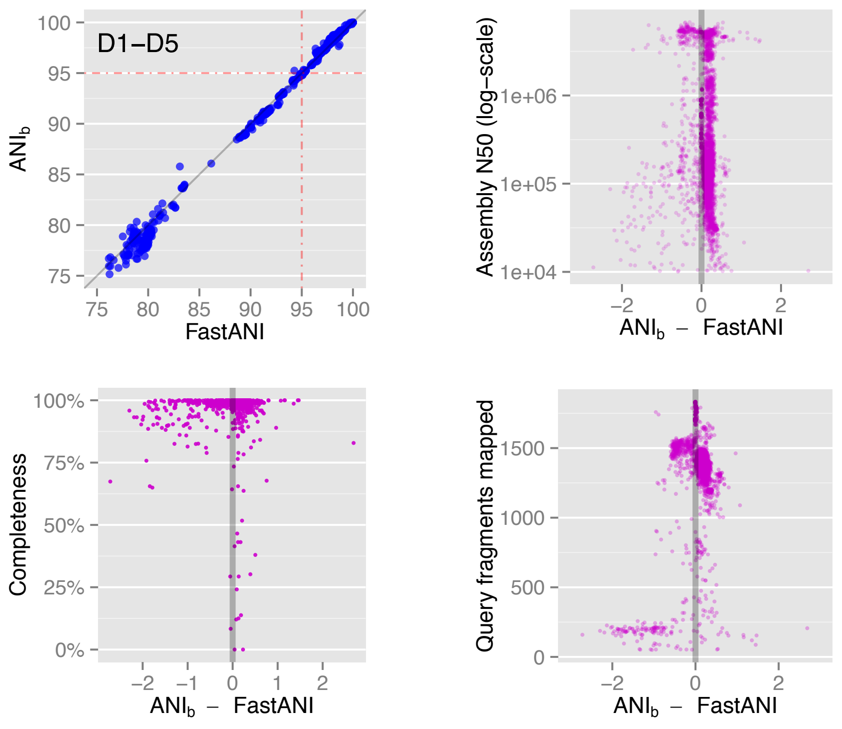


Figure 4.4: FastANI's aggregate accuracy and error characteristics based on datasets D1-D5. Upper left plot shows the FastANI and ANI_b correlation. The remaining three plots show differences between FastANI and ANI_b value versus reference genome assembly quality (N50 and genome completeness) and the number of reciprocal fragments that matched between query and reference genome for each comparison. Overall, these results show no significant biases associated with these factors. In the bottom-right plot, we observe that the difference is relatively higher when there are few reciprocal fragments, which typically happens for distant genomes (i.e., ANI close to 80%).

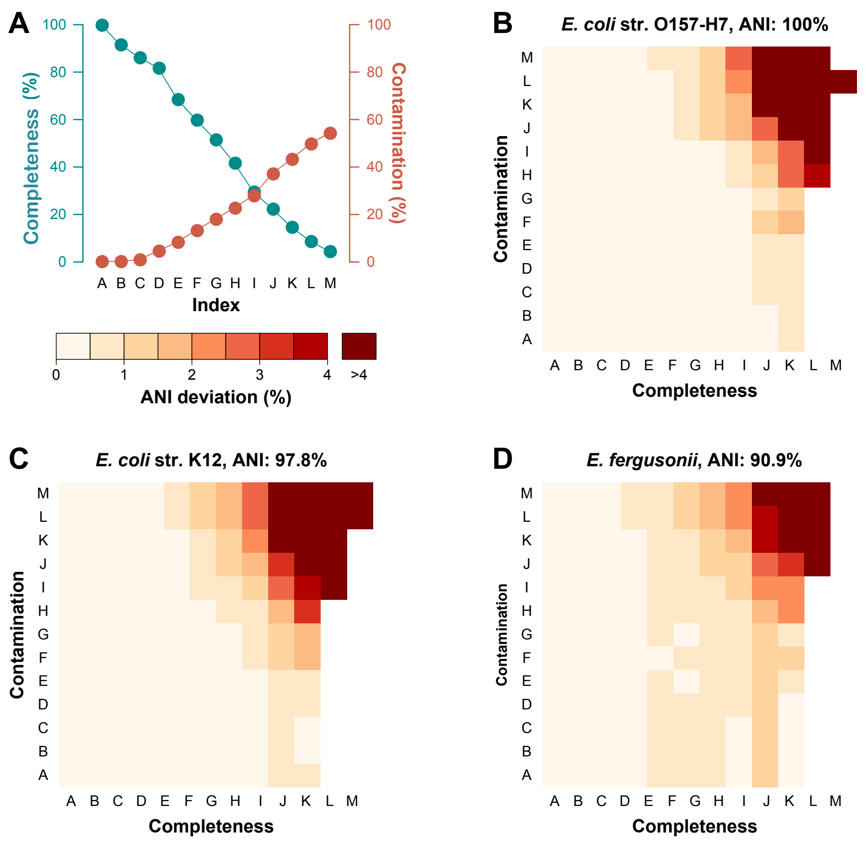


Figure 4.5: FastANI's accuracy evaluation using simulated datasets. Levels A to M in Panel A indicate decreasing genome completeness and increasing contamination. For each combination of completeness and contamination, FastANI's accuracy with respect to BLAST-based ANI (ANI_b) is presented. Panels B-D show FastANI's deviation from ANI_b when computing ANI between the simulated E. coli str. O157-H7 genome against the reference genomes of E. *coli* str. O157-H7 genome (ANI=100%), E. *coli* str. K12 genome (ANI=97.8%) and E. *fergusonii* genome (ANI=90.9%) respectively. Missing (blank) values are those that FastANI failed to estimate due to insufficient hits.

output quality, except when there was also a very low completeness. Also, when completeness was $> 50\%$ there was basically no effect, but around 20% completeness and below, the estimate became unreliable which is likely true for BLAST-based ANI as well. This implies that FastANI can tolerate variable assembly quality, completeness and contamination. Most importantly, it correlates well with ANI_b in the desired identity range of $80\% - 100\%$.

4.3.3 Computational Speedup

FastANI is designed to efficiently process large assembly datasets with modest compute resources. For FastANI's sequential and parallel runtime evaluation, we used a single compute node with two Intel Xeon E5-2698 v4 20-core processors. First we show runtime comparison of FastANI and ANI_b using serial execution (single thread, single process) using all datasets in Table 4.3. FastANI operation consists of indexing phase followed by compute phase, for which we measured the runtime separately. For any database, indexing all the reference genomes needs to be done only once, and thereafter, FastANI can compute ANI estimates for any number of input query genomes against the reference genomes. Therefore, speedup in Table 4.3 is measured with respect to FastANI compute time. We observe that the runtime improvement due to FastANI varied from 50x for D3 to 4608x for D5. FastANI speed-up is much higher on D1 and D5 because these datasets contain a diverse set of prokaryotic genomes. This is attributable to the fact that the algorithm underlying FastANI is able to prune distant genomes (ANI $\ll 80\%$) efficiently. On the contrary, ANI values for all genomes in datasets D2-D4 were high ($> 80\%$). Note that replacing BLASTn with faster alignment software in ANI_b does not improve its performance significantly. A recent survey of ANI methods [79] reported speedups of only up to 4.7x by using Usearch [80] and MUMmer [81].

To accelerate ANI computation even further, FastANI can be trivially parallelized using multi-core parallel execution. One way to achieve this is to split the reference genomes in several equal-size parts. This way, each instance of FastANI process can search query genome(s) against each part of the reference database independently. We utilized this scheme and evaluated scalability using up to 80 FastANI parallel processes. Compared to the sequential execution time listed in Table 4.3, runtime of the compute phase reduced to 2, 8, 46, 6 and 1 second for datasets D1-D5 respectively (Figure 4.6). These results confirm that FastANI can be used to query against databases containing thousands of genomes in a few seconds.

Table 4.3: Comparison of execution time of FastANI versus ANI$_b$. Speedup in the last column is measured as the ratio of ANI$_b$'s runtime and FastANI's compute time.

Dataset	FastANI		ANI$_b$ (sec)	Speedup
	Indexing (sec)	Compute (sec)		
D1	468.2	16.76	13,113	**782x**
D2	195.7	264.8	18,155	**69x**
D3	1,538	1,981	99,317	**50x**
D4	128.8	214.5	11,051	**52x**
D5	2,784	14.88	68,571	**4608x**

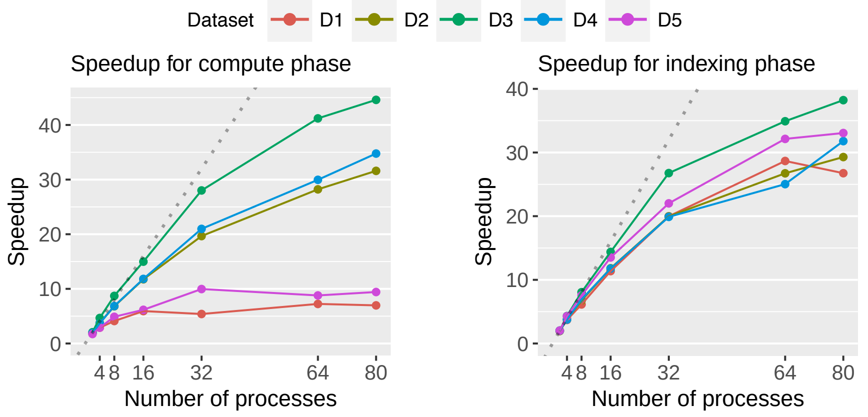
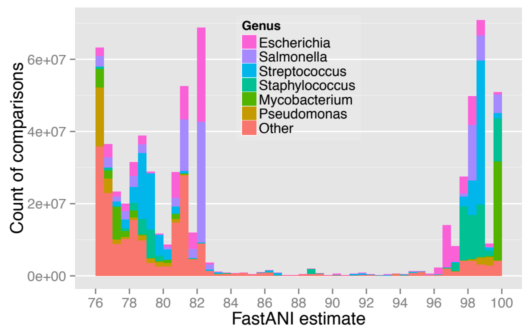


Figure 4.6: Scaling results of FastANI's execution time using datasets D1-D5 on a compute node with 40 physical cores. We executed parallel FastANI processes where each process was assigned an equal sized random part of the reference database for computing ANI. Left and right plots evaluate FastANI's compute and indexing phase, respectively. FastANI achieves reasonable speedups on all datasets except the compute phase in D1 and D5, as their runtime on a single core is too small to begin with (Table 4.3).

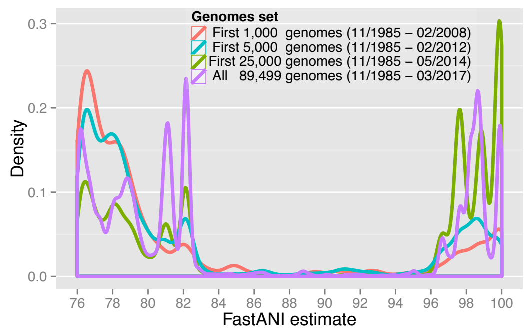
For the above experiments, FastANI required a maximum 62 GB memory for D5, our largest dataset for this experiment. For databases much larger than D5, peak memory usage can be reduced by either distributing the compute across multiple nodes in a cluster or processing chunks of the reference database one by one, as necessary.

4.3.4 Large-scale Pairwise Comparison Indicates Genetic Discontinuity

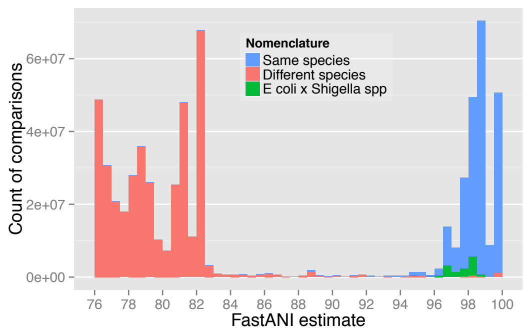
We examined the distribution of pairwise ANI values between all 91,761 prokaryotic assemblies that existed in the NCBI Genome database as of March 15, 2017. Prior to analysis, we removed 2,262 genomes due to short N50 length ($<$ 10 Kbp). In our evaluation, the ANI



(a)



(b)



(c)

Figure 4.7: **a.** Histogram plot showing the distribution of ANI values among the 90K genomes. Only ANI values in the 76-100% range are shown. Out of total 8.01 billion pairwise genome comparisons, FastANI reported only 17M ANI values (0.21%) with ANI between 83% and 95% indicating a wide genetic discontinuum. Multiple colors are used to show how genomes from different genera are contributing to this distribution. Few peaks in the histogram arise from genera that have been extensively sequenced and dominate the database. **b.** Density curves of ANI values in the ANI range 76-100%. Each curve shows the density curve corresponding to the database at a particular time period. Wide discontinuity in all four curves is observed consistently. **c.** Distribution of ANI values with each comparison labeled by the nomenclature of genomes being compared. All the comparisons between *Escherichia coli* and *Shigella* spp. have been labeled separately. The 95% ANI threshold on x-axis serves as a valid classifier for comparisons belonging to same and different species.

between each pair of genomes A and B is computed twice, once with A as query genome and again with B as query genome. This choice did not meaningfully alter the ANI value reported by FastANI unless the draft genomes are incorrectly assembled or contaminated. Computing pairwise ANI values for the entire database took 77K CPU hours for all 8.01 billion comparisons. To our knowledge, this is the largest cohort of genomes for which ANI has been computed. In comparison, the largest previously published ANI analysis included 86 million comparisons and took 190K CPU hours [149]. Among the total of 8.01 billion pairwise comparisons, 679,765,100 yielded ANI values in the 76-100% range. The distribution of these ANI values reveals a clear and wide discontinuity in the identity range of 83-95% (Figure 4.7a). FastANI reported only 17,132,536 ANI values (i.e., 2.5% of the 679,765,100 pairs) within the range of 83% to 95%. When performing this analysis using Mash, the bimodal distribution of ANI values was persistent (Figure A.3).

The frequency of intra- vs. inter-species genomes sequenced in the NCBI database has changed over time, with earlier sequencing efforts targeting distantly related organisms in order to cover phylogenetic diversity while efforts in more recent years targeted more closely related organisms for micro-diversity or epidemiological studies. We confirmed that discontinuity pattern has been maintained at different time points in the past (Figure 4.7b). In previous taxonomic studies, 95% ANI cutoff is the most frequently used standard for species demarcation. Density curves in the figure show that the two peaks consistently lie on either side of the 95% ANI value.

To further test the validity of the hypothesis that 95% ANI score can demarcate species boundaries, we examined correlation between standing nomenclature and the 95% ANI-based demarcation. As per this standard, we should expect a pair of genomes to have ANI value $\geq 95\%$ if and only if both genomes are classified as same species in the existing taxonomy. From the complete set of 89,499 genomes, we identified the subset for which we could determine the named species for each genome. Whenever available (9% of the total genomes), we recovered the links to NCBI taxonomy to determine the species.

For the remainder of the genomes, we inferred the species from the organism name given in the GenBank file, excluding all entries with ambiguous terms (sp, cf, aff, bacterium, archeon, endosymbiont), resulting in the species-wise classification of an additional 78% of the genomes. The remainder 13% of the genomes lacked clear nomenclature and hence could not be reliably assigned to a named species for the purpose of this test.

We evaluated the distribution of ANI values in comparison to the named species that the corresponding genomes were assigned to (Figure 4.7c). The $\geq$ 95% ANI criterion reflects same named species with a recall frequency of 98.5% and a precision of 93.1%. We further explored the values affecting precision, i.e., 6.9% of ANI values above 95% that were obtained for genomes assigned to different named species. Among those, 5.6% are due to comparisons between *Escherichia coli* and *Shigella* spp., a case in which the inconsistency between taxonomy and genomic relatedness is well documented [145] (highlighted in green in Figure 4.7c). The remaining 1.3% of the cases mostly exist within the *Mycobacterium* genus (0.5%), which includes a group of closely related named species as part of the *M. tuberculosis* complex such as *M. tuberculosis* (reference), *M. canettii* (ANI 97-99% against reference), *M. bovis* (ANI 99.6%), *M. microti* (ANI 99.8-99.9%), and *M. africanum* (ANI 99.9%), among others. An additional 0.2% of the cases correspond to comparisons between *Neisseria gonorrhoeae* and *N. meningitidis*, two species with large representation in the database and ANI values close to 95% (Inter-quartile range: 94.9-95.2%). Excluding the cases of *E. coli* vs. *Shigella* spp. alone, precision increases to 98.7%. With both recall and precision values $\geq$ 98.5%, these results corroborate the utility of ANI for species demarcation, which is consistent with previous studies based on a much smaller datasets of genomes [71, 75, 158, 149].

4.3.5 Weighing the Cultivation Bias

The genetic discontinuity was apparent even when the species with large count of sequenced representatives such as pathogenic bacteria of human or animal hosts were iteratively removed from the analysis (Figure A.4), or when genomes were randomly drawn with species-dependent probabilities that ensured equal representation of highly sampled and sparsely sampled species in the final set (Figure A.5). To account for the possible influence of cultivation bias on our conclusions, we sampled five genomes from each of the 750 named species with ≥ 5 genomes present in the database. Even though the percentage of the inter-species pairs remains small within the 83-95% valley range (0.2%), discontinuity appears to be less pronounced (Figure A.5). The latter was attributable to the fact that several highly sampled species have closely related species (of "intermediate" identity) that include relatively fewer sequenced representatives; thus, subsampling the genomes of the former species affected more the frequency of ANI values in the 95-100% relative to the 83-95% range.

The above results might indicate that cultivation biases could have accounted, at least in part, for the wide discontinuity observed. Cultivation biases could include, for instance, a historical tendency to preserve the isolates that meet the known/expected phenotypic criteria of the species and discard the remaining ones, which could represent "outlier" or "intermediate" genomes in terms of phenotypic and genetic similarity, or biases of the cultivation media and conditions against such "intermediate" genomes. However, given that these highly sampled species represent several distinct major prokaryotic lineages, it is likely that the discontinuity represents a real biological signature and is not driven by cultivation or other biases (or the latter should have been uniformly applied to several different isolation procedures and lineages of the highly sampled species and their close relatives). It is also important to note that these results are consistent with cultivation-independent metagenomics analysis of natural microbial communities, which have showed that the communities are composed of predominantly sequence-discrete populations [159]. Moreover, the discontinuity pattern observed using the collection of 8,000 MAGs recovered from different habitats (Figure 4.3, D5) is remarkably similar to the discontinuity observed among isolate genomes (Figure 4.7).

The biological mechanisms underlying this genetic discontinuity are not clear but should be subject of future research for a more complete understanding of prokaryotic species. The mechanisms could involve a dramatic drop in recombination frequency around 90-95% ANI, which could account for the discontinuity if bacteria evolve sexually [160], ecological sweeps that remove diversity due to competition [161, 162], or stochastic neutral processes [163, 164]. A genomic nucleotide diversity of 5-10% translates to tens of thousands of years of evolution time, which provides ample opportunities for ecological or genetic sweeps to occur. Nonetheless, the existence of genetic discontinuity among 90K genomes represents a major finding that can help define species more accurately and has important practical consequences for recognizing and communicating about prokaryotic species.

4.4 Summary

FastANI accurately estimates ANI values between both complete and draft genomes while reducing the computing time by two to three orders of magnitude. We leveraged the computational efficiency offered by FastANI to evaluate the distribution of ANI values in a set of over 90,000 genomes, and demonstrate that genetic relatedness discontinuity can be consistently identified among these genomes around 95% ANI. This discontinuity is recovered with or without the most frequently represented species in the database, is robust to historic additions in the public databases, and it represents an accurate threshold for demarcating almost all currently named prokaryotic species. While this genetic discontinuity has been observed previously [71, 75, 158, 149], the FastANI-based results reported here show a sharper discontinuity while using a much larger set of genomes by at least an order of magnitude.

