IMPROVING SEQUENCE ANALYSIS WITH PROBABILISTIC DATA STRUCTURES AND ALGORITHMS

by

Justin Chu

B.Sc., The University of British Columbia, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

The Faculty of Graduate and Postdoctoral Studies

(Bioinformatics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

December 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the dissertation entitled:

Improving Sequence Analysis using Probabilistic Data structures and Algorithms

submitted by Justin Chu in partial fulfillment of the requirements for the degree of Doctor of Philosophy

in Bioinformatics

Examining Committee:

Inanc Birol

Supervisor

Anne Condon

Supervisory Committee Member

Cedric Chauve

Supervisory Committee Member

Leonard Foster

University Examiner

Jane Wang

University Examiner

Additional Supervisory Committee Members:

Joerg Bohlmann

Supervisory Committee Member

Abstract

In the biological sciences, sequence analysis refers to analytical investigations that use nucleic acid or protein sequences to elucidate biological insights from them, such as their function, species of origin, or evolutionary relationships. However, sequences are not very meaningful by themselves, and useful insights generally come from comparing them to other sequences. Indexing sequences using concepts borrowed from the computational sciences may help perform these comparisons. One such concept is a probabilistic data structure, the Bloom filter, which enables low memory indexing with high computational efficiency at the cost of false-positive queries by storing a signature of a sequence rather than the sequence itself. This thesis explores high-performance applications of this probabilistic data structure in sequence classification (BioBloom Tools) and targeted sequence assembly (Kollector) and shows how these implemented tools outperform state-of-the-art methods.

To remedy some weaknesses of Bloom filters, such as the inability to index multiple targets, I have developed a novel probabilistic data structure called a multi-index Bloom filter (miBF), used to facilitate alignment-free classification of thousands of references. The data structure also synergizes with spaced seeds. Sequences are often broken up into subsequences when using a hashed-based algorithm, and spaced seeds are subsequences with wildcard positions to improve classification sensitivity and specificity. This novel data structure enables faster classification and higher sensitivity than sequence alignment-based methods and executes in an order of magnitude less time while using half the memory compared to other spaced seed-based approaches. This thesis features formulations of classification false-positive rates in relation to the indexed and queried sequences, and benchmarks the data structure on simulated data. In

addition to my work on short read data, I explore and evaluate of methods for finding sequence overlaps in error-prone long read datasets.

Lay Summary

Sequence analysis is the process of using computers to extract meaningful information from biological sequences such as DNA. Analysis of sequencing data has led to great advances in our understanding of living things and in the treatment of diseases. Recently, the technology that reads DNA sequences has advanced to the point that powerful computers cannot analyze them due to the sheer amount of data generated. In this thesis I address this problem by designing new efficient computer programs for many tasks in sequence analysis. I focus on storing and accessing data in a special way that allow for faster programs using less computational resources. These special “probabilistic data structures” purposely store data as approximate signatures to surpass the memory and speed limits of representing data perfectly. Here I demonstrate and expand the applications of these data structures in sequence analysis.

Preface

Chapter 2: BioBloom Tools. A version of this material has been published as Justin Chu, Sara Sadeghi, Anthony Raymond, Shaun D. Jackman, Ka Ming Nip, Richard Mar, Hamid Mohamadi, Yaron S. Butterfield, A. Gordon Robertson, and Inanc Birol. "BioBloom tools: fast, accurate and memory-efficient host species sequence screening using bloom filters." Bioinformatics 30, no. 23 (2014): 3402-3404 (https://doi.org/10.1093/bioinformatics/btu558). JC drafted the manuscript. JC, SS and RM performed testing and evaluation of the tool. AR, SDJ, KMN, HM provided technical and implementational help. YSA and AGR help with editing of the manuscript. IB supervised the project. I thank Irene Li, Rene Warren and Karen Mungall for useful discussions.