As genome sequences from isolates, and more recently using metagenomics and single-cell amplification are flooding public databases [46, 154, 165], FastANI provides a scalable solution and a useful addition to the genomic toolkit. We expect FastANI to be useful for analysis of both clinical and environmental microbial genomes. It can be used for studying the inter- and intra-species diversity within large collections of genomes, including genomes showing various levels of completeness. It can also accelerate the study of the novelty of new species or phenotypic similarity of a query genome sequence in comparison to all available genomes. FastANI and Mash gave comparable ANI estimates for complete genomes, but the advantages of FastANI for draft (incomplete), divergent (<90% ANI) or highly related (>99.5% ANI) genomes are significant (Figure 4.3), and thus, FastANI should be the preferable method.

CHAPTER 5
COMPLEXITY OF READ TO GRAPH ALIGNMENT

Aligning reads to graphs is becoming increasingly important in the context of several applications in computational biology. The sequence to graph alignment problem seeks the best matching path in a graph $G(V, E)$ for an input query sequence. In this chapter, we study sequence to graph alignment problems under Hamming and edit distance models, and linear and affine gap penalty functions, for multiple variants of the problem that allow changes in query alone, graph alone, or in both. We prove that when changes are permitted in graphs either standalone or in conjunction with changes in the query, the sequence to graph alignment problem is $\mathcal{NP}$-complete under both Hamming and edit distance models for alphabets of size ≥ 2. These results improve upon previous hardness results which assume an alphabet of size $\geq |V|$ [107]. On the other hand, it is known that the alignment problem is polynomially solvable when changes are allowed on the query sequence alone [107, 108]. Based on the recently discovered lower bounds, it is unlikely that the best existing polynomial time algorithms can be further improved. See Figure 5.1 for an overview.

5.1 Preliminaries

Let Σ denote an alphabet, and x and y be two strings over Σ. We use $x[i]$ to denote the i^{th} character of x, and $|x|$ to denote its length. Let $x[i, j]$ $(1 \leq i \leq j \leq |x|)$ denote $x[i]x[i+1] \ldots x[j]$, the substring of x beginning at the i^{th} position and ending at the j^{th} position. Concatenation of x and y is denoted as xy. Let x^k denote string x concatenated with itself k times.

Definition 5.1.1. Sequence Graph: A sequence graph $G(V, E, \sigma)$ is a directed graph with

This chapter interpolates material from a paper by the author [108]

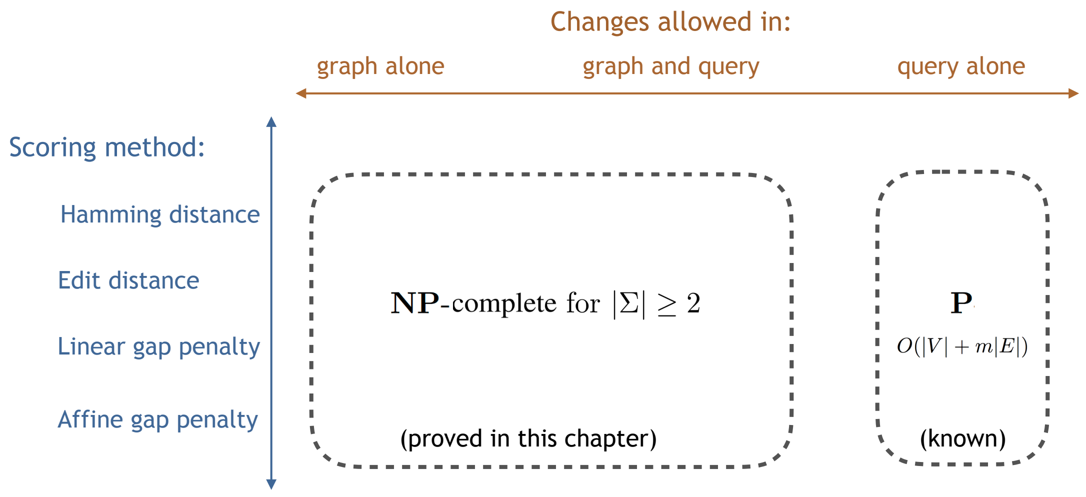


Figure 5.1: Complexity results for the read to graph alignment problem variants.

vertices V and edges E. Function $\sigma : V \to \Sigma^+$ labels each vertex $v \in V$ with string $\sigma(v)$ over the alphabet Σ.

Naturally, path $p = v_i, v_{i+1}, \ldots, v_j$ in $G(V, E, \sigma)$ spells the sequence $\sigma(v_i)\ \sigma(v_{i+1}) \ldots \sigma(v_j)$. Given a query sequence q, we seek its best matching path sequence in the graph. Alignment problems are formulated such that distance between the computed path and the query sequence is minimized, subject to a specified distance metric such as Hamming or edit distance. Typically, an alignment is scored using either a linear or an affine gap penalty function. The cost of a gap is proportional to its length, when using a linear gap penalty function. An affine gap penalty function imposes an additional constant cost to initiate a gap.

5.2 Asymmetry of Edit Locations

An alignment between two sequences also specifies possible changes to the sequences (e.g. substitutions, insertions, deletions) to make them identical, with alignment distance specifying the cumulative penalty for the changes. The changes can be individually applied either to the first or the second sequence, or any combination thereof. Such a symmetry is no longer valid when aligning sequences to graphs [107]. This is because alignments can occur along cyclic paths in the graph. If the label of a vertex in the graph is changed, then an alignment path visiting that vertex k times reflects the same change at k different positions in the alignment. On the other hand, a change in one position of the sequence only reflects that change in the corresponding position in the alignment. As such, optimal alignment scores vary depending on whether changes are permitted in just the sequence, just the graph, or both (see Figure 5.2 for an illustration). This characteristic leads to *three different problems*, with each potentially resulting in a different optimal distance.

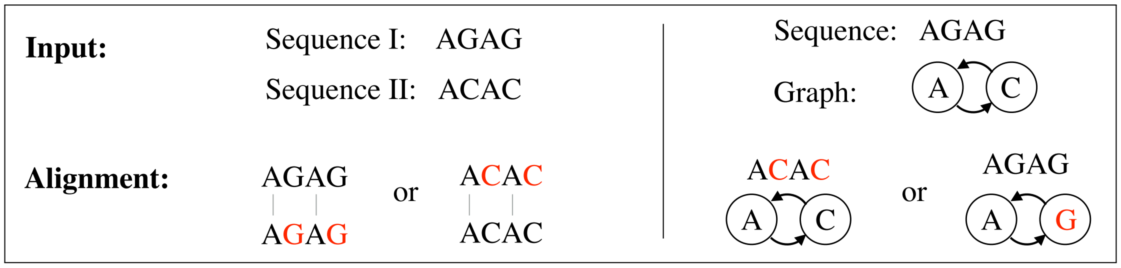


Figure 5.2: Asymmetry with respect to the location of changes in sequence to graph alignment illustrated using Hamming distance. Two substitutions are required in the sequence, whereas just one is sufficient if made in the graph.

Consider the sequence to graph alignment problem under the Hamming or edit distance metrics. For each distance metric, there are three versions of the problem depending on whether changes are allowed in query alone, graph alone, or both in the query and graph. Consider the decision versions of these problems, which ask whether there exists an alignment with $\leq d$ modifications (substitutions or edits), as per the distance metric. Restricting substitutions or edits to the query sequence alone admits polynomial time solutions [107, 20, 109]. In the pioneering work of Amir *et al.* [107] in the domain of string to hypertext matching, it has been proved that the other problem variants which permit changes to graph are $\mathcal{NP}$-complete. The proofs provided in their work assume an alphabet size $\geq |V|$. To date, tractability of these problems remains unknown for the case of constant sized alphabets (e.g., for DNA, RNA, or protein sequences). In what follows, we close this knowledge gap by showing that the problems remain $\mathcal{NP}$-complete for any alphabet of size at least 2.

5.3 Alignment using Hamming Distance

Theorem 4. *The problem "Can we substitute a total of $\leq d$ characters in graph G and query q such that q will have a matching path in G?" is $\mathcal{NP}$-complete for $|\Sigma| \geq 2$.*

Proof. The problem is in $\mathcal{NP}$. Given a solution, the set of substitutions can be used to obtain the corrected graph and query. Next, we can leverage any polynomial time algorithm [107, 20, 105] to verify if the corrected query matches a path in the corrected graph.

To show that the problem is $\mathcal{NP}$-hard, we perform a reduction using the directed Hamiltonian cycle problem. Suppose $G'(V, E)$ is a directed graph in which we seek a Hamiltonian cycle. Let $n = |V|$. We transform it into a sequence graph $G(V, E, \sigma)$ over the alphabet $\Sigma = \{\alpha, \beta\}$ by simply labeling each vertex $v \in V$ with α^n (Figure 5.3). Note that the graph structure remains unchanged. Next, we construct query sequence q. Let token t_i be the sequence of n characters $\alpha^{n-i-1}\beta\alpha^i$. We choose query q to be the $n^2(2n+2)$ long sequence: $(t_0 t_1 \dots t_{n-1})^{2n+2}$. We claim that a Hamiltonian cycle exists in $G'(V, E)$ if and only if q can be matched after substituting a total of $\leq n$ characters in $G(V, E, \sigma)$ and q.

Suppose there is a Hamiltonian cycle in $G'(V, E)$. We can follow the corresponding loop in $G(V, E, \sigma)$ from the first character of any vertex label. To match each token in the query q, we require one $\alpha \rightarrow \beta$ substitution per vertex. Thus, the query q matches $G(V, E, \sigma)$ after making exactly n substitutions in the graph.

Conversely, suppose the query q matches the graph $G(V, E, \sigma)$ after making $\leq n$ substitutions in the query and the graph. Consider the following substring q_{sub} of q: $t_0 t_1 \dots t_{n-1} t_0 t_1$. Note that there are $n + 1$ non-overlapping instances of q_{sub} in q. Even if all the n substitutions occur in the query, at least one instance of q_{sub} must remain unchanged. As a result, q_{sub} must match to a path in the corrected $G(V, E, \sigma)$.

Case 1: q_{sub} starts matching from the first character of a vertex label. Note that the first n tokens $q_{sub}[1, n] = t_0$, $q_{sub}[n+1, 2n] = t_1$, …, $q_{sub}[n^2 - n + 1, n^2] = t_{n-1}$ are all unique followed by $q_{sub}[n^2+1, n^2+n] = t_0$. Therefore, this requires a Hamiltonian cycle in $G(V, E, \sigma)$. Accordingly, there is a Hamiltonian cycle in $G'(V, E)$.

Case 2: q_{sub} starts somewhere other than the starting position within a vertex label. Let $q_{sub}[k]$ $(1 < k \leq n)$ be the first character that matches at the beginning of the next vertex on the path matching q. Similar to the previous case, the following n sequences $q_{sub}[k, n+k-1]$, $q_{sub}[n+k, 2n+k-1], \dots, q_{sub}[n^2-n+k, n^2+k-1]$ are unique due to the spacing between β characters in q_{sub}. Therefore, the matching path must yield a Hamiltonian cycle.

□

Corollary 1. *The problem "Can we substitute $\leq d$ characters in graph G such that q will have a matching path in G?" is $\mathcal{NP}$-complete for $|\Sigma| \geq 2$.*

Proof. The setup used in the proof of Theorem 4 can be trivially extended to prove the above claim. Alternatively, we can simplify the proof by using the query sequence $q = (t_0 t_1 \dots t_{n-1})^2$ since only one instance of the substring q_{sub} in q is needed for the subsequent arguments. This is because substitutions in the query sequence are not permitted. □

Using the above two results, we conclude that Hamming-distance based decision formulations of sequence to graph alignment problems are $\mathcal{NP}$-complete when substitutions are allowed in graph labels, for $|\Sigma| \geq 2$. In fact, it can be easily shown that $|\Sigma| \geq 2$ reflects a tight bound. Using $|\Sigma| = 1$, all the problem instances can be decided in polynomial time using straightforward application of standard graph algorithms.

5.4 Alignment using Edit Distance

We next show that edit distance based decision problems that permit changes in graph labels are $\mathcal{NP}$-complete if $|\Sigma| \geq 2$. Similar to our previous claims, allowing edits in the graph makes the sequence to graph alignment problem intractable. Proofs used for Hamming distance do not apply here as edits also permit insertions and deletions. Length of vertex labels can grow or shrink using insertion and deletion edits respectively.

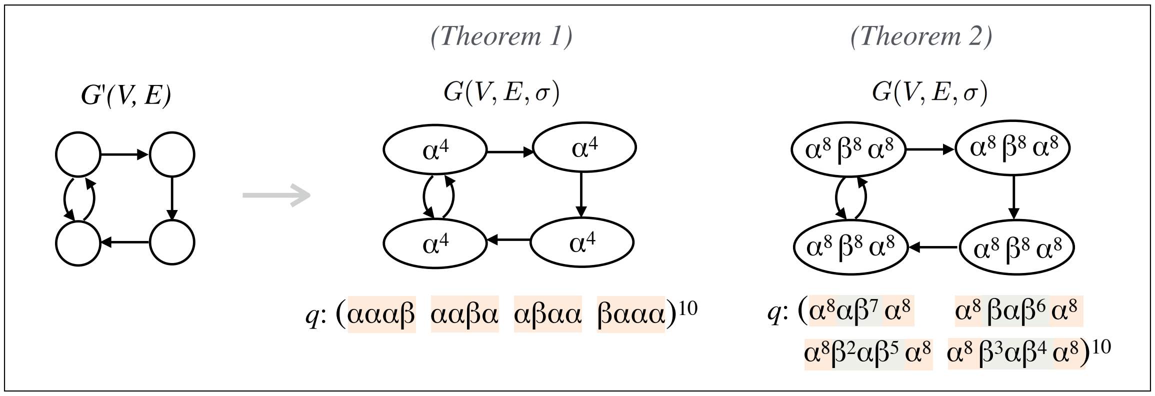

(Theorem 1)
(Theorem 2)
G'(V, E)
G(V, E, σ)
α⁴
α⁸ β⁸ α⁸
q: (αααβ ααβα αβαα βααα)¹⁰
q: (α⁸αβ⁷α⁸ α⁸βαβ⁶α⁸ α⁸β²αβ⁵α⁸ α⁸β³αβ⁴α⁸)¹⁰

Figure 5.3: The constructs used for reductions in proofs of Theorems 4 and 5.

Theorem 5. *The problem "Can we perform a total of $\leq d$ edits in graph G and query q so that q will match in G?" is $\mathcal{NP}$-complete for $|\Sigma| \geq 2$.*

Proof. Clearly the problem is in $\mathcal{NP}$. We again use the directed Hamiltonian cycle problem for reduction. Given an instance $G'(V, E)$ of the directed Hamiltonian cycle problem, we design an instance $G(V, E, \sigma)$ using $\Sigma = \{\alpha, \beta\}$. Let $n = |V|$. Label each vertex v in V using a sequence of $6n$ characters $\alpha^{2n}\beta^{2n}\alpha^{2n}$ (Figure 5.3). Let token t_i be a sequence of length $6n$: $\alpha^{2n}\ \beta^i\alpha\beta^{2n-1-i}\ \alpha^{2n}$. Using such tokens, we build a query sequence q of length $6n^2(2n+2)$ as $(t_0t_1 \dots t_{n-1})^{2n+2}$. We claim that a Hamiltonian cycle exists in $G'(V, E)$ if and only if we can match the sequence q to the graph $G(V, E, \sigma)$ using $\leq n$ total edits.

If there is a Hamiltonian cycle in $G'(V, E)$, we can follow the same loop in $G(V, E, \sigma)$ to align q. The alignment requires one substitution per vertex. To prove the converse, suppose query q matches graph $G(V, E, \sigma)$ after making a total of $\leq n$ edits in q and $G(V, E, \sigma)$. Consider the substring q_{sub} of q: $t_0t_1 \dots t_{n-1}t_0$. Note that there are $n + 1$ non-overlapping instances of q_{sub} in q, at least one of which must remain unchanged. Accordingly, the substring q_{sub} must match corrected $G(V, E, \sigma)$.

For the token t_i, let $k_i = \beta^i\alpha\beta^{2n-1-i}$ be its *kernel* sequence of length $2n$. It follows that $t_i = \alpha^{2n}k_i\alpha^{2n}$. We show that a kernel must be matched entirely within a vertex in $G(V, E, \sigma)$ using the following two arguments. First, since any vertex label cannot shrink from length $6n$ to $< 5n$, a kernel cannot be matched to an entire vertex after the edits. It implies that a kernel must match to ≤ 2 vertices. Second, if a kernel aligns across two vertices, $(2n-1)$ β's must be required in place of α's at the two vertex ends, thus requiring $> n$ edits. Therefore, a kernel can only be matched within a single vertex label. Finally, it is easy to observe that any vertex label after $\leq n$ edits cannot be matched to more than one kernel. When combining these arguments with the fact that all n consecutive kernels in q_{sub} are unique, we establish that the alignment path of q_{sub} must follow a Hamiltonian cycle in $G(V, E, \sigma)$. Accordingly, there is a Hamiltonian cycle in $G'(V, E)$. □

Corollary 2. *The problem "Can we perform $\leq d$ edits in graph G so that q will match in G?" is $\mathcal{NP}$-complete for $|\Sigma| \geq 2$.*

Proof. The setup used to prove Theorem 5 can be trivially extended to prove the above claim. □

It is straightforward to prove that other problem variants, e.g., with linear gap penalty or affine gap penalty scoring functions are at least as hard as the edit-distance based formulations. Therefore, the sequence to graph alignment problem remains $\mathcal{NP}$-complete even on constant sized alphabets for these classes of scoring functions also if changes are permitted in the graph. Finally, we note that all the above problems remain equally hard even for planar sequence graphs of max-degree 3, similar to the Hamiltonian cycle problem [166].

5.5 Permitting Edits to Sequence Alone

Allowing changes to the query sequence alone makes the problem polynomially solvable [107]. Table 5.1 summarizes the latest algorithmic developments for this problem while allowing edits to the query sequence alone. It is worth noting that the best algorithm [108] achieves the same time complexity $O(|V| + m|E|)$, as required for the easier problem of partial order alignment, i.e., sequence alignment to acyclic graphs [104, 20, 12].

We next discuss whether there can exist faster algorithms for solving the sequence to graph alignment problem, when permitting the edits to sequence alone. As noted by [109],

	Linear gap penalty		Affine gap penalty
	Edit distance	Arbitrary costs	
Amir *et al.* [107]	$O(m(\|V\|\log\|V\|+\|E\|))$	$O(m(\|V\|\log\|V\|+\|E\|))$	-
Navarro [20]	$O(m(\|V\|+\|E\|))$	-	-
HybridSpades [91]	$O(m(\|V\|\log(m\|V\|)+\|E\|))$	$O(m(\|V\|\log(m\|V\|)+\|E\|))$	-
V-ALIGN [167]	$O(m\|V\|\|E\|)$	$O(m\|V\|\|E\|)$	$O(m\|V\|\|E\|)$
Rautiainen and Marschall [109]	$O(\|V\|+m\|E\|)$	$O(m(\|V\|\log\|V\|+\|E\|))$	$O(m(\|V\|\log\|V\|+\|E\|))$
Jain *et al.* [108]	$O(\|V\|+m\|E\|)$	$O(\|V\|+m\|E\|)$	$O(\|V\|+m\|E\|)$

Table 5.1: Comparison of run-time complexity achieved by different algorithms for the sequence to graph alignment problem when changes are allowed in the query sequence alone. In this table, m denotes the query length, and V, E denote the vertex and edge sets in a graph with character-labeled vertices respectively.

the sequence to sequence alignment problem is a special case of the sequence to graph alignment problem because a sequence can be represented as a directed chain graph with character labels. As a result, existence of either $O(m^{1-\epsilon}|E|)$ or $O(m|E|^{1-\epsilon}), \epsilon > 0$ time algorithm for solving the sequence to graph alignment problem (for both acyclic or cyclic graphs) is unlikely because it would also yield a strongly sub-quadratic algorithm for solving the sequence to sequence alignment problem, further contradicting SETH [151]. Equi *et al.* [168, 169] prove that exact and approximate matching to graphs are equally hard problems under the SETH assumption. An implication of this result is that the sequence to graph alignment problem is unlikely to have a faster 'banded alignment' solution [170], for the problem variant where the count of edits allowed is an input parameter.

5.6 Summary

We prove that the sequence to graph alignment problem versions are $\mathcal{NP}$-complete when changes are allowed in the sequence graph, for any alphabet of size ≥ 2. The theoretical results presented in this chapter enhance the fundamental understanding of the problem. In the case of classic sequence to sequence comparison, the quadratic-time alignment algorithms can account for errors or mutation events in reference. As it turns out, we lose this privilege when reference is a graph.

CHAPTER 6
ACCELERATING LONG READ ALIGNMENT TO GRAPHS

Given a variation graph in the form of a directed acyclic string graph (DAG), solving the long read to graph alignment problem exactly using a sequential dynamic programming algorithm takes quadratic time in terms of the graph size and read length, making it difficult to scale to high throughput DNA sequencing data. In this chapter, we propose the first parallel algorithm for computing read to graph alignments that leverages multiple cores and single-instruction multiple-data (SIMD) operations. We take advantage of the available inter-task parallelism, and provide a novel blocked approach to compute the score matrix while ensuring high memory locality. Using a 48-core Intel Xeon Skylake processor, the proposed algorithm achieves peak performance of 317 billion cell updates per second (GCUPS), and demonstrates near linear weak and strong scaling on up to 48 cores. It delivers significant performance gains compared to existing algorithms, and results in run-time reduction from multiple days to three hours for the problem of optimally aligning high coverage long (PacBio/ONT) or short (Illumina) DNA reads to an MHC human variation graph containing 10 million vertices.

6.1 Preliminaries

6.1.1 Problem Formulation

The classic sequence to sequence alignment problems for approximate matching are typically classified as either global, semi-global or local alignment. These problems can be solved exactly using dynamic programming (e.g., using Needleman-Wunsh [172] or Smith-Waterman [173] algorithms). Given a scoring scheme to reward matches and penalize

This chapter interpolates material from a paper by the author [171]

mismatches, insertions and deletions, the alignment problems are formulated to compute alignments that achieve maximum score. Similarly, when aligning a query sequence to a variation graph, the problem is to identify the highest scoring alignment between the query sequence and any path in the graph.

In this work, we focus on sequence to DAG alignment problem in local mode, i.e., computing local regions of similarity [173]. The proposed parallelization algorithm in this paper generalizes to other alignment modes, but they are not discussed for brevity. Following previous works [115, 15, 118, 119, 114], we consider variation graph as a DAG $G(V, E, \sigma)$, where function σ assigns each vertex a character from the alphabet set $\Sigma = \{A, C, G, T\}$ describing DNA bases. Naturally, any path p in the graph spells a DNA sequence. Let $q \in \Sigma^*$ be a query sequence of length m.

Definition 6.1.1. Sequence to DAG Local Alignment Problem: Given a query sequence q and a DAG $G(V, E, \sigma)$, identify a path p in the DAG and a substring of $q : q[i..j]$ s.t. the optimal alignment score between $q[i..j]$ and the sequence specified by p is maximum over all possible choices for p, i, and j. In addition, report the corresponding alignment.

In cases when the query has multiple optimal alignments, we aim to output one of them. Although the problem definition includes a single query sequence for convenience, we are required to solve numerous instances of the problem, twice for each input sequence (counting both the complementary DNA strands) in the set of reads being mapped.

6.1.2 Sequential Algorithm

Sequence alignment to DAGs is computed using dynamic programming (DP), essentially by extending the Smith-Waterman algorithm to DAGs [20, 12]. Assume the DAG G is topologically sorted. Suppose $C_{i,j}$ denotes the highest score of an optimal alignment between any suffix of $q[1..i]$ and any path ending at vertex v_j. Then, a sequential $O(m(|V| + |E|))$ time algorithm follows from the recurrence below:

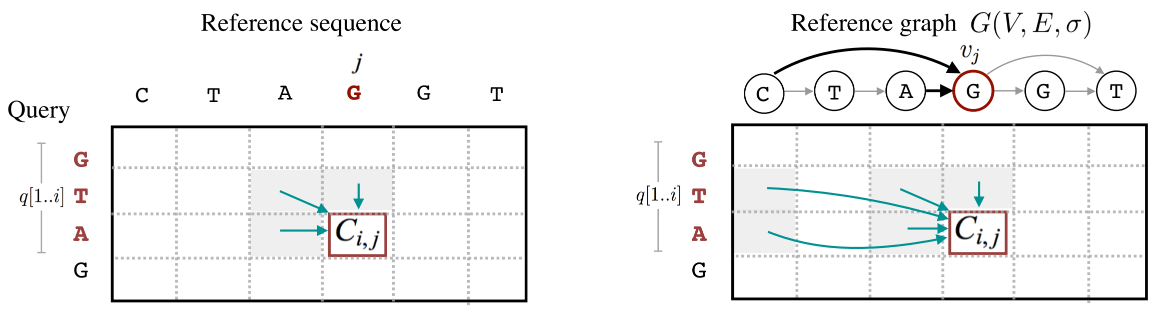


Figure 6.1: Example to illustrate difference between Smith-Waterman sequence to sequence alignment and sequence to DAG alignment procedures.

$$
\begin{aligned}
&C_{0,j} = 0 \\
&C_{i,j} = \max \begin{cases} 0 \\ \Delta_{i,j} \\ C_{i-1,k} + \Delta_{i,j} & \forall k : (v_k, v_j) \in E \\ C_{i,k} - \Delta_{ins} & \forall k : (v_k, v_j) \in E \\ C_{i-1,j} - \Delta_{del} \end{cases}
\end{aligned}
\tag{6.1}
$$

where $\Delta_{i,j}$ denotes the score of a match/mismatch, and Δ_{ins} and Δ_{del} denote the insertion and deletion penalties, respectively. The DP score matrix has a height of $m + 1$ and width of $|V|$. Note that score of each cell depends on cells in the previous row as well as cells to the left in the same row. Once the location of optimal alignment in the matrix is known, final base-to-base alignment is reported using a traceback procedure that follows the chain of decisions made in computing the $C_{i,j}$'s. Similar to the Smith-Waterman algorithm, the traceback begins at the highest scoring cell and proceeds until a cell with zero score is encountered. The two cells with the zero and the highest score denote the start and the end of an optimal alignment respectively.

6.1.3 Constraints on Design of Parallel Algorithm

The described sequential algorithm is similar to the Smith-Waterman algorithm, the only difference being that each vertex can now have multiple neighbor vertices instead of just one (Figure 6.1). This one difference, however, makes numerous parallelization strategies [174, 175, 176, 177] developed for accelerating the Smith-Waterman algorithm either inapplicable or inefficient for the sequence to DAG alignment problem. We list the challenges below:

1. Storing complete DP score matrix in memory is usually impossible with real input data. In the Smith-Waterman algorithm, score matrix can be computed either one column, row, or diagonal at a time. This is possible because storing one previous column (or row) is sufficient as the DP progresses. However, vertices in the variation graph can be connected to many (near and distant in topological order) predecessor vertices, leaving row-wise computation as the only choice.

2. Unlike the Smith-Waterman algorithm, count of arithmetic operations required to compute score for each cell $C_{i,j}$ in a row is not uniform, and depends on in-degree of vertex v_j. This further makes SIMD-based intra-task parallelization challenging.

We present a new inter-task based parallelization approach that takes into account the above constraints.

6.2 Proposed Parallel Algorithm

6.2.1 Graph Representation

Solving Recurrence (6.1) requires frequent access to graph vertices and edges. Therefore, it is important that we spend as few CPU cycles as possible to access graph information while computing scores. Variation graphs are highly sparse, in fact the edge to vertex ratio is typically close to one [178]. Therefore, we choose the standard 'compressed sparse row' (CSR) format. It allows constant time access to adjacency list of any vertex. In this format, we use three arrays: the first one of size $|E|$ for contiguous storage of adjacency list, another $(|V|+1)$-sized pointer array to mark start and end offsets for each vertex within the adjacency list, and the last array of size $|V|$ to store DNA character labels of each vertex.

6.2.2 A Three-Stage Algorithm

The proposed algorithm is designed to produce not only the optimal alignment scores, but also the base-to-base alignments corresponding to them. The base-to-base alignments are computed using a traceback procedure which requires access to the entire score matrix, or an appropriate section of it. In most practical cases, the sizes of the score matrices are too large to be able to completely store in memory. Therefore, we execute a three-stage algorithm to keep the memory-usage low. The first two stages of the algorithm are executed to identify starting and ending positions of the optimal alignments. In particular, the first stage *DP-fwd* computes ending position of an optimal alignment for each read by executing the DP to solve Recurrence (6.1). The second stage *DP-rev* solves the same DP in reverse direction, i.e., from bottom to top to locate the starting positions of the optimal alignments. Since score of a cell depends only on its current and previous row (Section 6.1.1), we only need to keep two rows in memory. Hence, the two stages use low memory. Finally, the third stage uses the starting and ending locations to recompute the corresponding section of the score matrix, and executes a traceback to report the base-to-base alignments (Section 6.2.4). This approach is similar to the one proposed by Huang *et al.* [179] to contain the memory usage while computing local alignment between two sequences.

It may seem that the first two stages are exactly symmetric, but there is a caveat. If there are multiple alignments with the best score, it may so happen that the second stage reports the starting point of an alignment which is different from the one obtained in the first stage. This can be avoided using a simple trick, which is to add an artificial score at the known alignment end coordinate during reverse DP [179]. Using this, we ensure that the two stages return the two ends of an optimal alignment.

The first DP-fwd stage returns optimal score, ending position of an optimal alignment, and optimally aligning strand of each input read. DP-rev stage only uses sequences corresponding to the optimally aligning read strands as its input. Figure 6.2 summarizes the role of all three stages in the algorithm. Note that both DP-fwd and DP-rev stages compute the entire DP matrix using the same recurrence relation, therefore designing a single parallel strategy suffices to accelerate them. We propose a parallel algorithm in the following section.

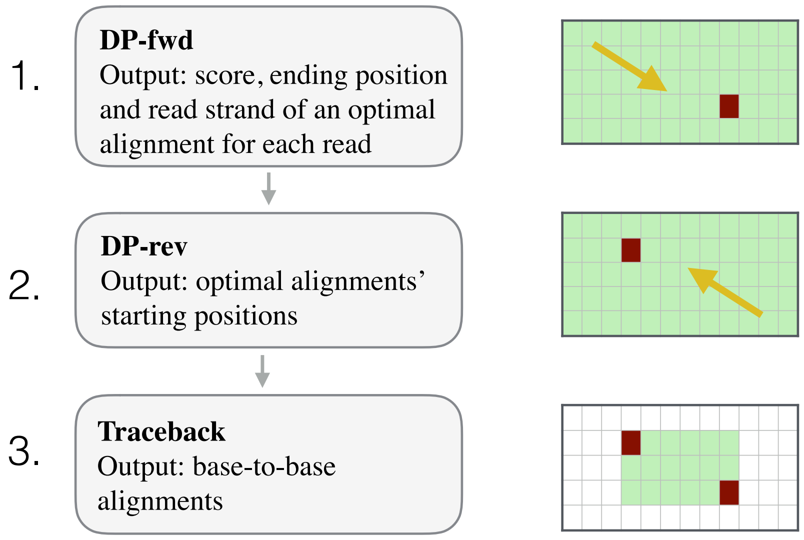

1.
DP-fwd
Output: score, ending position and read strand of an optimal alignment for each read
2.
DP-rev
Output: optimal alignments' starting positions
3.
Traceback
Output: base-to-base alignments

Figure 6.2: Role of the three stages used in our algorithm.

6.2.3 Parallel Computation of the Score Matrix

Prior to describing full details, we give an overview of the algorithm. Both DP-fwd and DP-rev stages compute the entire score matrix containing $(m + 1) \times |V|$ elements for each input sequence. Computing score matrices is highly compute-intensive, and consumes most of the time in sequential as well as our parallel algorithm. The proposed algorithm is inspired from previous optimization efforts targeted towards accelerating Smith-Waterman alignment using SIMD instructions [180, 181]. Alignment of a single sequence is called a task. We present an inter-task parallel algorithm to accelerate the matrix computation.

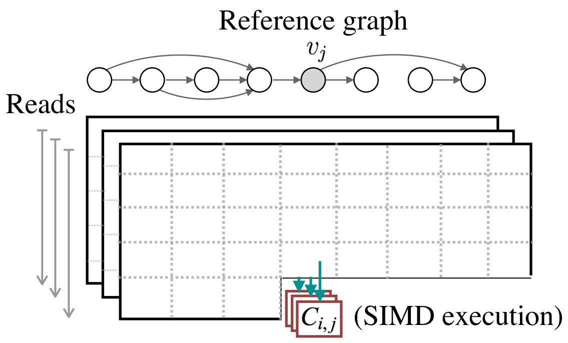
In other words, rather than parallelizing the alignment of a single read, the algorithm processes multiple reads simultaneously (Section 6.2.3). To compute each task, we can choose to follow a naive sequential algorithm, essentially computing the scores row by row. However, it turns out that traversing $O(|V|)$-sized row buffers repeatedly makes the algorithm memory-bound (Section 6.2.3). To address this issue, we subsequently introduce a new blocking strategy that leverages a domain-specific property of variation graphs, and enhances memory access locality.

Inter-task parallelism

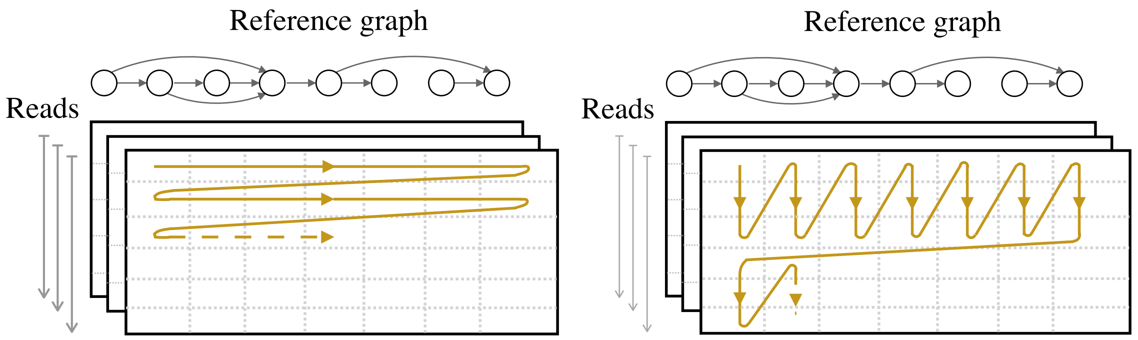
Our algorithm leverages inter-task parallelism by aligning multiple reads simultaneously. Below, we discuss how to make use of multiple threads and SIMD instructions to realize this efficiently:

Multi-threading We divide the input read set into batches that are individually scheduled to different threads. Because runtime to align reads of different lengths varies, we use the dynamic scheduling policy in OpenMP.

Vectorization Within each thread, we vectorize our implementation to process all reads in a batch simultaneously (Figure 6.3a). Count of reads in a single batch is set to SIMD width to keep all vector lanes busy. For instance, recent Intel® Xeon® Skylake processors support AVX512 integer instructions (512 bit vectors). Therefore, depending on the requirement of precision to compute scores (e.g., int8, int16 or int32), there is scope of 16-64x speedup using vectorization. For each batch of reads, we convert read characters from AoS to SoA format to ensure that we can load the read characters for one cell update using just one vector load instruction. Suppose in-degree of vertex v_j is δ_j, then computing $C_{i,j}$ across all vector lanes uses $10 + 4\delta_j$ vector operations (using cmpeq, blend, set, max, and add; not counting load and store) in our implementation. Finally, because read lengths within a batch of reads can vary, we pad shorter reads with dummy characters to obtain uniform lengths.



(a) Vectorization by using inter-task parallelism.



(b) Modifying sequential procedure of each task to enable blocking.

Load balancing Lengths of long reads tend to vary significantly in a single sequencing run. Therefore, to avoid wasteful work due to padding, we sort the complete input read set by their length before dividing them into batches. In this way, variation in the lengths of adjacent set of reads is reduced. The sorting is done in decreasing order of read lengths to make sure that processing of longer reads is initiated first.

Optimizing Precision Operating at lower precision (e.g., int16 vs. int32) yields higher scope for parallelism using vector units. Note that the product of maximum input read length and match parameter is an upper bound on the score value in all DP matrices. Based on this value, the algorithm decides the required precision at runtime. Besides maximum score, we also keep track of its column and row position during DP computation. The row positions use the same precision as score values because they are bounded by maximum read length. The column positions, however, can range from 1 to $|V|$, therefore they are always operated using int32 precision.

In the above inter-task parallelization scheme, each individual task can still be executed using a naive sequential algorithm, essentially computing the scores row by row. Working independently on individual reads gives us an advantage that there is no synchronization needed across threads or vector-units, which favors both performance and programmability. For each task, we need to maintain two score buffers for the current and previous rows. The two buffers can be used inter-changeably, i.e., one for reading previous row and one for computing current row. However, each of them uses $O(|V|)$ memory, and does not fit in cache. As shown later in results (Section 6.3), this issue limits scalability by making the algorithm memory bandwidth bound due to frequent access to DRAM. We next describe a blocked computation strategy which modifies the sequential procedure of each task to resolve this issue.

Improving Memory Locality using Blocked Computation

We propose a blocked algorithm to compute the score matrix which significantly reduces the average count of reads and writes to DRAM. The first step is to increase the granularity by computing multiple rows rather than one row in a single horizontal sweep (Figure 6.3b). For a subsequent horizontal sweep, we just need to preserve scores associated with the last row of the current block. Next, while processing multiple rows in a single horizontal sweep, we modify memory access pattern to ensure majority of accesses are cached. This modification leverages a domain-specific topological property of the graphs.

In the variation graphs, we find that the fraction of vertices connected to 'distant' vertices in the topological order is significantly small. More formally, let B_{width} be an appropriately chosen distance threshold, then the number of vertices in set $V' = \{v_i : (v_i, v_j) \in E, j - i \geq B_{width}\}$ is much smaller compared to $|V|$, for even small values of B_{width}. In our implementation for instance, we selected $B_{width} = 8$ as this value was appropriate for various graphs tested empirically. This particular graph property is attributed to the fact that $> 99.9\%$ of genetic variations in a human genome are either single nucleotide substitutions or small insertions/deletions [19]. Such genetic variants mostly appear as small bubbles in the variation graphs. Large structural variants which would result in connecting farther vertices occur at much less frequency. As a result, majority of vertices in variation graphs are expected to have all their neighbors in near vicinity in the topological order. We next show how to leverage this property to improve the memory access pattern.