Chapter 3: Kollector. A version of the material has been published as Erdi Kucuk, Justin Chu, Benjamin P. Vandervalk, S. Austin Hammond, René L. Warren, and Inanc Birol. "Kollector: transcript-informed, targeted de novo assembly of gene loci." Bioinformatics 33, no. 12 (2017): 1782-1788 (https://doi.org/10.1093/bioinformatics/btx405). EK and JC are joint first authors for this work. JC designed and implemented the machinery needed to perform read collection and progressive Bloom filter creation, aided in experimental design, and wrote and revised much of the manuscript. EK implemented the overarching Kollector scripts, performed the experiments and wrote much of the initial draft of the manuscript. IB supervised the project. The rest of the co-authors on this work provided valuable feedback during the development of the tool and helped in editing the manuscript. I would like to thank Golnaz Jahesh from Canada's Michael Smith Genome Research Centre for her contributions to packaging, testing and releasing of the

Kollector software. I also thank Gordon Robertson from Canada's Michael Smith Genome Research Centre for discussion of the HPV strain detection work.

Chapter 4: Multi-index Bloom Filters. A version of this work is currently under review as well as in preprint form as Justin Chu, Hamid Mohamadi, Emre Erhan, Jeffery Tse, Readman Chiu, Sarah Yeo, and Inanc Birol. "Improving on hash-based probabilistic sequence classification using multiple spaced seeds and multi-index Bloom filters." BioRxiv (2018): 434795 (https://doi.org/10.1101/434795). JC design designed and implemented the data structure within BioBloom Tools. HM helped implement the spaced seed hashing with ntHash used in this paper, as well as making some contributions to the background sections and editing of the paper. EE helped evaluate and designs of spaced seeds in relation to how they compare to each other and k-mers when classifying sequences. JT aided in the testing and development of the miBF and performed various experiments with the software. RC helped evaluate the miBF in relation to BWA in the applications of targeted assembly. SY aided in the implementation of the data structure into a library and in the evaluations of it in metagenomics applications. IB supervised all work, helped to develop the FPR formulation used in the program, developed a script for generating balanced random spaced seeds, and edited/revised the text of the manuscript. I would like to thank Rachid Ounit for providing support in the proper use of CLARK and CLARK-S, as well as providing insight into understanding the results produced using these tools with the datasets from the CLARK-S publication.

Chapter 5: Long read overlap review. A version of this work was published as Justin Chu, Hamid Mohamadi, René L. Warren, Chen Yang, and Inanc Birol. "Innovations and challenges in

detecting long read overlaps: an evaluation of the state-of-the-art." Bioinformatics 33, no. 8 (2016): 1261-1270 (https://doi.org/10.1093/bioinformatics/btw811). JC prepared the manuscript, including the experiments performed. HM helped refine sections relating to spaced seeds and created the data structure/algorithm overview figure. CY provided the means to generate simulated oxford nanopore data and edited the manuscript. IB supervised the project and edited/revised the text of the manuscript. I would like to thank Sergey Koren, Ivan Sović for their help and suggestions when running MHAP and GraphMap, respectively, as well as their insights into the behaviour and results of each tool on different datasets.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BF: Bloom filter

bp: base pair

DBG: de Bruijn graph

FDR: False Discovery Rate

FPR: False Positive Rate

Indel: Insertions or deletion

miBF: Multi-index Bloom filter

OLC: Overlap layout consensus

ONT: Oxford Nanopore Technologies

PB: Pacific Biosciences

PCR: Polymerase chain reaction

Acknowledgements

I thank my supervisor Inanc Birol, and all the members of Bioinformatics Technology Lab. I thank my supervisory committee Anne Condon, Cedric Chauve and Jorge Bohlmann. Finally, I am grateful for the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Killam Fellowship Program who funded my studies, and the agencies that funded the individual research projects, described below.

Chapter 2: The work was funded by Genome Canada, the British Columbia Cancer Foundation, and Genome British Columbia.

Chapter 3: The research presented here was funded by the National Human Genome Research Institute of the National Institutes of Health (under award number R01HG007182), with additional support provided by Genome Canada, Genome British Columbia, the British Columbia Cancer Foundation, and Canada Foundation for Innovation.

Chapter 4: This work was supported by the National Human Genome Research Institute of the National Institutes of Health (R01HG007182), with additional support provided by Genome Canada and Genome British Columbia (BCB2 - 251LRD).

Chapter 6: I thank Genome Canada, Genome British Columbia, British Columbia Cancer Foundation, and the University of British Columbia for their financial support. The work is also partially funded by the National Institutes of Health under Award Number R01HG007182.

Dedication

To my parents, sisters, mentors, friends, colleagues and everyone who supported me through the years.

Chapter 1: Introduction

1.1 Motivation

A common theme in the introduction of countless papers is the cost of sequencing is decreasing at a rate [1] that exceeds our ability to develop computational hardware to process it (i.e. Moore's Law [2]). This simplistic view is correct in some respects, but realistically the need for better computational methods is more complex than that. That is, under the expectation that data generation is growing exponentially if hardware scales accordingly, one could reason that there would be no need to write new algorithms for common tasks in computational biology. Researchers in the field of bioinformatics would likely say that this is absurd, but each researcher would probably state different reasons as to why it was absurd because the demand for higher performance computational tools in bioinformatics is multifaceted. Finally, because the work presented here contains new data structures and algorithms, these improvements to computational methods and concepts generated in this thesis are not necessarily only useful for the biological sciences but may have other applications in the other domains of computational sciences.

1.1.1 New sequence data is often incompatible with older methods

New technologies often disrupt the current set of algorithms used for sequence analysis. Basically, though powerful in some respects, they do not always bring with them better qualities. The most salient example of this transition is from the Sanger era to the highly parallelized short read era. In this case, response to such a change in the nature of the data meant that new tools utilizing data structures such as FM-Indexes [3] and de Bruijn Graphs (DBG) [4] were developed (for alignment and de novo assembly respectively). We are at another stage again, with error-

prone long read sequencing from Pacific Biosciences (PB) [5] and Oxford Nanopore Technologies (ONT) [6] and linked read technology like 10X Genomics Chromium Sequencing [7].

1.1.2 Computation is not always proportional to the data sizes

Some of the most common computational tasks, if done without heuristics, would not scale with a one-to-one relationship with the data we have generated. For instance, the classic local alignment method - the Smith-Waterman algorithm [8], is by its very nature quadratic to the size of the reference and query (assuming that they are both roughly the same size), so other methods must be applied as data size increases. Algorithm design must continually strive to reduce the time complexity of sequence analysis tasks to ensure that the methods can scale into the future.

1.1.3 Improving methods can facilitate new types of analysis in a multitude of projects

Improving on current methods of sequence analysis will facilitate studies with a large cohort of sequencing experiments like cancer studies within The Cancer Genome Atlas (TCGA) project [9] as well as the analysis of large genomes like the 20Gbp genome of white spruce [10]. Furthermore, streamlining current analysis pipelines and algorithms paves the way for additional applications, like actionable diagnostics of pathogens or mutations present in cancer samples (e.g. the Personal Oncogenomics (POG) Project [11, 12]), which require accurate results within a timely fashion.

1.2 Overview of sequence analysis tasks within thesis

In the field of bioinformatics, the term sequence analysis refers to any analytical analysis that takes as raw input nucleic acid or protein sequence data as input including quantitative measurements [13]. This is, of course, a very broad term, so this research thesis cannot cover all aspects of sequence analysis. The thesis focuses on the application of sequence classification but also touches on other topics such as targeted assembly, *de novo* assembly. Additionally, though this thesis focuses on algorithms that work with Illumina sequencing data, I also touch on other data types, such as linked read data and single-molecule long read data.

1.2.1 Sequence classification

Sequence classification can either refer to a supervised or unsupervised binning of sequences [14]. For the purposes of this thesis, we will focus on the former case, that is classification using previously labelled sequences. Sequence classification to a set of known reference sequences has many applications in contamination screening [15], pathogen detection [16], metagenomics [17], and in the preprocessing for targeted assembly from shotgun sequence data. This thesis focuses on alignment-free methods to minimize computational costs.

1.2.2 Targeted assembly

Targeted assembly refers to the assembly of a subset of your sequences data often guided by known sequences. Such localization can be beneficial to reduce the computational burden compared to that of a whole-genome assembly and can ensure that biologically important sequences (such as gene sequences) are preferentially reconstructed. Targeted assembly technologies were originally designed to reconstruct specific transcript variants, fusion

transcripts or genes from large consortium shotgun data, and have now found applications in human health research [18-20].