In the blocked-procedure, suppose the count of rows processed in a single horizontal sweep is denoted as B_{height}. We use a small circular buffer of size $B_{height} \cdot B_{width}$ for temporary storage of scores while processing the B_{height} rows (see Figure 6.3). Using this buffer, score of a vertex v_i is available while computing score of v_j whenever $j - i < B_{width}$. This modification ensures that majority of DRAM accesses are cached. To manage the scores of 'long-hopped' vertices $\in V'$, we use a separate buffer to save their scores for subsequent access. This buffer is also small and manageable because $|V'| \ll |V|$. In our implementation, we set B_{height} and B_{width} to 16 and 8 respectively as these values resulted in the least memory latency and best performance during execution (further discussed later in Results section).

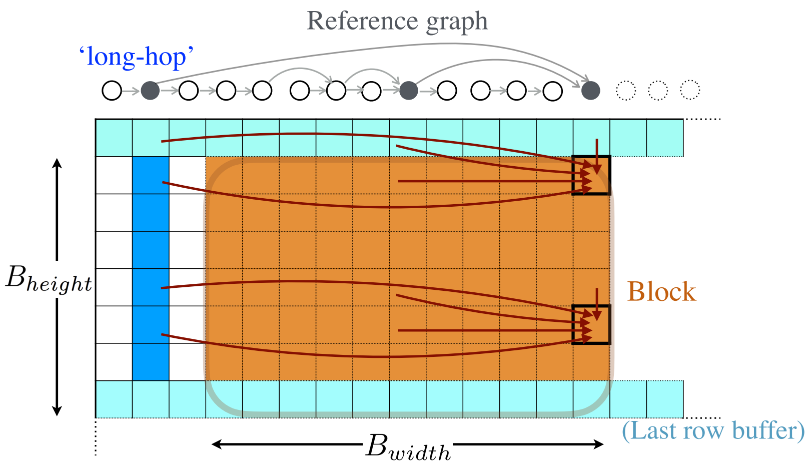


Figure 6.3: Visualizing a section of DP matrix to illustrate different memory accesses occurring using the blocked approach. Blocking improves memory locality because majority of accesses (red arrows) occur within the circular $B_{width} \times B_{height}$ block buffer.

6.2.4 Computing Base-to-Base Alignments

The first two stages output the starting and ending alignment coordinates of each read. In the third stage, we recompute the enclosed section of the score matrix to execute the final traceback. Note that the coordinates of the optimal alignments of each read vary. Therefore, different matrix columns need to be computed for each read. This implies that we cannot re-use the inter-task SIMD parallelism that is developed for the first two stages. However, each of these tasks can still be executed independently using multi-threading. In this stage, we preserve the computed scores in memory. Because read alignments can hop through long edges (w.r.t. the topological vertex order) in the graph, even the score sub-matrices can be large. To reduce memory usage by a factor of four, we save the matrix using differences between adjacent rows instead. This helps because maximum absolute value of the differences ($C_{i,j} - C_{i-1,j}$) is bounded by the sum of match and gap parameters [37], and can be saved as 8-bit integer values. We note that it is possible to extend alternate approaches to graphs such as Hirschberg's divide and conquer algorithm [182] or external memory algorithms [183] which use less memory, but they require more computation time in practice. Finally, the algorithm finishes after computing base-to-base alignments by tracing the paths associated with the respective optimal alignments of the input reads.

6.3 Results

We refer to the C++ implementation of our **pa**rallel **s**equence to **g**raph **al**ignment algorithm as PaSGAL. This section provides details of the experimental setup, performance characteristics of PaSGAL, and advantages of the proposed optimizations. Subsequently, we demonstrate significant performance gains compared to existing read to graph aligners.

Table 6.1: Summary of input graphs and read sets used for evaluation. First three columns show input graphs and their sizes while the remaining columns show characteristics of simulated read sets.

Reference graph	# vertices	# edges	Read set	Mean length (bp)	Coverage	# reads
LRC	1 099 856	1 144 498	L1	100	30x	317 606
			L2	9882	30x	3214
			L3	24 871	30x	1277
MHC1	5 138 362	5 318 019	M1	100	10x	497 046
MHC2	10 618 991	10 698 615	M2	10 060	10x	4941
			M3	24 877	10x	1998

6.3.1 Experimental Setup

Data-sets

We used three input graphs and multiple read sets for evaluating PaSGAL (Table 6.1).

Variation Graphs Leukocyte Receptor Complex (LRC) and Major Histocompatibility Complex (MHC) regions are among the most diverse variant hot-spots, spanning about 1.06 Mbp and 4.97 Mbp of the human genome [15, 178], respectively. We leveraged existing tools to build variation graphs using real public data. The first two graphs, labeled as LRC and MHC1 were built using the vg toolkit [16]. We supplied human genome (GRCh38) and variant files from 1000 genome project [19] as input to vg. The variant files constitute small-scale variations (≤ 50 bp) in genomes of 2504 individuals. To also evaluate using more complex graphs, we used a second MHC variation graph (MHC2) from a previous study [84], which also includes large structural variations.

Read Sets Multiple sequencing read sets of different characteristics were simulated from the LRC (L1-L3) and MHC (M1-M3) regions in the human genome (GRCh38) (Table 6.1). These read sets are representative of outputs produced using different technologies, lengths, and error characteristics. We used mason2 [184] and pbsim [185] tools to simulate single-end short Illumina and long noisy PacBio reads respectively. Sampling was done at 30x and

10x coverage from the LRC and MHC regions respectively. Because Illumina sequencers produce fixed-length short reads, L1 and M1 data sets contain uniform length reads of size 100 bp. On the contrary, long read technologies produce noisy reads of variable lengths; therefore L2-M2 and L3-M3 were sampled using mean read length 10 Kbp and 25 Kbp respectively. The minimum length, maximum length and mean error-rate parameters were set to 1 Kbp, 30 Kbp and 15% respectively during simulation. The match score, and mismatch, insertion, and deletion penalties were all set to 1. For each input read, the alignment program considers both complementary DNA strands, and outputs an optimal alignment to the input graph.

Hardware/Software Description

Unless otherwise mentioned, all our experiments used a single node consisting of Intel® Xeon® Platinum 8160 (Skylake) processor in Stampede2 cluster located at Texas Advanced Computing Center. Each node is equipped with 192 GB RAM and two sockets, each containing 24 cores. Peak memory bandwidth of a node is 220 GB/s spread over two NUMA domains, one on each socket. These nodes operate at base frequency of 2.1 GHz, although frequency can vary due to turbo boost feature. Skylake platforms support AVX512 (512 bit) vector processing for 8-bit, 16-bit, and 32-bit integer operations.

PaSGAL was compiled using Intel® compiler (v18.0.2). We used OpenMP for multi-threading and hand-written SIMD intrinsics for vectorization.

Measurements

In all experiments, we measured runtime of the main alignment routine, and ignored pre-processing time, i.e., the time spent to load the input and converting graph into CSR format (Section 6.2.1). Loading and pre-processing the input in PaSGAL took an insignificant fraction of time ($< 1\%$). For all multi-threaded executions, we mapped a single thread to a single physical core.

6.3.2 Performance Results

Time to Solution

We first show the time to solution using PaSGAL for all nine input combinations in Table 6.2 using 48 threads. PaSGAL makes efficient use of multiple cores as well as vector units within each core to achieve fast time to solution. Using PaSGAL, we aligned 30x coverage read sets (L1-L3) to the LRC graph in < 15 minutes. For the larger MHC1 graph, 10x coverage read sets (M1-M3) were aligned in < 1.5 hours. Finally, the largest graph MHC2 took the longest time of 1.5 to 3.5 hours.

Table 6.2: Performance evaluation of PaSGAL using all the test data sets.

Reference graph	LRC			MHC1			MHC2		
Read set	L1	L2	L3	M1	M2	M3	M1	M2	M3
Total time (s)	358	650	824	2592	4325	4963	5481	9218	11 580
DP-fwd time (%)	62	56	49	63	62	57	62	60	51
DP-rev time (%)	36	34	31	37	36	38	37	36	35
Traceback time (%)	2	10	20	1	2	5	1	5	14
Memory usage (GB)	7	8	34	34	33	34	88	88	142

We also show the performance achieved for score matrix computation as billion cell updates per second (GCUPS), the standard metric to evaluate Smith-Waterman algorithms (Figure 6.4). The GCUPS metric indicates the count of score matrix cells that are computed in a second, therefore higher is better. PaSGAL achieved peak performance of 317 GCUPS using the LRC/L1 input. To our knowledge, this is the highest performance achieved till date when aligning sequences to DAGs. Note that short read alignment (L1 and M1) was consistently fastest for all the three graphs because PaSGAL selects the required SIMD precision level based on the input read length (Section 6.2.3). For L1 and M1, 8-bit precision is sufficient as read length is only 100. On the other hand, the other read sets require 16-bit precision.

Large count of reads in an input set ensures that all the vector lanes do useful work, thus

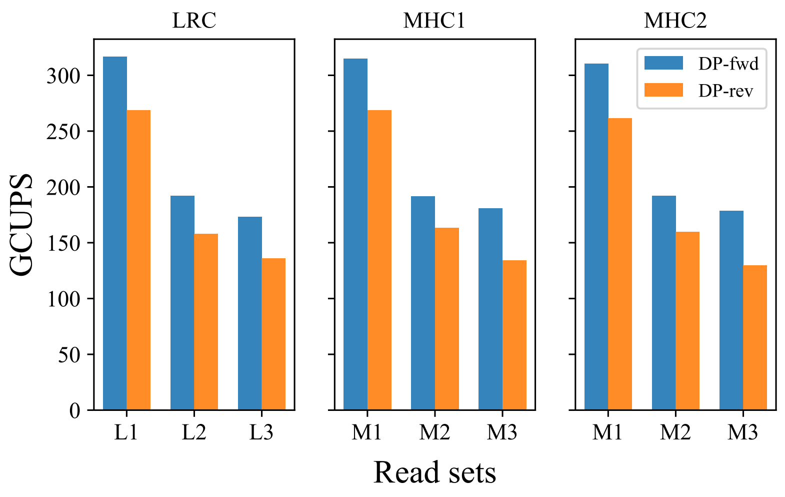


Figure 6.4: Performance achieved during DP-fwd and DP-rev stages of PaSGAL measured in billions of cell updates per second (GCUPS).

DP-fwd and DP-rev stages of the algorithm achieve high efficiency. We also note that the GCUPS performance numbers for forward DP are relatively higher compared to reverse DP. Even though the recurrences computed in DP-rev and DP-fwd stages are identical, DP-rev requires additional logic to ensure that it reports the end point of the same optimal alignment that was identified by the DP-fwd stage for each read (Section 6.2.2).

Finally, we also include a break-down of the total execution time into time spent in the three individual stages DP-fwd, DP-rev, and traceback (Table 6.2). Even though the traceback phase is not vectorized, time to compute the two end points of optimal alignments using the DP-fwd and DP-rev stages still took majority of the time. This is because during the traceback stage, we are only required to compute a small portion of the score matrix.

Load balance

Unlike short reads, long read lengths tend to vary significantly, therefore splitting the work equally can be challenging in an inter-task parallel approach. In PaSGAL, we address this issue by sorting the reads by their lengths and adopting dynamic scheduling policy (Section 6.2.3). We measured individual timings on 48 threads for all three stages of the algorithm. In Figure 6.5, we report 'load imbalance ratio' which is equal to maximum time divided by average time on all threads. Ideally, this ratio should equal one. We observe that this ratio is below 1.5 for all data sets. Better load balance is achieved for short read sets (L1, M1) relative to long reads, owing to their uniform lengths. Further, better load balance is observed for DP-fwd stage relative to DP-rev stage. DP-fwd stage processes both strands of DNA sequences, where as DP-rev only processes one, reducing its input size by half (Section 6.2.2). The sorting approach is inherently more effective with higher read counts (Section 6.2.3).

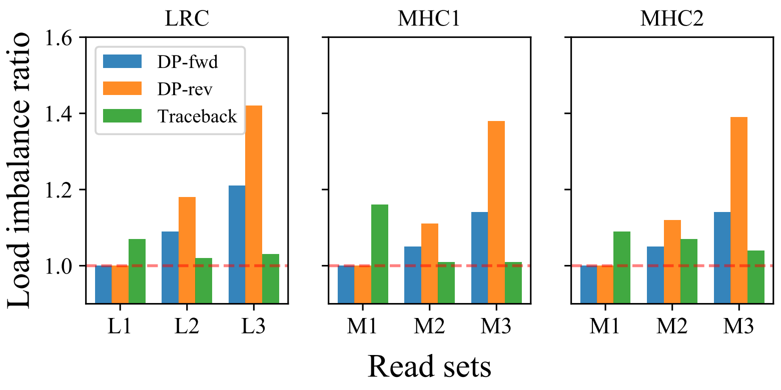


Figure 6.5: Load imbalance observed in PaSGAL using all test data sets while using 48 threads.

Benefits of Proposed Optimizations

We next verified performance gains from the two optimizations – blocked strategy (Section 6.2.3) and vectorization (Section 6.2.3) used to accelerate the score matrix computation in DP-fwd and DP-rev stages, which account for majority of the time spent. We also collected critical performance counter numbers (e.g., memory latency, bandwidth etc.) using Intel® VTune® Amplifier tool. VTune profiling requires the experiments to be short to avoid counter overflow. Therefore, we used a small simulated short-read set from the LRC region, adding to 12,288 reads of length 96 bp, and aligned them to the LRC graph.

Blocked approach Goal of the blocked approach is to remove memory access bottlenecks due to frequent access to DRAM (Section 6.2.3). PaSGAL uses default block width (B_{width})

and height (B_{height}) values of 8 and 16 respectively. Using 48 threads and vectorized execution, we analyzed benefit of this approach by manipulating block dimensions from 1 to 32 (see Table 6.3). Note that using block height of one is equivalent to computing the score matrix one row at a time. The results support that the blocked approach succeeds in improving runtime by reducing memory latency, LLC (last-level cache) misses and DRAM bandwidth. This is because majority of reads and writes occur in the small $B_{width} \times B_{height}$-sized block. Each cell in the block is a SIMD register of size 64 bytes, therefore total storage memory equals 8 KB, making it small enough to fit in L1 cache memory (capacity=32 KB) of all cores.

Table 6.3: Significance of using blocked approach.

(a) Performance analysis while increasing block height from 1 to 32, while keeping width fixed to the default value of 8.

Block height	1	4	8	16	32
Time (s)	40.9	11.1	8.4	7.8	8.0
Avg. memory latency (cycle)	27	8	7	7	8
LLC miss count ($\times 10^6$)	632.3	14.4	1.4	0	3.6
Avg. DRAM bandwidth (GB/s)	189.3	176.3	125.1	71.7	35.5

(b) Performance analysis while increasing block width from 1 to 32, and keeping height fixed to the default value of 16.

Block width	1	4	8	16	32
Time (s)	32.5	7.9	7.8	7.8	7.9
Avg. memory latency (cycle)	13	7	7	7	7
LLC miss count ($\times 10^6$)	44.6	0.5	0	0	0
Avg. DRAM bandwidth (GB/s)	177.2	69.8	69.6	70.8	70.6

Vectorization Using Figure 6.6, we show the benefit of vectorization in PaSGAL by comparing it to our sequential scalar code, which computes score matrix sequentially in a row-wise manner. To isolate the benefit of vectorization and the enhanced memory locality, we executed this experiment using single thread only. Here we also experimented with three precision levels for integer instructions (8-bit, 16-bit and 32-bit). The plot shows that vectorization coupled with improved memory locality resulted in up to 58.7x speedup compared to the scalar code.

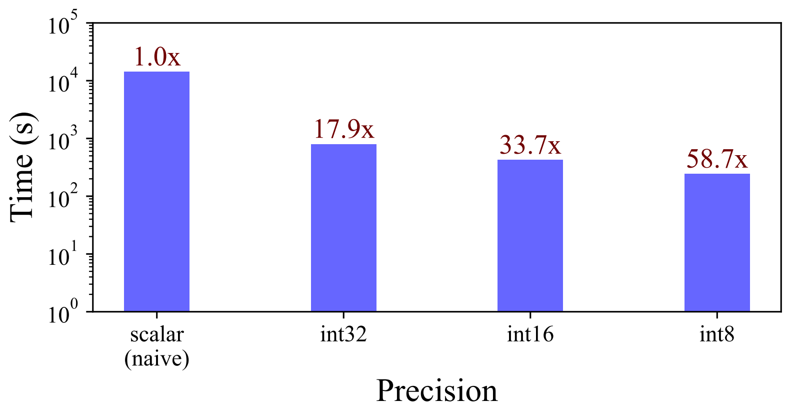


Figure 6.6: Performance improvement in PaSGAL obtained using improved memory locality and vectorization supporting different precision levels. This experiment was executed using a single thread.

Scalability

The combination of our efficient vectorization strategy and blocking algorithm drive the high-performance in PaSGAL. These optimizations also helped us achieve near-linear strong scaling and weak scaling results going from 1 to 48 cores. The Skylake processors in Stampede2 use turbo technology, thus making scaling studies less reliable. Therefore, we conducted our scaling experiments on a different Skylake CPU (Intel® Xeon® Platinum 8180) with turbo technology disabled. In the two scaling experiments, we aligned L1-L3 read sets to the LRC graph, and report scaling behavior of the DP-fwd stage and overall runtime separately.

Strong scaling Using the strong scaling experiment, we study the ability of our algorithm to compute the alignment problem faster with increasing core counts. Using 48 cores, the total runtime was reduced by 47x, 41x and 38x for the read sets L1, L2 and L3 respectively (Figure 6.7). Speedup factors for the DP-fwd stage were roughly similar- 47x, 43x and 40x respectively. Short read set (L1), in particular delivered close to ideal scaling behavior because of following two reasons- a) read count in L1 is very high, thus all SIMD lanes were busy doing useful work throughout the execution, and b) read lengths are uniform, therefore there was no overhead from load imbalance (Section 6.3.2). We also evaluated the scaling behavior of the DP-fwd stage by manipulating the block dimensions (Figure 6.8). Results reveal that the blocked approach is critical for the near-linear speedups achieved.

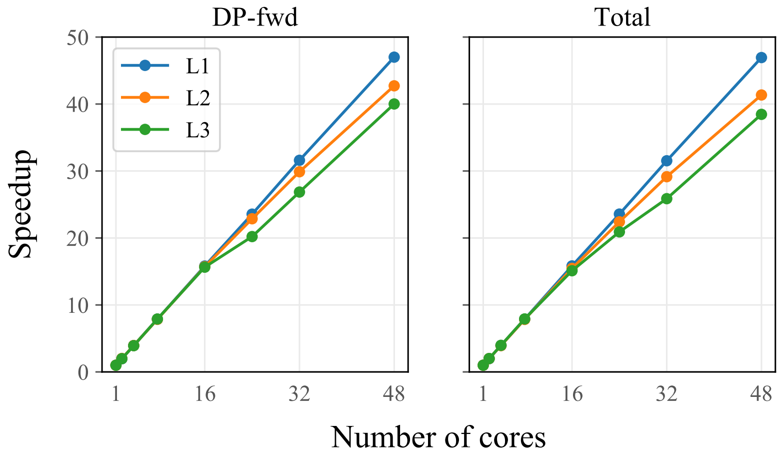


Figure 6.7: Strong scaling: Speedup achieved using PaSGAL with increasing core count relative to its single-core execution time. Left plot shows speedups achieved for the DP-fwd stage whereas right plot shows the overall speedup.