1.2.3 *De novo* assembly

De novo assembly refers to the process where DNA sequence read fragments are reconstructed from scratch into a longer sequence and is fundamental to the study the genomes. It remains a highly difficult and computationally intensive problem due to the complexity of sequence data and current limitations of our sequencing technology. Other technologies, such a long read sequencing, as opposed to short-read sequencing, are often hailed as the solution to assembly, but these technologies bring their own unique sets of challenges and problems, which are explored in detail in Chapter 6.

1.3 Background

1.3.1 DNA sequencing

DNA Sequencing is the process of taking physical DNA sequences and using an apparatus to read them into a computer. There are several different technologies capable of doing so however, this thesis will only focus on technologies explored within this thesis: Illumina sequencing [21, 22], long read sequencing such as Oxford Nanopore sequencing and Pacbio sequencing, and 10x Chromium sequencing [23].

The methods explored in this thesis may work on other existing technologies such as Sanger sequencing [24], pyrosequencing [25], SOLiD Sequencing [21], and Ion Semiconductor

sequencing [26], and yet-to-be-invented technologies provided they have similar error rates or biases.

1.3.1.1 Illumina sequencing

Illumina sequencing, formally Solexa [27], is a type of so-called 2nd generation sequencing currently capable of producing many high accuracy paired 150 to 250bp reads with a typical insert length between the pairs 500bp in length. This sequencing technology works by amplifying clusters (in a process called bridge amplification [27]) of short fragments of DNA on a surface and sequentially incorporating a single fluorophore-labeled nucleotide to every DNA molecule in each cluster, taking an image and removing the fluorophore (unblocking the reaction to incorporates another base) and repeating the process. Currently, this technology still dominates the market due to its throughput and low cost [28], as such development of algorithms that work with this data is still very common. Notable properties of this datatype include having low rates of indel errors, Polymerase Chain Reaction (PCR [29])-induced GC biases and read quality drop-offs at one end of the read.

1.3.1.2 10X Genomics Chromium sequencing

10X Genomics Chromium [30] (formally Gemcode) is a library preparation methodology (before sequencing with an Illumina sequencer) that generates linked reads. It works by binding single fragments (range from a size of ten to several hundred kilobases [23]) of DNA to beads that are feed through a microfluidic chip that adds reagents and oil (to encapsulate the reactions). These beads are thermocycled to generating barcoded smaller fragments that are then pooled and sent for sequencing on the Illumina platform.

Unlike Illumina TruSeq Synthetic Long reads (formerly Moleculo) [29], another linked read technology, Chromium reads sample each molecule with less than 1x coverage, and these low coverage reads cannot be used for generating synthetic long reads. Instead, this information simply indicates that some reads of the same barcode are located close to each other on the same DNA strand. Generally speaking, this methodology can be thought as valued added to standard Illumina sequencing, however, in its current form, 16bp (barcode) + 7 bp (spacer) sequences are added to the first read. Chromium data may have additional artifacts introduced during the library preparation stage (such as higher rates of PCR duplicates).

1.3.1.3 Long read sequencing

Considered the 3rd generation of sequencing, long read single-molecule sequencing works by processing a single-molecule of DNA and notably does not employing PCR-like DNA amplification during sequencing. The two major technologies for this are Pacific Bioscience (PB) sequencing [5], and Oxford Nanopore Technologies (ONT) sequencing [6]. These single-molecule long read technologies are characterized with the high error rates [31], in particular for insertions and deletions, and thus methodologies that work on Illumina data may not work on this type of data. More background on these datatypes is explored in Chapter 5.

1.3.2 Relevant data structures

1.3.2.1 Probabilistic data structures

Probabilistic data structures [32] are a class of data structures that focus on representing data approximately, so query operations can sometimes produce an incorrect result. Thus, unlike succinct data structures, these data structures are not lossless, but because they do not represent

data exactly, they have the potential to use even less memory than succinct data structures. Concepts from the domain of succinct data structures may be used in the design and implementation of probabilistic data structures, however, due to the lossy nature of probabilistic data structures, the reverse is seldom true.