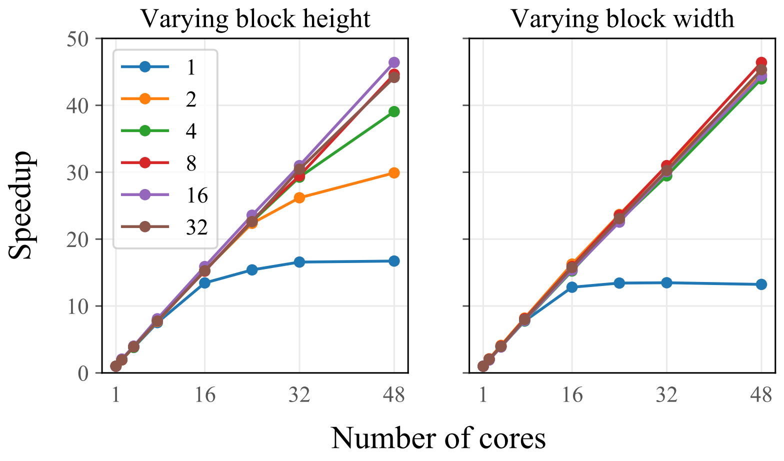


Figure 6.8: Illustration of how scaling behavior varies using different block sizes. Default values for block width and height were set to 8 and 16 respectively. We aligned the 96-bp short read set (Section 6.3.2) to the LRC graph in this experiment.

Weak scaling In the weak scaling experiment, we maintained size of input read set proportional to core count. This metric measures the ability of an algorithm to handle larger input sizes given more resources. Therefore, an ideal weak scaling behavior translates to constant execution time regardless of core count. To conduct this experiment using 1 to 48 cores, we re-simulated read sets with coverage proportional to core counts (i.e., 30x for 48 cores, 15x for 24 cores etc.). Results show that nearly uniform runtime was achieved going from 1 to 48 cores (Figure 6.9).

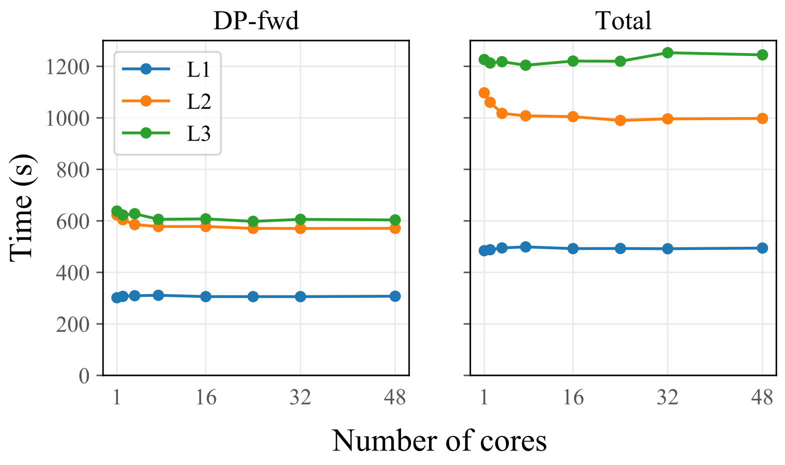


Figure 6.9: Weak scaling: PaSGAL's execution time remains nearly uniform with larger input data and proportionally increased core counts.

6.3.3 Comparison with Previous Algorithms

We compared PaSGAL against two recently published tools for sequence alignment to variation graphs – Graphaligner (commit:241565c) [120] and vg (v1.9.0-196) [16]. The software vg supports both exact and heuristic alignment modes. For convenience, we refer to its exact implementation as *vg-exact*, and heuristic implementation as *vg-heuristic*. We find that PaSGAL achieves up to 10x and 25x speedup against Graphaligner and vg-exact respectively, while using the lowest amount of memory. Compared to the vg-heuristic algorithm, we observe significant benefit in output quality. The comparisons against the exact and heuristic methods are discussed below separately.

Comparison With Exact Algorithms

Graphaligner uses bit-level parallelism to compute edit distance between input reads and graph. The algorithm outputs edit distance scores, therefore we compared its runtime against the equivalent DP-fwd stage of our algorithm. We also utilized the sequential implementation available in the Graphaligner repository as our sequential baseline. vg-exact extends Farrar's intra-task SIMD algorithm [121] to DAGs. It uses SSE (128-bit) intrinsics to accelerate the computation, and reports both optimal alignment scores and base-to-base alignment. As such, we compared its runtime against the total execution time of PaSGAL. Spoa [94], like vg-exact, also uses a intra-task SIMD parallelization algorithm for sequence alignment to DAGs, however, we could not compare against it because its implementation is designed to compute multiple sequence alignment for a different application. Both Graphaligner and vg do not support multi-threading, therefore, we used a single thread for a fair comparison. To allow all the experiments to finish in reasonable time, we re-simulated six read sets: L1′-L3′, M1′-M3′ with 0.5x coverage by following the exact same procedure as before (Section 6.3.1).

We show the speedups achieved using PaSGAL when compared to the other algorithms in Table 6.4. The speedups ranged from 40-98x, 3-11x and 13-25x when compared to the sequential baseline, Graphaligner and vg-exact, respectively. In three out of the six runs, vg-exact ran out of memory because it processes the DP column-wise and allocates the complete DP matrix in memory during the alignment. vg-exact supports affine gap penalty, which we plan to support in future versions of PaSGAL. Besides being fastest on a single-core, we conclude that PaSGAL also uses the lowest memory among all the algorithms.

Our previous results validate that PaSGAL supports efficient scalability using multiple cores (Section 6.3.2), whereas current exact methods use single-thread only. Based on the observed numbers, it will take other algorithms multiple days to process the 10x coverage MHC read sets (M1-M3), which PaSGAL does in about three hours or less (Table 6.2). Overall, PaSGAL provides significant advantage over existing exact algorithms in terms of

Table 6.4: Comparison with other exact algorithms.

(a) Runtime improvement achieved using PaSGAL relative to a sequential implementation, Graphaligner and vg-exact using single thread execution.

Reference graph	LRC			MHC1		
Read set	L1′	L2′	L3′	M1′	M2′	M3′
vs. sequential	94.9x	50.3x	41.1x	98.3x	56.3x	51.7x
vs. Graphaligner	10.7x	4.2x	3.0x	10.0x	3.7x	3.7x
vs. vg-exact	25.3x	13.3x	-	23.3x	-	-

(b) Peak memory-usage of all exact algorithms.

Reference graph	LRC			MHC1		
Read set	L1′	L2′	L3′	M1′	M2′	M3′
PaSGAL (GB)	0.2	0.3	0.8	0.9	0.9	0.9
sequential (GB)	1.0	1.0	1.0	1.4	1.4	1.4
Graphaligner (GB)	1.1	1.1	1.1	2.0	2.0	2.0
vg-exact (GB)	0.7	108.8	-	2.9	-	-

its ability to process high throughput read sets and larger graphs for biological and clinical applications.

Comparison with Seed-and-Extend Based Heuristic Algorithm

PaSGAL being an exact aligner, guarantees to produce an optimal alignment, irrespective of graph topology and sequence characteristics. Heuristic algorithms accelerate the mapping process by avoiding a full-scale DP when mapping a read. The seed-and-extend based heuristics identify local graph regions using either maximal or fixed length exact matches, and validate the matches using an extension step. Although this approach can be significantly faster, both seed-computation and extension stages are still challenging to execute in complex regions of graphs, and when the read lengths are long.

We ran vg-heuristic with default parameter settings using the L1′-L3′ read sets, and found it to be orders of magnitude (28-56x) faster than our exact algorithm. Next, we looked at the output accuracy. Since we executed both tools with the same scoring parameters, we compare the optimal alignment scores from PaSGAL against the scores computed by vg-heuristic (Table 6.5). We note that a large fraction of output scores reported by vg-heuristic are sub-optimal for L2′ and M2′ read sets. It failed to produce reasonable output for L3′ indicating that the algorithm is suited for short reads only.

Table 6.5: Accuracy evaluation of vg-heuristic algorithm for short and long read data sets. We compare its alignment score against the optimal score computed by PaSGAL.

Read set	L1′	L2′	L3′
Fraction of alignments with $> 5\%$ diff. from optimal score (%)	0.04	24.53	100
Fraction of alignments with $> 20\%$ diff. from optimal score (%)	0.00	9.40	100

6.4 Summary

In this chapter, we presented an inter-task based parallel algorithm PaSGAL to accelerate alignment of sequences to DAGs. Although conceptually similar to the classic Smith-Waterman problem which admits easy parallelization, significant variability in the number and structure of dependencies in the dynamic programming table make parallelization of alignment to DAGs quite challenging. Given an input set of reads and a variation graph, PaSGAL outputs optimal alignment scores and base-to-base alignments, a requirement for downstream biological analysis. To the best of our knowledge, it is the first parallel algorithm for solving this problem that fully utilizes modern architectures by leveraging multiple cores and wide SIMD width. To achieve these goals, we presented a three-stage algorithm and several optimizations to maximize integer operations per second. As a result, we are able to compute alignments of high-coverage long or short read sets to large variation graphs associated with clinically important human genome segments in the order of few minutes or hours, which was not feasible with prior algorithms. There is plenty of evidence in recent scientific literature that justifies the utility of variation graphs as a reference for studying genetic variants. The scalable and exact approach presented in this chapter constitutes a useful step towards fully realizing the potential of graph-based references for accurate genotyping.

CHAPTER 7

CONCLUSIONS AND FUTURE DIRECTIONS

As the pace of whole-genome sequencing continues to increase, faster practical algorithms and theoretical advances will be critical for biological data analyses. The presented research is a step in this direction. We propose new scalable algorithms for long read mapping to a large collection of reference genomes, and variation graphs. The proposed algorithms build on top of classic computational techniques such as sketching using MinHash [40] and winnowing [127, 36], plane sweep from computational geometry [135], and string pattern matching to hypertext [107, 20].

Using a fast alignment-free formulation coupled with a space-efficient indexing strategy, Mashmap is the first long read mapper to demonstrate scaling to the entire NCBI database. We further generalize the algorithm to compute gapped mappings, and show its utility for mapping ultra-long nanopore reads. We find that the solutions developed in the context of long read mapping are also adaptable to other related biological applications, such as computing whole-genome comparisons, detecting duplications in the genomes, long read-based metagenomics classification, etc. Accordingly, we extend Mashmap to accelerate the computation of homology maps and whole-genome distances (ANI). The computational efficiency and sensitivity guarantees allow us to demonstrate its applicability in addressing fundamental biological questions. For instance, we recovered all long duplications in the human genome using Mashmap2, and found that a significant 10.3% fraction of the human genome comprises of ≥ 1 Kbp duplications, with potential to uncover novel segmental duplications. In addition, we performed 8 billion pairwise ANI computations to conclude that $\geq 95\%$ ANI criterion reflects the same named species with a recall and precision of $\geq 98.5\%$.

We also shed light on the computational complexity of long read alignment to a graph-based reference. In particular, we prove that alignment to general graphs is $\mathcal{NP}$-complete unless edits are restricted to reads. Subsequently, we narrow down our focus on long read alignment to variation graphs. We take advantage of the domain-specific properties of these graphs, and develop a new parallel algorithm PaSGAL capable of producing optimal alignments of high coverage long read sets to medically relevant variation graphs derived from the human population. The algorithms presented in this dissertation are available as open-source software using the following links:

- Mashmap, Mashmap2: `https://github.com/marbl/MashMap`
- FastANI: `https://github.com/ParBLiSS/FastANI`
- PaSGAL: `https://github.com/ParBLiSS/PaSGAL`

7.1 Open Problems

Several interesting future directions are highlighted below that could be worthy of further investigation based on the results attained in this thesis.

Towards better error models for alignment-free mapping We make a few assumptions to model alignment edits (or errors) in query sequences while correlating the Jaccard similarity of k-mer sets to alignment quality (Section 2.1). The crucial assumptions are: i) k-mers in the query being mapped are unique, ii) the alignment edits occur independently, and iii) the edits follow a Poisson distribution. We rely on these to establish the sensitivity guarantees of the algorithm. While the model should be reasonable to account for errors in long reads, there might be instances where these assumptions fall short; for example, simple repeats may include repeated occurrence of k-mers, or alignment edits occurring as long contiguous insertion or deletions due to structural variations. Future research to improve the Mashmap model should be directed towards using alternate k-mer statistics such as weighted Jaccard similarity to account for multiplicity of k-mers [143], and modeling the structural variations separately [68]. These modifications will likely improve accuracy in practice.

Repeat handling and specificity of the alignment-free model The Mashmap model allows us to quantify its sensitivity guarantees, implying that the alignments with desired minimum length and identity are reported with high probability. This analysis is useful as we are able to auto-tune the parameters to guarantee sensitivity in the high 90s. But the model lacks theoretical guarantees on specificity. Currently it is not robust to prune mappings which fall short of meeting the criterion by small margins. This becomes critical when aligning to a highly repetitive reference genome (Section 3.2.2). In the context of computing duplications in the human genome, we address this by using alignment to prune the false positives. However, it is desirable to have a fast alignment-free solution with specificity guarantees in the future.

Locality sensitive hashing for graphs The runtime and memory usage of Mashmap grows linearly with the database size. As the genomic databases are expected to grow much faster than memory devices, future research should be aimed towards sub-linear alignment-free algorithms. We expect that majority of new data will bear high redundancy (e.g., strains within species). Leveraging graph-based references is a promising approach for compact representations of such data. Accordingly, it is tempting to think of whether the Jaccard similarity-based formulation, and its estimation using MinHash, can be extended to graphs. To achieve this, one possibility is to perform minimizer sampling along the chains of a graph, while preserving its junction vertices. Similar to Mashmap, we can target paths on graphs that yield sufficient Jaccard similarity of k-mers. The biggest challenge here is to deal with all possible paths in the graph, the count of which can be large when aligning long reads. An efficient algorithm for this problem will not only be useful for mapping reads to graphs, but also for zooming into an assembly graph to query and analyze unassembled genes [186, 187].

Extending complexity results for special graph instances The alignment problem for sequence graphs is a rich area with several unsolved problems. For the intractable problem variants when edits are allowed in graphs, development of practically fast or polynomial-time approximation algorithms are fertile grounds for future research. The results obtained in this research hold for general string labeled graphs. It remains open whether the problems remain hard for alignment to specific graph types such as de Bruijn graphs [9], Wheeler graphs [188] and string graphs [10].

Extending PaSGAL for wider utility Besides pan-genomics, our exact parallel sequence to graph alignment algorithm PaSGAL can be useful for other applications that benefit from sequence to DAG alignment, e.g., sequence alignment to splicing graphs in transcriptomics [11] and antibiotic resistance profiling [87]. Utility of PaSGAL can be further expanded by development of intra-task algorithms for use-cases with small count of query sequences (e.g., when aligning assembly contigs) or partial order alignment [12, 94], and eventually extending this framework to handle general sequence graphs (e.g., de-Bruijn graphs). Our research demonstrates that exact algorithms can be leveraged for graphs with millions of vertices and edges. Therefore, making PaSGAL available as a flexible library for exact pattern matching to different graph formats while supporting multiple alignment modes, will be of significant value. An exact algorithm combined with an appropriate graph localization heuristic can further scale to variation graphs built using whole vertebrae genomes.

Appendices

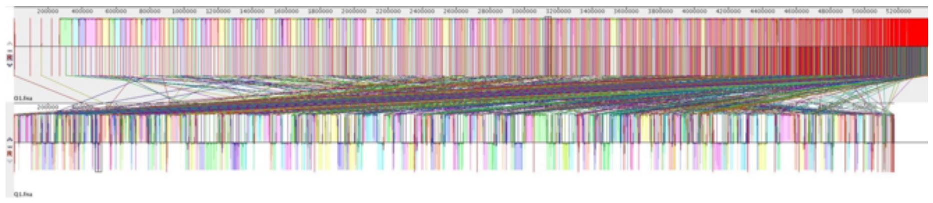
APPENDIX A
GENOME DISTANCE ESTIMATION FOR HIGH-THROUGHPUT ANI ANALYSIS

In this section, we report additional experiments to further support the conclusions in Chapter 4:

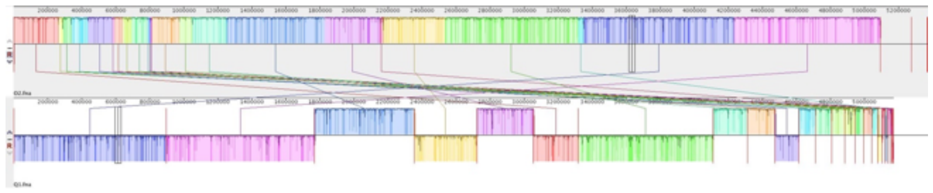
- We selected the fragment length parameter in FastANI as 3 Kbp during our analyses (Section 4.2.2). Table A.1 highlights the accuracy vs. runtime trade-off associated with varying this parameter, and justifies that 3 Kbp is a reasonable choice.
- ANI estimates using FastANI can be distorted if the input data has significant contamination (Section 4.3.2). We further analyzed the pair of outlier strains in dataset D4. Figure A.1 shows that their whole-genome alignment pattern against the query strain looks significantly different from what we would normally expect.
- Our accuracy analysis with five datasets D1-D5 was executed using a single query genome (Figure 4.3). Using Figure A.2, we further show that the correlation behavior remains unaffected even with multiple randomly selected query genomes.
- In Figure A.3, we note that the genetic discontinuity (i.e., the bimodal ANI distribution) that we observed using FastANI can be observed using Mash [82] also.
- The last two figures (Figures A.4, A.5) show the revised ANI distribution curves on reduced genome databases to account for the cultivation bias in the NCBI database.

Table A.1: Evaluation of FastANI accuracy and performance while varying the fragment length l used in the algorithm. We measured Pearson correlation coefficients of FastANI estimate with BLAST-based ANI computation (ANI_b) as well as runtime and memory usage for each value of fragment size (1 Kbp - 10 Kbp). This experiment was conducted using datasets D3 and D4. From the table, it is evident that increasing fragment size improves runtime and memory usage, but negatively affects accuracy. Based on these trade-offs, we set the fragment size to 3 Kbp in the FastANI implementation.

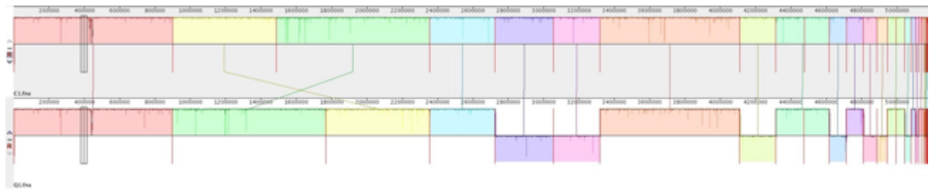
Dataset	Metric	Fragment length l			
		1 Kbp	3 Kbp	5 Kbp	10 Kbp
D3	Correlation with ANI_b	0.998	0.995	0.992	0.987
	Runtime (index phase) in seconds	2590	1667	1435	1145
	Runtime (compute phase) in seconds	4738	2099	1286	547
	Memory (GB)	114	48	29	15
D4	Correlation with ANI_b	0.965	0.944	0.902	0.787
	Runtime (index phase) in seconds	257	175	159	125
	Runtime (compute phase) in seconds	747	254	173	74
	Memory (GB)	12.6	4.4	3.2	1.6



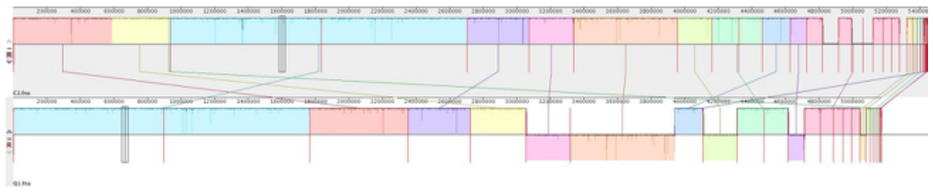
(a) Mauve alignment of first outlier B. *anthracis* strain 2002734165 against the query strain.