1.3.2.1.1 Bloom filters

Bloom Filters are probabilistic, constant-time access data structures that identify whether elements belong to a set [33]. Bloom filters are similar to hash tables, but do not store the elements themselves; instead, they store a bit signature for every element into a common bit array (Figure 1.1). This means that one can represent a sequence with an arbitrary length with a fixed amount of space. The trade-off to this is that the filter may return false membership because of hash collisions in the bit array. The false-positive rate can be managed by increasing the size of the filter.

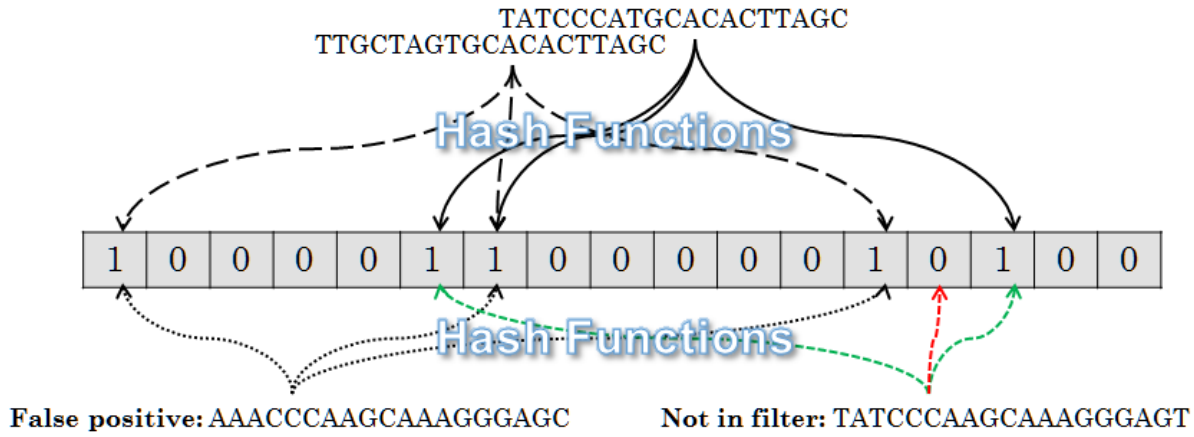


Figure 1.1 A visualization of a Bloom Filter storing k-mers. The hash values of two sequences at the top are inserted into the filter. A false positive is shown on the bottom left and a true negative shown on the bottom right.

There are multiple variants of Bloom filters all that attempt to improve performance or extend the functionality of a basic Bloom Filter that:

- Improve speed (Blocked Bloom filters [34])
- Improve memory (Compressed Bloom Filters [35], COCA Filters [36])
- Allow for counting (Spectral Bloom filters [37], Cascading Bloom filters [38])
- Allow scaling expansion (Bitwise Bloom Filters[39])
- Can handle streaming data (Stable Bloom filters [40])
- Allow the association of values (See Section 1.3.3.2)

1.3.2.2 Key-value probabilistic data structures

Unlike the classical Bloom filter, which only allows for presence-absence queries, hash table-like key-value associations can also be performed probabilistically, and many methods perform this with different characteristics. Bloomier filters are first probabilistic data structures that allowed key-value associations, utilizing perfect hashing to prevent collisions between values [41], though has seen no prominent use in bioinformatics; Marchet *et al* suspect that this may be because no free implementation exists for this data structure as of yet [42]. Bloomier filters have been superseded for applications in bioinformatics by high-performance data structures, such as sequence Bloom trees [43], quotient filters [44, 45], the quasi-dictionary [42, 46], the Othello data structure [47, 48], and interleaved Bloom filters [49].

1.3.2.2.1 Sequence Bloom trees

Sequence Bloom trees are a set of Bloom filters arranged into a binary tree. Each leaf node Bloom filter determines what value the key will map to. There are two recent variants of the

sequence Bloom tree, Allsome sequence bloom trees [43] and split sequence Bloom trees [50]. Both variants improve on the runtime of query and memory of the method, but split sequence Bloom trees use less space than both for the same false-positive rate. Unlike most data structures mentioned here, the lookup is O(log n) where n is the number of values, though negative queries may terminate faster.

1.3.2.2.2 Quotient filters

Quotient filters [44] are like Bloom filters, except they use only one hash function when inserting an element to allow for cache-efficient queries. A “quotient” derived from the key is stored in the hash position. Collisions are mitigated by using linear probing, though false positives can still occur if both the quotient and hash position collide. In the hash position, a value can also be stored and have been used to facilitate k-mer counting [45] and key-value lookup[51]. Due to linear probing, the worst-case time lookup is O(n) depending on the occupancy of the filter, but in practice tend to be constant.

1.3.2.2.3 Quasi-dictionaries

The quasi-dictionary allows for key-value lookups and was originally proposed for applications in metagenomics [46], and uses bbHash [52], a minimal perfect hashing scheme that only requires 3 bits per element. Though this solves issues around collisions of values, false positives are still possible. To reduce false positives a quotient is stored in the position, though no linear probing is needed due to perfect hashing. At suggested parameters, assuming 16-values, each key-value pair inserted will take 3 bbHash bits + 12 quotient bits + 16 value bits = 31 bits per

element. Pufferfish [53], an indexing scheme to store compacted coloured de Bruijn graphs [54], uses a similar data structure that also uses bbHash and works on similar principles.

1.3.2.2.4 Othello data structure

The Othello data structure was originally designed for network forwarding lookup optimization [47], and in the bioinformatics domain, use cases include metagenomics applications [48] and RNA-seq analysis [55]. Othello also bears some resemblance to another probabilistic data structure called Cuckoo filter[56]. Cuckoo filters use two hash functions for two potential hash locations determined by a hashing of the key and the exclusive-or of a quotient derived from the key. If both locations are filled, then existing values can be moved to an empty location using the exclusive-or of the quotient to make room for the value. The Othello data structure also uses two hash positions (split into two tables) to help deal with collisions but does not store a quotient and instead stores exclusive-or of the value to the existing value in the data structure. When a collision occurs that is unresolvable (both hash locations are in use) the element cannot be inserted, resulting in missing elements, though in practice this is a rare event [48]. The memory usage is O(ln), where l is the size of the value type stored and n is the number of elements stored. Othello uses two tables that are at least 1.33n in size, thus assuming 16-bit values, it will have a memory footprint of (16+16)×1.33 = 43 bits per element. The false-positive k-mers (called as "alien k-mers") are mitigated using a sliding window approach where each query considers adjacent k-mers, like the methods presented in this thesis.

1.3.2.3 Succinct data structures

A succinct data structure is a lossless (no information loss) data structure that stores its contents close to its information-theoretic minimum, but still allows for efficient query operations [57]. This concept is commonly associated with efficient access and encoding of bit vectors, graphs, and trees (a famous example is the Compressed Suffix Array used in BWA [58] or Bowtie2 [59]). We note that an extensive exploration of all data structures in this field is beyond the scope of this thesis and to understand this thesis only some subtopics like the efficient access of bit vectors is needed.

1.3.2.3.1 Rank operations on bit vectors

Rank and Select operations on Succinct indexable dictionaries (which can be represented by bit vectors) are the core of many succinct data structures [60, 61]. Conceptually, these operations are important because they enable efficient indexing of succinct data.

Bit array	0	1	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0
Rank	0	-	-	-	1	-	-	-	3	-	-	-	3	-	-	-	4

What is actually stored:	0	1	3	3	4

Figure 1.2 An example of a bit vector with intervals of rank information. The rank can be obtained by taking the modulus 4 of your positions of interest, taking the rank and counting up the remaining filled positions.

The rank at a position is the cumulative count of the filled positions since the start of the bit vector (cf. Figure 1.2 for an example). Select operations are the reverse of rank operations. Given

a count, a select operation will return the position in the array that contains the first position that satisfies the cumulative count of the filled positions since the start of the bit vector.