(b) Mauve alignment of second outlier B. *anthracis* strain Ba_A2012_AAAC01000001 against the query strain.

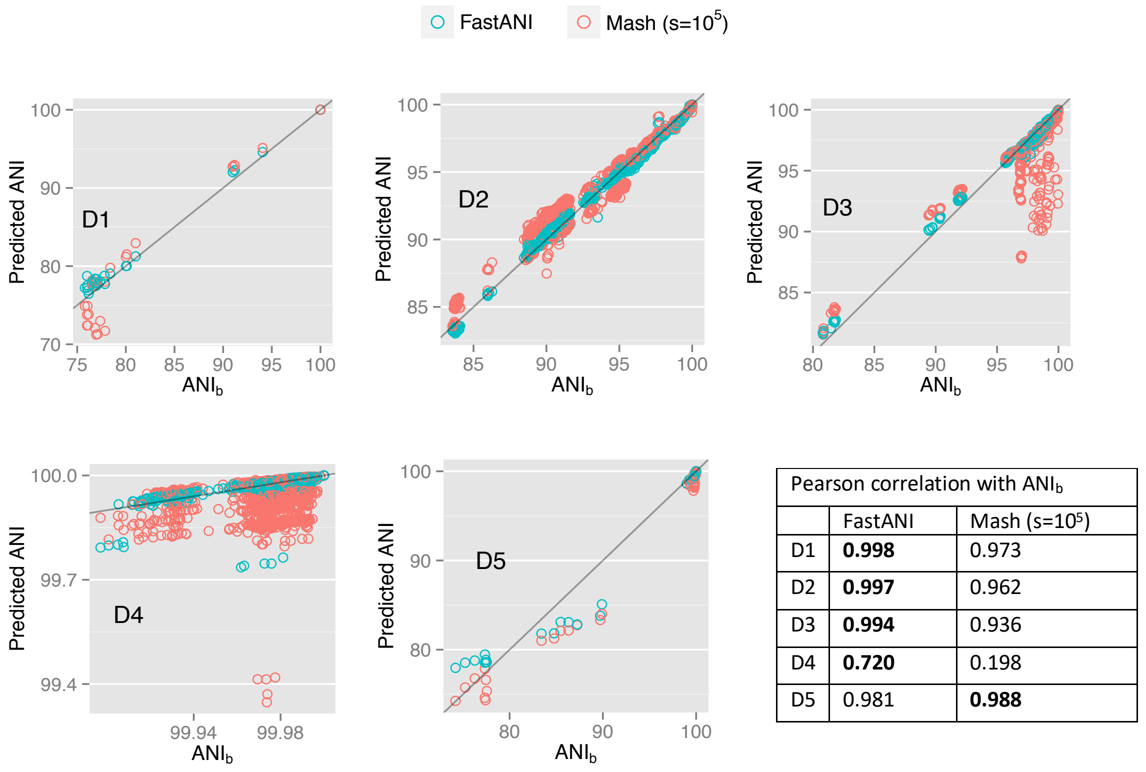


(c) Mauve alignment of a randomly picked B. *anthracis* strain (2000031757) against the query strain.



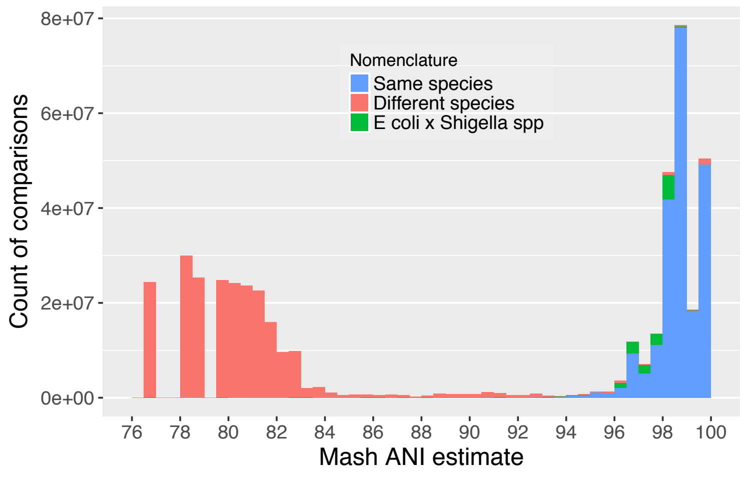
(d) Mauve alignment of another randomly picked B. *anthracis* strain (2002734211) against the query strain.

Figure A.1: Top two plots show the mauve alignments of the two outlier B. *anthracis* strains (2002734165 and Ba_A2012_AAAC01000001) against the query strain (2000031001) used in D4 dataset. Bottom two plots show the mauve alignments of two randomly picked B. *anthracis* strains against the query strain. The top two outlier strains show unusually higher degree of recombination and gaps than we expect between any two correctly sequenced and assembled B. *anthracis* strains. Same behavior was also observed using visualization support in FastANI software (figures not shown here). CheckM statistics for the first outlier genome indicated high strain heterogeneity (i.e., contamination from closely related taxa) of 9%. Mean strain heterogeneity in dataset D4 is 0.13%. Quality control and reassembly of raw sequences using Sickle [189] and Spades [190] respectively didn't improve the assembly quality, indicating contamination at the read level. Based on the CheckM estimates, the second outlier genome had the highest incompleteness of 7% in dataset D4. Reads for the second genome were not publicly accessible to perform a re-assembly.

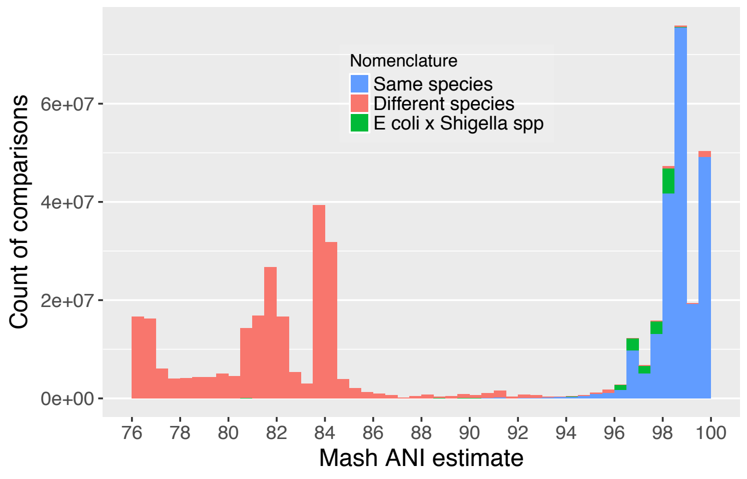


Pearson correlation with ANI$_b$		
	FastANI	Mash (s=10^5)
D1	**0.998**	0.973
D2	**0.997**	0.962
D3	**0.994**	0.936
D4	**0.720**	0.198
D5	0.981	**0.988**

Figure A.2: Correlation of FastANI and Mash output with ANI$_b$ using the five datasets D1-D5 listed in Table 1. For a more robust comparison, we re-executed the experiment in main text (Figure 4.3) with five randomly picked query genomes per dataset that were typically assigned to different species. Similar to what we observed before (Figure 4.3, Table 4.2), FastANI continues to demonstrate either superior or competitive performance than Mash using all the datasets.



(a) Mash sketch size = 1000



(b) Mash sketch size = 10000

Figure A.3: Histogram plot showing the distribution of ANI values among the 90K genomes with ANI estimated using Mash. Similar to the FastANI results presented in main text (Figure 4.7c), the bimodal ANI distribution is persistent. Moreover, the 95% ANI species cutoff is evident using Mash results as well. Considering the shapes of the two peaks, the right peak matches with the one obtained using FastANI, but shape of the left peak differs. In lower identity range however, we expect FastANI's output (Figure 4.7c) to be more reliable than Mash. For this all vs. all comparison, Mash took much less time compared to FastANI- only 51 CPU hours and 359 CPU hours with sketch sizes 10^3 and 10^4 respectively. However, Mash failed to produce output with sketch size 10^5 due to a runtime error.

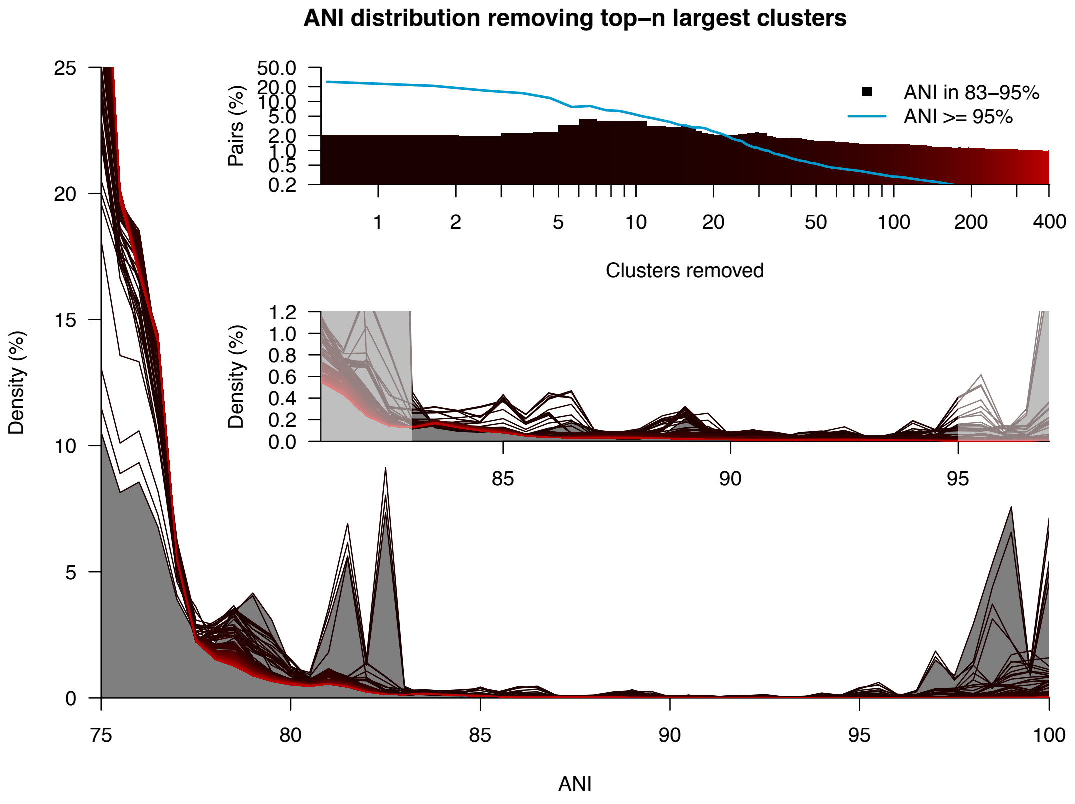


Figure A.4: The sequence discontinuity was also evident when the species with large numbers of representatives in the database were iteratively removed from the database. We computed genome clusters corresponding to genome pairs with ≥95% ANI values. The main plot shows a histogram of ANI values (filled grey) for all genomes in the database (same as Figure 4.7a). The different lines shown correspond to the resulting histograms after removing the top-n largest clusters, with n ranging from zero (black line) to 400 (brightest red line). Notice that the second peak on the right side starts to disappear when all 400 clusters are removed due to reduced intra-species comparisons in the database. The top inset shows how the fraction of pairs that fall into the 83-95% ANI region (with respect to the 75%-100% ANI region) vary with the number of clusters removed (black-to-red filled area, both in log scale), as well as the fraction of pairs above 95% (blue line). Notice that while removing the top-n largest clusters, the fraction of pairs inside the valley is consistently below 5%. The bottom inset shows a magnification of the valley region (83% - 95% ANI).

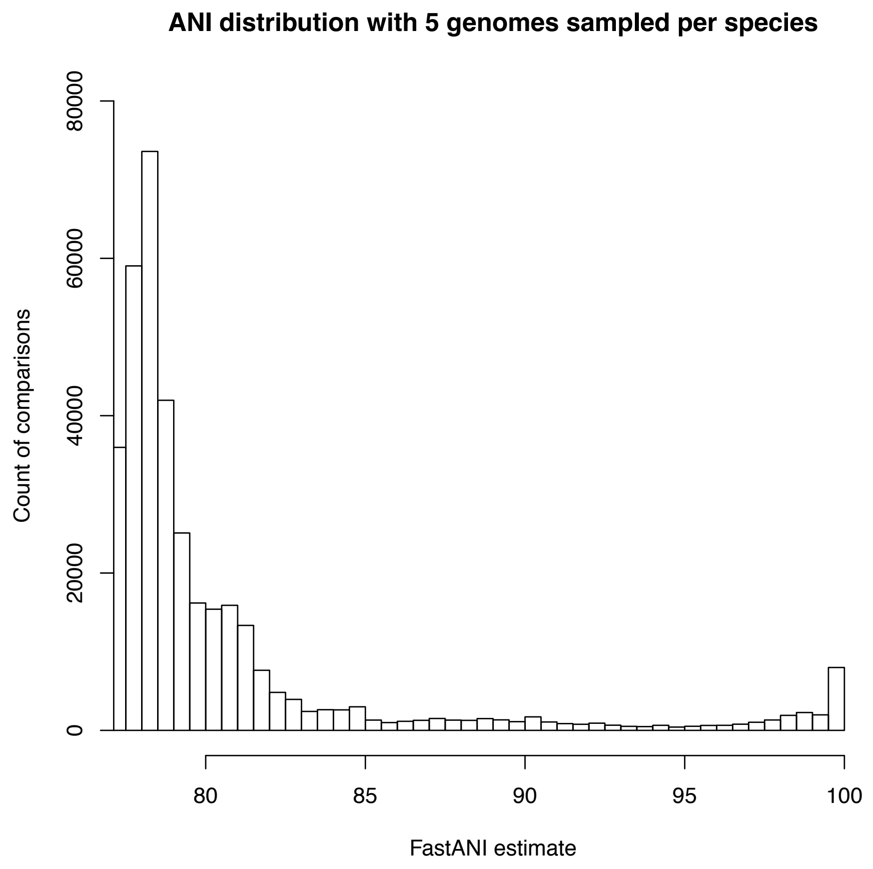


Figure A.5: Distribution of pairwise ANI values in a genome set that is characterized by equal representation of named species in our dataset. First, we selected all named species for which there were five or more genomes available in the NCBI database (750 in count). A custom database was created with five genomes randomly picked for each named species, yielding a total of 3,750 genomes. Discontinuity is still evident with FastANI reporting only 0.2% inter-species pairs in the valley region (83%-95%) out of the total inter-species pair count present in the sampled set. The right-hand side peak is small (relative to Fig. 4.7a) because we have only five genomes per species yielding only 18,750 intra-species pairs in the sampled set. Similar observations were drawn when we sampled two genomes per species yielding a set of 4,838 genomes (2,419 species x 2) [data not shown].

REFERENCES

[1] E. S. Lander, L. M. Linton, B. Birren, C. Nusbaum, M. C. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, W. FitzHugh, *et al.*, "Initial sequencing and analysis of the human genome," *Nature*, vol. 409, no. 6822, pp. 860–921, 2001.

[2] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, *et al.*, "The sequence of the human genome," *science*, vol. 291, no. 5507, pp. 1304–1351, 2001.

[3] J. Shendure, S. Balasubramanian, G. M. Church, W. Gilbert, J. Rogers, J. A. Schloss, and R. H. Waterston, "DNA sequencing at 40: Past, present and future," *Nature*, vol. 550, no. 7676, p. 345, 2017.

[4] M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes, *et al.*, "Nanopore sequencing and assembly of a human genome with ultra-long reads," *Nature biotechnology*, 2018.

[5] M. Jain, H. E. Olsen, B. Paten, and M. Akeson, "The oxford nanopore minion: Delivery of nanopore sequencing to the genomics community," *Genome biology*, vol. 17, no. 1, p. 239, 2016.

[6] J. Quick, N. J. Loman, S. Duraffour, J. T. Simpson, E. Severi, L. Cowley, J. A. Bore, R. Koundouno, G. Dudas, A. Mikhail, *et al.*, "Real-time, portable genome sequencing for ebola surveillance," *Nature*, vol. 530, no. 7589, pp. 228–232, 2016.

[7] I. Rytsareva, D. S. Campo, Y. Zheng, S. Sims, S. V. Thankachan, C. Tetik, J. Chirag, S. P. Chockalingam, A. Sue, S. Aluru, *et al.*, "Efficient detection of viral transmissions with next-generation sequencing data," *BMC genomics*, vol. 18, no. 4, p. 372, 2017.

[8] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[9] P. A. Pevzner, H. Tang, and M. S. Waterman, "An eulerian path approach to DNA fragment assembly," *Proceedings of the National Academy of Sciences*, vol. 98, no. 17, pp. 9748–9753, 2001.

[10] E. W. Myers, "The fragment assembly string graph," *Bioinformatics*, vol. 21, no. suppl_2, pp. ii79–ii85, 2005.

[11] S. Heber, M. Alekseyev, S.-H. Sze, H. Tang, and P. A. Pevzner, “Splicing graphs and est assembly problem,” in *ISMB*, 2002, pp. 181–188.

[12] C. Lee, C. Grasso, and M. F. Sharlow, “Multiple sequence alignment using partial order graphs,” *Bioinformatics*, vol. 18, no. 3, pp. 452–464, 2002.

[13] Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean, “De novo assembly and genotyping of variants using colored de bruijn graphs,” *Nature genetics*, vol. 44, no. 2, p. 226, 2012.

[14] C. P.-G. Consortium, “Computational pan-genomics: Status, promises and challenges,” *Briefings in Bioinformatics*, bbw089, 2016.

[15] A. Dilthey, C. Cox, Z. Iqbal, M. R. Nelson, and G. McVean, “Improved genome inference in the MHC using a population reference graph,” *Nature genetics*, vol. 47, no. 6, p. 682, 2015.

[16] E. Garrison, J. Sirén, A. M. Novak, G. Hickey, J. M. Eizenga, E. T. Dawson, W. Jones, S. Garg, C. Markello, M. F. Lin, *et al.*, “Variation graph toolkit improves read mapping by representing genetic variation in the reference,” *Nature biotechnology*, 2018.

[17] R. R. Wick, L. M. Judd, C. L. Gorrie, and K. E. Holt, “Unicycler: Resolving bacterial genome assemblies from short and long sequencing reads,” *PLoS computational biology*, vol. 13, no. 6, e1005595, 2017.

[18] H. Zhang, C. Jain, and S. Aluru, “A comprehensive evaluation of long read error correction methods,” *bioRxiv*, 2019. eprint: `https://www.biorxiv.org/content/early/2019/01/13/519330.full.pdf`.

[19] G. P. Consortium *et al.*, “A global reference for human genetic variation,” *Nature*, vol. 526, no. 7571, p. 68, 2015.

[20] G. Navarro, “Improved approximate pattern matching on hypertext,” *Theoretical Computer Science*, vol. 237, no. 1-2, pp. 455–463, 2000.

[21] H. Li and N. Homer, “A survey of sequence alignment algorithms for next-generation sequencing,” *Briefings in Bioinformatics*, vol. 11, no. 5, pp. 473–483, 2010.

[22] H. Li and R. Durbin, “Fast and accurate short read alignment with burrows–wheeler transform,” *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[23] B. Langmead and S. L. Salzberg, “Fast gapped-read alignment with bowtie 2,” *Nature Methods*, vol. 9, no. 4, pp. 357–359, 2012.

[24] M. J. Chaisson and G. Tesler, “Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): Application and theory,” *BMC Bioinformatics*, vol. 13, no. 1, p. 238, 2012.

[25] H. Li, “Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM,” *arXiv preprint arXiv:1303.3997*, 2013.

[26] P. Ferragina and G. Manzini, “Opportunistic data structures with applications,” in *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, IEEE, 2000, pp. 390–398.

[27] I. Sović, M. Šikić, A. Wilm, S. N. Fenlon, S. Chen, and N. Nagarajan, “Fast and sensitive mapping of nanopore sequencing reads with graphmap,” *Nature communications*, vol. 7, p. 11 307, 2016.

[28] H.-N. Lin and W.-L. Hsu, “Kart: A divide-and-conquer algorithm for ngs read alignment,” *Bioinformatics*, vol. 33, no. 15, pp. 2281–2287, 2017.

[29] F. J. Sedlazeck, P. Rescheneder, M. Smolka, H. Fang, M. Nattestad, A. von Haeseler, and M. C. Schatz, “Accurate detection of complex structural variations using single-molecule sequencing,” *Nat Methods*, vol. 15, no. 6, pp. 461–468, 2018.

[30] B. Liu, Y. Gao, and Y. Wang, “Lamsa: Fast split read alignment with long approximate matches,” *Bioinformatics*, vol. 33, no. 2, pp. 192–201, 2017.

[31] P. T. Afshar and W. H. Wong, “Cosine: Non-seeding method for mapping long noisy sequences,” *Nucleic acids research*, vol. 45, no. 14, e132–e132, 2017.

[32] D. Nashta-ali, A. Aliyari, A. A. Moghadam, M. A. Edrisi, S. A. Motahari, and B. H. Khalaj, “Meta-aligner: Long-read alignment based on genome statistics,” *BMC bioinformatics*, vol. 18, no. 1, p. 126, 2017.

[33] B. Liu, D. Guan, M. Teng, and Y. Wang, “Rhat: Fast alignment of noisy long reads with regional hashing,” *Bioinformatics*, vol. 32, no. 11, pp. 1625–1631, 2015.

[34] H. Li, “Minimap2: Pairwise alignment for nucleotide sequences,” *Bioinformatics*, vol. 1, p. 7, 2018.

[35] E. Haghshenas, S. C. Sahinalp, and F. Hach, “Lordfast: Sensitive and fast alignment search tool for long noisy read sequencing data,” *Bioinformatics*, vol. 35, no. 1, pp. 20–27, 2018.

[36] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke, “Reducing storage requirements for biological sequence comparison,” *Bioinformatics*, vol. 20, no. 18, pp. 3363–3369, 2004.

[37] H. Suzuki and M. Kasahara, "Introducing difference recurrence relations for faster semi-global alignment of long sequences," *BMC bioinformatics*, vol. 19, no. 1, p. 45, 2018.

[38] H. Gamaarachchi, S. Parameswaran, and M. Smith, "Featherweight long read alignment using partitioned reference indexes," *bioRxiv*, 2018. eprint: `https://www.biorxiv.org/content/early/2018/08/07/386847.full.pdf`.

[39] M. H. N. Yousefi, M. Goudarzi, and S. A. Motahari, "Imos: Improved meta-aligner and minimap2 on spark," *BMC bioinformatics*, vol. 20, no. 1, p. 51, 2019.

[40] A. Z. Broder, "On the resemblance and containment of documents," in *Compression and Complexity of Sequences 1997. Proceedings*, IEEE, 1997, pp. 21–29.

[41] K. Berlin, S. Koren, C.-S. Chin, J. P. Drake, J. M. Landolin, and A. M. Phillippy, "Assembling large genomes with single-molecule sequencing and locality-sensitive hashing," *Nature Biotechnology*, vol. 33, no. 6, pp. 623–630, 2015.

[42] V. Popic and S. Batzoglou, "A hybrid cloud read aligner based on minhash and kmer voting that preserves privacy," *Nature communications*, vol. 8, p. 15 311, 2017.

[43] J. Quedenfeld and S. Rahmann, "Variant tolerant read mapping using min-hashing," *arXiv preprint arXiv:1702.01703*, 2017.

[44] C. Jain, S. Koren, A. M. Phillippy, A. Dilthey, and S. Aluru, "A fast adaptive algorithm for computing whole-genome homology maps," *Bioinformatics*, vol. 34, no. 17, pp. i748–i756, 2018.

[45] C. Jain, L. M. Rodriguez-R, A. M. Phillippy, K. T. Konstantinidis, and S. Aluru, "High throughput ani analysis of 90k prokaryotic genomes reveals clear species boundaries," *Nature communications*, vol. 9, no. 1, p. 5114, 2018.

[46] N. C. Kyrpides, P. Hugenholtz, J. A. Eisen, T. Woyke, M. Göker, C. T. Parker, R. Amann, B. J. Beck, P. S. Chain, J. Chun, *et al.*, "Genomic encyclopedia of bacteria and archaea: Sequencing a myriad of type strains," *PLoS biology*, vol. 12, no. 8, e1001920, 2014.

[47] G. England, "The 100,000 genomes project," 2016, `https://www.genomicsengland.co.uk`.

[48] K.-P. Koepfli, B. Paten, G. K. C. of Scientists, and S. J. O'Brien, "The genome 10K project: A way forward," *Annu. Rev. Anim. Biosci.*, vol. 3, no. 1, pp. 57–111, 2015.

[49] E. D. Jarvis, S. Mirarab, A. J. Aberer, B. Li, P. Houde, C. Li, S. Y. Ho, B. C. Faircloth, B. Nabholz, J. T. Howard, *et al.*, “Whole-genome analyses resolve early branches in the tree of life of modern birds,” *Science*, vol. 346, no. 6215, pp. 1320–1331, 2014.

[50] G. P. Consortium *et al.*, “A map of human genome variation from population-scale sequencing,” *Nature*, vol. 467, no. 7319, pp. 1061–1073, 2010.

[51] D. Haussler, S. J. O’Brien, O. A. Ryder, F. K. Barker, M. Clamp, A. J. Crawford, R. Hanner, O. Hanotte, J. A. McGuire, W. Miller, *et al.*, “Genome 10K: A proposal to obtain whole-genome sequence for 10 000 vertebrate species,” 2009.

[52] D. Earl, N. Nguyen, G. Hickey, R. S. Harris, S. Fitzgerald, K. Beal, I. Seledtsov, V. Molodtsov, B. J. Raney, H. Clawson, *et al.*, “Alignathon: A competitive assessment of whole-genome alignment methods,” *Genome research*, vol. 24, no. 12, pp. 2077–2089, 2014.

[53] J. Armstrong, I. T. Fiddes, M. Diekhans, and B. Paten, “Whole-genome alignment and comparative annotation,” *Annual review of animal biosciences*, 2018.

[54] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, “Gapped blast and psi-blast: A new generation of protein database search programs,” *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, 1997.

[55] B. Ma, J. Tromp, and M. Li, “Patternhunter: Faster and more sensitive homology search,” *Bioinformatics*, vol. 18, no. 3, pp. 440–445, 2002.

[56] S. Schwartz, W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller, “Human–mouse alignments with BLASTZ,” *Genome research*, vol. 13, no. 1, pp. 103–107, 2003.

[57] D. Yorukoglu, Y. W. Yu, J. Peng, and B. Berger, “Compressive mapping for next-generation sequencing,” *Nature biotechnology*, vol. 34, no. 4, p. 374, 2016.

[58] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg, “Alignment of whole genomes,” *Nucleic acids research*, vol. 27, no. 11, pp. 2369–2376, 1999.

[59] M. Brudno, M. Chapman, B. Göttgens, S. Batzoglou, and B. Morgenstern, “Fast and sensitive multiple alignment of large genomic sequences,” *BMC bioinformatics*, vol. 4, no. 1, p. 66, 2003.

[60] N. Bray, I. Dubchak, and L. Pachter, “AVID: A global alignment program,” *Genome research*, vol. 13, no. 1, pp. 97–102, 2003.

[61] M. Vyverman, B. De Baets, V. Fack, and P. Dawyndt, “Essamem: Finding maximal exact matches using enhanced sparse suffix arrays,” *Bioinformatics*, vol. 29, no. 6, pp. 802–804, 2013.

[62] A. L. Marçais G Delcher, A. M. Phillippy, C. Rachel, S. L. Salzberg, and Z. Aleksey, “MUMmer4: A fast and versatile genome alignment system,” *PLoS Comput Biol*, vol. 14, no. 1, 2018.

[63] M. G. Grabherr, P. Russell, M. Meyer, E. Mauceli, J. Alföldi, F. Di Palma, and K. Lindblad-Toh, “Genome-wide synteny through highly sensitive sequence alignment: Satsuma,” *Bioinformatics*, vol. 26, no. 9, pp. 1145–1151, 2010.

[64] M. R. Vollger, P. C. Dishuck, M. Sorensen, A. E. Welch, V. Dang, M. L. Dougherty, T. A. Graves-Lindsay, R. K. Wilson, M. J. Chaisson, and E. E. Eichler, “Long-read sequence and assembly of segmental duplications,” *Nature methods*, vol. 16, no. 1, p. 88, 2019.

[65] J. A. Bailey and E. E. Eichler, “Primate segmental duplications: Crucibles of evolution, diversity and disease,” *Nature Reviews Genetics*, vol. 7, no. 7, pp. 552–564, 2006.

[66] B. S. Emanuel and T. H. Shaikh, “Segmental duplications: An expanding role in genomic instability and disease,” *Nature Reviews Genetics*, vol. 2, no. 10, pp. 791–800, 2001.

[67] L. Pu, Y. Lin, and P. A. Pevzner, “Detection and analysis of ancient segmental duplications in mammalian genomes,” *Genome research*, vol. 28, no. 6, pp. 901–909, 2018.

[68] I. Numanagić, A. S. Gökkaya, L. Zhang, B. Berger, C. Alkan, and F. Hach, “Fast characterization of segmental duplications in genome assemblies,” *Bioinformatics*, vol. 34, no. 17, pp. i706–i714, 2018.

[69] J. A. Bailey, Z. Gu, R. A. Clark, K. Reinert, R. V. Samonte, S. Schwartz, M. D. Adams, E. W. Myers, P. W. Li, and E. E. Eichler, “Recent segmental duplications in the human genome,” *Science*, vol. 297, no. 5583, pp. 1003–1007, 2002.

[70] J. A. Bailey, A. M. Yavor, H. F. Massa, B. J. Trask, and E. E. Eichler, “Segmental duplications: Organization and impact within the current human genome project assembly,” *Genome research*, vol. 11, no. 6, pp. 1005–1017, 2001.

[71] K. T. Konstantinidis and J. M. Tiedje, “Genomic insights that advance the species definition for prokaryotes,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 7, pp. 2567–2572, 2005.

[72] J. Chun, A. Oren, A. Ventosa, H. Christensen, D. R. Arahal, M. S. da Costa, A. P. Rooney, H. Yi, X.-W. Xu, S. De Meyer, *et al.*, "Proposed minimal standards for the use of genome data for the taxonomy of prokaryotes," *International journal of systematic and evolutionary microbiology*, vol. 68, no. 1, pp. 461–466, 2018.

[73] J. Goris, K. T. Konstantinidis, J. A. Klappenbach, T. Coenye, P. Vandamme, and J. M. Tiedje, "DNA-DNA hybridization values and their relationship to whole-genome sequence similarities," *International journal of systematic and evolutionary microbiology*, vol. 57, no. 1, pp. 81–91, 2007.

[74] R. Rosselló-Mora, "Updating prokaryotic taxonomy," *Journal of bacteriology*, vol. 187, no. 18, pp. 6255–6257, 2005.

[75] M. Richter and R. Rosselló-Móra, "Shifting the genomic gold standard for the prokaryotic species definition," *Proceedings of the National Academy of Sciences*, vol. 106, no. 45, pp. 19 126–19 131, 2009.

[76] S. Ciufo, S. Kannan, S. Sharma, A. Badretdin, K. Clark, S. Turner, S. Brover, C. L. Schoch, A. Kimchi, and M. DiCuccio, "Using average nucleotide identity to improve taxonomic assignments in prokaryotic genomes at the ncbi," *International journal of systematic and evolutionary microbiology*, 2018.

[77] L. M. Rodriguez-R and K. T. Konstantinidis, "The enveomics collection: A toolbox for specialized analyses of microbial genomes and metagenomes," *PeerJ Preprints*, vol. 4, e1900v1, Mar. 2016.

[78] I. Lee, Y. O. Kim, S.-C. Park, and J. Chun, "OrthoANI: An improved algorithm and software for calculating average nucleotide identity," *International Journal of Systematic and Evolutionary Microbiology*, vol. 66, no. 2, pp. 1100–1103, 2016.

[79] S.-H. Yoon, S.-m. Ha, J. Lim, S. Kwon, and J. Chun, "A large-scale evaluation of algorithms to calculate average nucleotide identity," *Antonie van Leeuwenhoek*, vol. 110, no. 10, pp. 1281–1286, 2017.

[80] R. C. Edgar, "Search and clustering orders of magnitude faster than BLAST," *Bioinformatics*, vol. 26, no. 19, pp. 2460–2461, 2010.

[81] S. Kurtz, A. Phillippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. L. Salzberg, "Versatile and open software for comparing large genomes," *Genome biology*, vol. 5, no. 2, R12, 2004.

[82] B. D. Ondov, T. J. Treangen, P. Melsted, A. B. Mallonee, N. H. Bergman, S. Koren, and A. M. Phillippy, "Mash: Fast genome and metagenome distance estimation using minhash," *Genome Biology*, 2016.

[83] M. R. Olm, C. T. Brown, B. Brooks, and J. F. Banfield, "dRep: A tool for fast and accurate genomic comparisons that enables improved genome recovery from metagenomes through de-replication.," *The ISME journal*, 2017.

[84] A. T. Dilthey, P.-A. Gourraud, A. J. Mentzer, N. Cereb, Z. Iqbal, and G. McVean, "High-accuracy HLA type inference from whole-genome sequencing data using population reference graphs," *PLoS computational biology*, vol. 12, no. 10, e1005151, 2016.

[85] H. P. Eggertsson, H. Jonsson, S. Kristmundsdottir, E. Hjartarson, B. Kehr, G. Masson, F. Zink, K. E. Hjorleifsson, A. Jonasdottir, A. Jonasdottir, *et al.*, "Graphtyper enables population-scale genotyping using pangenome graphs," *Nature genetics*, vol. 49, no. 11, p. 1654, 2017.

[86] J. A. Sibbesen, L. Maretty, and A. Krogh, "Accurate genotyping across variant classes and lengths using variant graphs," Nature Publishing Group, Tech. Rep., 2018.

[87] W. P. Rowe and M. D. Winn, "Indexed variation graphs for efficient and accurate resistome profiling," *Bioinformatics*, vol. 1, p. 8, 2018.

[88] L. Salmela and E. Rivals, "Lordec: Accurate and efficient long read error correction," *Bioinformatics*, vol. 30, no. 24, pp. 3506–3514, 2014.

[89] J. R. Wang, J. Holt, L. McMillan, and C. D. Jones, "Fmlrc: Hybrid long read error correction using an FM-index," *BMC bioinformatics*, vol. 19, no. 1, p. 50, 2018.

[90] F. Ribeiro, D. Przybylski, S. Yin, T. Sharpe, S. Gnerre, A. Abouelleil, A. M. Berlin, A. Montmayeur, T. P. Shea, B. J. Walker, *et al.*, "Finished bacterial genomes from shotgun sequence data," *Genome research*, gr–141 515, 2012.

[91] D. Antipov, A. Korobeynikov, J. S. McLean, and P. A. Pevzner, "Hybridspades: An algorithm for hybrid assembly of short and long reads," *Bioinformatics*, vol. 32, no. 7, pp. 1009–1015, 2015.

[92] S. Garg, M. Rautiainen, A. M. Novak, E. Garrison, R. Durbin, and T. Marschall, "A graph-based approach to diploid genome assembly," *Bioinformatics*, vol. 34, no. 13, pp. i105–i114, 2018.

[93] C.-S. Chin, D. H. Alexander, P. Marks, A. A. Klammer, J. Drake, C. Heiner, A. Clum, A. Copeland, J. Huddleston, E. E. Eichler, *et al.*, "Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data," *Nature Methods*, vol. 10, no. 6, pp. 563–569, 2013.

[94] R. Vaser, I. Sović, N. Nagarajan, and M. Šikić, "Fast and accurate de novo genome assembly from long uncorrected reads," *Genome research*, 2017.

[95] A. Kuosmanen, A. Sobih, R. Rizzi, V. Mäkinen, and A. I. Tomescu, "On using longer RNA-seq reads to improve transcript prediction accuracy.," in *BIOINFORMATICS*, 2016, pp. 272–277.

[96] A. Kuosmanen, T. Paavilainen, T. Gagie, R. Chikhi, A. Tomescu, and V. Mäkinen, "Using minimum path cover to boost dynamic programming on DAGs: Co-linear chaining extended," in *International Conference on Research in Computational Molecular Biology*, Springer, 2018, pp. 105–121.

[97] S. Beretta, P. Bonizzoni, L. Denti, M. Previtali, and R. Rizzi, "Mapping RNA-seq data to a transcript graph via approximate pattern matching to a hypertext," in *International Conference on Algorithms for Computational Biology*, Springer, 2017, pp. 49–61.

[98] L. Denti, R. Rizzi, S. Beretta, G. Della Vedova, M. Previtali, and P. Bonizzoni, "Asgal: Aligning rna-seq data to a splicing graph to detect novel alternative splicing events," *BMC bioinformatics*, vol. 19, no. 1, p. 444, 2018.

[99] K. Thompson, "Programming techniques: Regular expression search algorithm," *Communications of the ACM*, vol. 11, no. 6, pp. 419–422, 1968.

[100] R. A. Wagner, "Order-n correction for regular languages," *Communications of the ACM*, vol. 17, no. 5, pp. 265–268, 1974.

[101] E. W. Myers and W. Miller, "Approximate matching of regular expressions," *Bulletin of mathematical biology*, vol. 51, no. 1, pp. 5–37, 1989.

[102] R. Abarbanel, P. R. Wieneke, E Mansfield, D. A. Jaffe, and D. L. Brutlag, "Rapid searches for complex patterns in biological molecules," 1984.

[103] F. Cohen, R. Abarbanel, I. Kuntz, and R. Fletterick, "Turn prediction in proteins using a pattern-matching approach," *Biochemistry*, vol. 25, no. 1, pp. 266–275, 1986.

[104] U. Manber and S. Wu, "Approximate string matching with arbitrary costs for text and hypertext," in *Advances In Structural And Syntactic Pattern Recognition*, World Scientific, 1992, pp. 22–33.

[105] K. Park and D. K. Kim, "String matching in hypertext," in *Annual Symposium on Combinatorial Pattern Matching*, Springer, 1995, pp. 318–329.

[106] T. Akutsu, “A linear time pattern matching algorithm between a string and a tree,” in *Annual Symposium on Combinatorial Pattern Matching*, Springer, 1993, pp. 1–10.

[107] A. Amir, M. Lewenstein, and N. Lewenstein, “Pattern matching in hypertext,” *Journal of Algorithms*, vol. 35, no. 1, pp. 82–99, 2000.

[108] C. Jain, H. Zhang, Y. Gao, and S. Aluru, “On the complexity of sequence to graph alignment,” in *International Conference on Research in Computational Molecular Biology*, Springer, 2019 (to appear).

[109] M. Rautiainen and T. Marschall, “Aligning sequences to general graphs in O(V + mE) time,” *bioRxiv*, 2017. eprint: `https://www.biorxiv.org/content/early/2017/11/08/216127.full.pdf`.

[110] R. Vijaya Satya, N. Zavaljevski, and J. Reifman, “A new strategy to reduce allelic bias in RNA-Seq readmapping,” *Nucleic acids research*, vol. 40, no. 16, e127–e127, 2012.

[111] L. Huang, V. Popic, and S. Batzoglou, “Short read alignment with populations of genomes,” *Bioinformatics*, vol. 29, no. 13, pp. i361–i370, 2013.

[112] B. Liu, H. Guo, M. Brudno, and Y. Wang, “Debga: Read alignment with de bruijn graph-based seed and extension,” *Bioinformatics*, vol. 32, no. 21, pp. 3224–3232, 2016.

[113] D. Kim, J. M. Paggi, and S. Salzberg, “Hisat-genotype: Next generation genomic analysis platform on a personal computer,” *bioRxiv*, 2018. eprint: `https://www.biorxiv.org/content/early/2018/02/15/266197.full.pdf`.

[114] G. Rakocevic, V. Semenyuk, W.-P. Lee, J. Spencer, J. Browning, I. J. Johnson, V. Arsenijevic, J. Nadj, K. Ghose, M. C. Suciu, S.-G. Ji, G. Demir, L. Li, B. Ç. Toptaş, A. Dolgoborodov, B. Pollex, I. Spulber, I. Glotova, P. Kómár, A. L. Stachyra, Y. Li, M. Popovic, M. Källberg, A. Jain, and D. Kural, “Fast and accurate genomic analyses using genome graphs,” *Nature Genetics*, 2019.

[115] J. Sirén, N. Välimäki, and V. Mäkinen, “Indexing graphs for path queries with applications in genome research,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 11, no. 2, pp. 375–388, 2014.

[116] J. Sirén, “Indexing variation graphs,” in *2017 Proceedings of the ninteenth workshop on algorithm engineering and experiments (ALENEX)*, SIAM, 2017, pp. 13–27.

[117] J. Pritt, N.-C. Chen, and B. Langmead, "Forge: Prioritizing variants for graph genomes," *Genome biology*, vol. 19, no. 1, p. 220, 2018.

[118] S. Maciuca, C. del Ojo Elias, G. McVean, and Z. Iqbal, "A natural encoding of genetic variation in a burrows-wheeler transform to enable mapping and genome inference," in *International Workshop on Algorithms in Bioinformatics*, Springer, 2016, pp. 222–233.

[119] E. Biederstedt, J. C. Oliver, N. F. Hansen, A. Jajoo, N. Dunn, A. Olson, B. Busby, and A. T. Dilthey, "Novograph: Genome graph construction from multiple long-read de novo assemblies," *F1000Research*, vol. 7, 2018.

[120] M. Rautiainen, V. Mäkinen, and T. Marschall, "Bit-parallel sequence-to-graph alignment," *bioRxiv*, 2018. eprint: `https://www.biorxiv.org/content/early/2018/05/15/323063.full.pdf`.

[121] M. Farrar, "Striped smith–waterman speeds database searches six times over other SIMD implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2006.

[122] S. M. Kiełbasa, R. Wan, K. Sato, P. Horton, and M. C. Frith, "Adaptive seeds tame genomic sequence comparison," *Genome research*, vol. 21, no. 3, pp. 487–493, 2011.

[123] C Jain, A Dilthey, S Koren, S Aluru, and A. Phillippy, "A fast approximate algorithm for mapping long reads to large reference databases.," *Journal of computational biology: a journal of computational molecular cell biology*, vol. 25, no. 7, p. 766, 2018.

[124] C. Jain, A. Dilthey, S. Koren, S. Aluru, and A. M. Phillippy, "A fast approximate algorithm for mapping long reads to large reference databases," in *International Conference on Research in Computational Molecular Biology*, Springer, 2017, pp. 66–81.

[125] H. Fan, A. R. Ives, Y. Surget-Groba, and C. H. Cannon, "An assembly and alignment-free method of phylogeny reconstruction from next-generation sequencing data," *BMC Genomics*, vol. 16, no. 1, p. 1, 2015.

[126] P. Jaccard, "Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines," *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 241–272, 1901.

[127] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, ACM, 2003, pp. 76–85.

[128] K. C. Smith, *Sliding window minimum implementations*, [Online; accessed 8-September-2016], 2016.

[129] H. Li, "Minimap and miniasm: Fast mapping and de novo assembly for noisy long sequences," *Bioinformatics*, btw152, 2016.

[130] N. J. Loman, *Nanopore R9 rapid run data release*, [Online; accessed 8-September-2016], 2016.

[131] M. J. Chaisson, J. Huddleston, M. Y. Dennis, P. H. Sudmant, M. Malig, F. Hormozdiari, F. Antonacci, U. Surti, R. Sandstrom, M. Boitano, *et al.*, "Resolving the complexity of the human genome using single-molecule sequencing," *Nature*, vol. 517, no. 7536, pp. 608–611, 2015.

[132] D. Laehnemann, A. Borkhardt, and A. C. McHardy, "Denoising DNA deep sequencing data–high-throughput sequencing errors and their correction," *Briefings in Bioinformatics*, vol. 17, no. 1, pp. 154–179, 2016.

[133] Pacific Biosciences, *Human microbiome mock community shotgun sequencing data*, [Online; accessed 8-September-2016], 2014.

[134] A. Dilthey, C. Jain, S. Koren, and A. Phillippy, "Metamaps - strain-level metagenomic assignment and compositional estimation for long reads," *bioRxiv*, 2018. eprint: `https://www.biorxiv.org/content/early/2018/07/20/372474.full.pdf`.

[135] M. I. Shamos and D. Hoey, "Geometric intersection problems," in *Foundations of Computer Science, 1976., 17th Annual Symposium on*, IEEE, 1976, pp. 208–215.

[136] A. Lubiw and A. Rácz, "A lower bound for the integer element distinctness problem," *Information and Computation*, vol. 94, no. 1, pp. 83–92, 1991.

[137] P. Berman, Z. Zhang, Y. I. Wolf, E. V. Koonin, and W. Miller, "Winnowing sequences from a database search," in *Proceedings of the third annual international conference on Computational molecular biology*, ACM, 1999, pp. 50–58.

[138] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy, "Canu: Scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation," *Genome research*, vol. 27, no. 5, pp. 722–736, 2017.

[139] W. J. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and D. Haussler, "The human genome browser at UCSC," *Genome research*, vol. 12, no. 6, pp. 996–1006, 2002.

[140] L. W. Hillier, R. S. Fulton, L. A. Fulton, T. A. Graves, K. H. Pepin, C. Wagner-McPherson, D. Layman, J. Maas, S. Jaeger, R. Walker, *et al.*, “The DNA sequence of human chromosome 7,” *Nature*, vol. 424, no. 6945, p. 157, 2003.

[141] International Human Genome Sequencing Consortium, “Finishing the euchromatic sequence of the human genome,” *Nature*, vol. 431, no. 7011, pp. 931–945, 2004.

[142] V. A. Schneider, T. Graves-Lindsay, K. Howe, N. Bouk, H.-C. Chen, P. A. Kitts, T. D. Murphy, K. D. Pruitt, F. Thibaud-Nissen, D. Albracht, *et al.*, “Evaluation of grch38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly,” *Genome research*, vol. 27, no. 5, pp. 849–864, 2017.

[143] G. Marcais, D. DeBlasio, P. Pandey, and C. Kingsford, “Locality sensitive hashing for the edit distance,” *bioRxiv*, 2019. eprint: `https://www.biorxiv.org/content/early/2019/01/29/534446.full.pdf`.

[144] A. R. Quinlan and I. M. Hall, “Bedtools: A flexible suite of utilities for comparing genomic features,” *Bioinformatics*, vol. 26, no. 6, pp. 841–842, 2010.

[145] C. Luo, S. T. Walk, D. M. Gordon, M. Feldgarden, J. M. Tiedje, and K. T. Konstantinidis, “Genome sequencing of environmental escherichia coli expands understanding of the ecology and speciation of the model bacterial species,” *Proceedings of the National Academy of Sciences*, vol. 108, no. 17, pp. 7200–7205, 2011.

[146] B. J. Shapiro, J. Friedman, O. X. Cordero, S. P. Preheim, S. C. Timberlake, G. Szabó, *et al.*, “Population genomics of early events in the ecological differentiation of bacteria,” *Science*, vol. 336, no. 6077, pp. 48–51, 2012.

[147] P. Yarza, P. Yilmaz, E. Pruesse, F. O. Glöckner, W. Ludwig, K.-H. Schleifer, *et al.*, “Uniting the classification of cultured and uncultured bacteria and archaea using 16S rRNA gene sequences,” *Nature Reviews Microbiology*, vol. 12, no. 9, p. 635, 2014.

[148] D. R. Mende, S. Sunagawa, G. Zeller, and P. Bork, “Accurate and universal delineation of prokaryotic species,” *Nature methods*, vol. 10, no. 9, p. 881, 2013.

[149] N. J. Varghese, S. Mukherjee, N. Ivanova, K. T. Konstantinidis, K. Mavrommatis, N. C. Kyrpides, *et al.*, “Microbial species delineation using whole genome sequences,” *Nucleic acids research*, vol. 43, no. 14, pp. 6761–6771, 2015.

[150] J. Staley, “The phylogenomic species concept,” *Microbiology Today*, vol. 36, pp. 80–83, 2009.

[151] A. Backurs and P. Indyk, "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)," in *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, ACM, 2015, pp. 51–58.

[152] L. Guy, J. Roat Kultima, and S. G. Andersson, "genoPlotR: Comparative gene and genome visualization in R," *Bioinformatics*, vol. 26, no. 18, pp. 2334–2335, 2010.

[153] K. T. Konstantinidis, A. Ramette, and J. M. Tiedje, "Toward a more robust assessment of intraspecies diversity, using fewer genetic markers," *Applied and environmental microbiology*, vol. 72, no. 11, pp. 7286–7293, 2006.

[154] D. H. Parks, C. Rinke, M. Chuvochina, P.-A. Chaumeil, B. J. Woodcroft, P. N. Evans, *et al.*, "Recovery of nearly 8,000 metagenome-assembled genomes substantially expands the tree of life," *Nature microbiology*, vol. 2, no. 11, p. 1533, 2017.

[155] D. H. Parks, M. Imelfort, C. T. Skennerton, P. Hugenholtz, and G. W. Tyson, "Checkm: Assessing the quality of microbial genomes recovered from isolates, single cells, and metagenomes," *Genome research*, vol. 25, no. 7, pp. 1043–1055, 2015.

[156] A. C. Darling, B. Mau, F. R. Blattner, and N. T. Perna, "Mauve: Multiple alignment of conserved genomic sequence with rearrangements," *Genome research*, vol. 14, no. 7, pp. 1394–1403, 2004.

[157] I. Natalia, A. Sorokin, I. Anderson, N. Galleron, B. Candelon, V. Kapatral, *et al.*, "Genome sequence of bacillus cereus and comparative analysis with bacillus anthracis," *Nature*, vol. 423, no. 6935, p. 87, 2003.

[158] M. Kim, H.-S. Oh, S.-C. Park, and J. Chun, "Towards a taxonomic coherence between average nucleotide identity and 16S rRNA gene sequence similarity for species demarcation of prokaryotes," *International journal of systematic and evolutionary microbiology*, vol. 64, no. 2, pp. 346–351, 2014.

[159] A. Caro-Quintero and K. T. Konstantinidis, "Bacterial species may exist, metagenomics reveal," *Environmental microbiology*, vol. 14, no. 2, pp. 347–355, 2012.

[160] C. Fraser, E. J. Alm, M. F. Polz, B. G. Spratt, and W. P. Hanage, "The bacterial species challenge: Making sense of genetic and ecological diversity," *Science*, vol. 323, no. 5915, pp. 741–746, 2009.

[161] F. M. Cohan, "Bacterial species and speciation," *Systematic biology*, vol. 50, no. 4, pp. 513–524, 2001.

[162] K. T. Konstantinidis, A. Ramette, and J. M. Tiedje, "The bacterial species definition in the genomic era," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 361, no. 1475, pp. 1929–1940, 2006.

[163] P. Wilmes, S. L. Simmons, V. J. Denef, and J. F. Banfield, "The dynamic genetic repertoire of microbial communities," *FEMS microbiology reviews*, vol. 33, no. 1, pp. 109–132, 2008.

[164] T. J. Straub and O. Zhaxybayeva, "A null model for microbial diversification," *Proceedings of the National Academy of Sciences*, p. 201 619 993, 2017.

[165] S. Mukherjee, R. Seshadri, N. J. Varghese, E. A. Eloe-Fadrosh, J. P. Meier-Kolthoff, M. Göker, *et al.*, "1,003 reference genomes of bacterial and archaeal isolates expand coverage of the tree of life.," *Nature biotechnology*, 2017.

[166] J Plesn *et al.*, "The np-completeness of the hamiltonian cycle problem in planar diagraphs with degree bound two," *Information Processing Letters*, vol. 8, no. 4, pp. 199–201, 1979.

[167] K. Vaddadi, K. Tayal, R. Srinivasan, and N. Sivadasan, "Sequence alignment on directed graphs," *Journal of Computational Biology*, 2018.

[168] M. Equi, R. Grossi, and V. Mäkinen, "On the complexity of exact pattern matching in graphs: Binary strings and bounded degree," *arXiv preprint arXiv:1901.05264*, 2019.

[169] M. Equi, R. Grossi, A. I. Tomescu, and V. Mäkinen, "On the complexity of exact pattern matching in graphs: Determinism and zig-zag matching," *arXiv preprint arXiv:1902.03560*, 2019.

[170] E. W. Myers, "An O(nd) difference algorithm and its variations," *Algorithmica*, vol. 1, no. 1-4, pp. 251–266, 1986.

[171] C. Jain, S. Misra, H. Zhang, A. Dilthey, and S. Aluru, "Accelerating sequence alignment to graphs," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, 2019 (to appear).

[172] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[173] T. F. Smith and M. S. Waterman, "Comparison of biosequences," *Advances in applied mathematics*, vol. 2, no. 4, pp. 482–489, 1981.

[174] S. Aluru, N. Futamura, and K. Mehrotra, “Parallel biological sequence comparison using prefix computations,” *Journal of Parallel and Distributed Computing*, vol. 63, no. 3, pp. 264–272, 2003.

[175] Y. Liu, D. L. Maskell, and B. Schmidt, “Cudasw++: Optimizing smith-waterman sequence database searches for CUDA-enabled graphics processing units,” *BMC research notes*, vol. 2, no. 1, p. 73, 2009.

[176] A. Khajeh-Saeed, S. Poole, and J. B. Perot, “Acceleration of the smith–waterman algorithm using single and multiple graphics processors,” *Journal of Computational Physics*, vol. 229, no. 11, pp. 4247–4258, 2010.

[177] C. Jain and S. Kumar, “Fine-grained GPU parallelization of pairwise local sequence alignment,” in *High Performance Computing (HiPC), 2014 21st International Conference on*, IEEE, 2014, pp. 1–10.

[178] A. M. Novak, G. Hickey, E. Garrison, S. Blum, A. Connelly, A. Dilthey, J. Eizenga, M. A. S. Elmohamed, S. Guthrie, A. Kahles, *et al.*, “Genome graphs,” *bioRxiv*, 2017. eprint: `https://www.biorxiv.org/content/early/2017/01/18/101378.full.pdf`.

[179] X. Huang, R. C. Hardison, and W. Miller, “A space-efficient algorithm for local similarities,” *Bioinformatics*, vol. 6, no. 4, pp. 373–381, 1990.

[180] T. Rognes, “Faster smith-waterman database searches with inter-sequence SIMD parallelisation,” *BMC bioinformatics*, vol. 12, no. 1, p. 221, 2011.

[181] S. Misra, T. C. Pan, K. Mahadik, G. Powley, P. N. Vaidya, M. Vasimuddin, and S. Aluru, “Performance extraction and suitability analysis of multi-and many-core architectures for next generation sequencing secondary analysis,” in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, ACM, 2018, p. 3.

[182] D. S. Hirschberg, “A linear space algorithm for computing maximal common subsequences,” *Communications of the ACM*, vol. 18, no. 6, pp. 341–343, 1975.

[183] J. A. Grice, R. Hughey, and D. Speck, “Reduced space sequence alignment,” *Bioinformatics*, vol. 13, no. 1, pp. 45–53, 1997.

[184] M. Holtgrewe, “Mason–a read simulator for second generation sequencing data,” *Technical Report FU Berlin*, 2010.

[185] Y. Ono, K. Asai, and M. Hamada, “PBSIM: Pacbio reads simulator—toward accurate genome assembly,” *Bioinformatics*, vol. 29, no. 1, pp. 119–121, 2012.

[186] E. I. Olekhnovich, A. T. Vasilyev, V. I. Ulyantsev, E. S. Kostryukova, and A. V. Tyakht, “Metacherchant: Analyzing genomic context of antibiotic resistance genes in gut microbiota,” *Bioinformatics*, vol. 34, no. 3, pp. 434–444, 2017.

[187] C. T. Brown, D. Moritz, M. O’Brien, F. Reidl, T. Reiter, and B. Sullivan, “Exploring neighborhoods in large metagenome assembly graphs reveals hidden sequence diversity,” *bioRxiv*, 2018. eprint: `https://www.biorxiv.org/content/early/2018/11/05/462788.full.pdf`.

[188] T. Gagie, G. Manzini, and J. Sirén, “Wheeler graphs: A framework for bwt-based data structures,” *Theoretical computer science*, vol. 698, pp. 67–78, 2017.

[189] N. Joshi, J. Fass, *et al.*, *Sickle: A sliding-window, adaptive, quality-based trimming tool for fastq files (version 1.33)[software]*, 2011.

[190] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, *et al.*, “Spades: A new genome assembly algorithm and its applications to single-cell sequencing,” *Journal of computational biology*, vol. 19, no. 5, pp. 455–477, 2012.

VITA

Chirag Jain received his B. Tech degree in computer science from Indian Institute of Technology Delhi in 2014, and an M.S. degree in computational science and engineering (CSE) from Georgia Institute of Technology (GT) in 2018. He enrolled as a full-time student in CSE Ph.D. program in 2014 at GT. During his Ph.D., he did three summer internships, two at the National Institutes of Health, and one at Intel Corporation.

Mr. Jain initially joined the doctoral program to pursue his interests in parallel computing, a field in which he had done research and an internship as undergraduate. Later, he discovered his true passion in computational biology, and also carried out some work at the intersection of the two. Mr. Jain has authored or co-authored over 7 journal articles and 7 conference publications, spanning three areas: bioinformatics, parallel processing, and biology. During his education, Mr. Jain received multiple accolades for his research. One of his papers earned him ACM SIGHPC Certificate of Recognition for Reproducible Methods in Scientific Research at Supercomputing conference in 2016. He is recipient of the first-place award for best project in undergraduate category in 2014 in an institute wide research exhibition (open house) at IIT Delhi. He has served as a co-advisor for GT's undergraduate team to compete in Student Cluster Competition at Supercomputing'17.

Outside work, Mr. Jain enjoys outdoor recreation, sports and non-profit fund-raising activities. He also coached Asha-Atlanta's marathon running team twice during 2016-17 and 2018-19.