To access rank information, a naïve method simply scans the vector, taking its sum, which is a linear time operation. A naïve constant time method to access these counts is storing them in another array, however, the memory costs would then be multiplied by a constant. In practice, there are multiple ways to store rank information with minimal memory overhead (cf. Figure 1.2 for an example of one method).

1.3.3 Sequence analysis

1.3.3.1 Sequence classification

As mentioned previously, sequence classification can refer the supervised (using labelled data) or unsupervised classification of sequences [14]. We will be focusing on the supervised methodologies, though these methods can be further subdivided into similarity-based and composition-based methods. Composition-based methods use many properties of the sequence like GC content, and typically utilize machine-learning [62]. However, it has been shown that similarity-based methods have better performance on short read such those produced from Illumina sequencing [62]. Thus, in this thesis, we will focus on similarity-based classification methods for general purpose, sequence-specific classification.

1.3.3.1.1 Alignment-free *k*-mer-based sequence classification

The first methods of sequence classification utilized primarily BLAST [14]. Newer methods would eventually adopt higher performance alignment algorithms and non-alignment based

similarity methods [17]. For the purposes of sequence classification, exact genomic coordinates are often unnecessary and may perform more computation than needed. At present, the fastest methods of sequence classification currently seem to *k*-mer based [15, 17].

K-mer based methods work by treating each *k*-mer in a sequence an element in a set. The elements in this set are compared to a reference set for shared *k*-mers. If a significant number of *k*-mers are found within the reference set, the read is then classified. Unlike alignment, these methods require only the *k*-merization the queried sequences (linear time complexity). The most popular sequence classification tools that utilize *k*-mers are generally specifically designed for metagenomics and tend to use a hash table-based database. Examples of such tools include Kraken [63], and CLARK [64], which both use hash table based databases.

1.3.3.1.2 Bloom filter-based classification

Bloom filter-based methods have a similar theoretical time complexity as hash tables but do not depend on the size of *k.* Thus, Bloom Filters can theoretically use much less memory than hash-based classification methods. Larger *k*-mers can be beneficial as they have higher information entropy and can help increase the specificity of classification.

The use of Bloom filters for sequence classification was originally developed in the tool Fast and Accurate Classification of Sequences (FACS) [65]. We later developed BioBloom Tools (BBT) [15], which used heuristics to optimize speed and reduce the effect of false positives. Though BBT proved effective when using a small number of filters, determining the specific reference of a queried sequence requires the usage of multiple Bloom filters, which can lead to an O(*n*) time

complexity where *n* is the number of references. In this thesis, we will explore a possible means to extend the functionality of Bloom filters for the classification against multiple references.

1.3.3.1.3 Spaced seeds

Spaced seeds are a modification to a standard *k*-mer where some positions on a *k*-mer are set to be “don’t care” or wildcard positions. They were originally used in Patternhunter [66] to improve the sensitivity and specificity of homology search algorithms. They have since been applied to improve the sensitivity of classification tools, which in the past used only *k*-mers such as Seed-Kraken [67] and CLARK-S [68].

Each spaced seed design has different properties and thus performance. Due to the number of different possible spaced seed designs tools such as IEDERA [69] and Rasbhari[70] have been developed, and will consider references used when designing seeds. However, such computation can be costly so it may be beneficial to use Quadratic Residue seeds [71], which have a single design for a set length and still present a good performance. We note methods for designing spaced seeds that work well for homology search, but may not be effective for alignment-free classification [72].

Patternhunter II [73] marked the development of the first multiple spaced seeds, in which multiple spaced seeds were used in a synergistic fashion to further improve the sensitivity of homology search. The design of multiple spaced seeds was shown to be NP-hard, prompting the development of polynomial time algorithms [74] to design them.

1.3.3.2 *De novo* assembly

Strategies for *de novo* assembly have changed over time to improve assembly quality and to adapt to the progression of sequencing data. These methods have progressed from greedy approaches (where sequences were joined based on the highest-scoring overlapping read [75]) to Overlap-layout-consensus (OLC), to de Bruijn graph [4] (DBG) assembly methods.

1.3.3.2.1 Overlap layout consensus

OLC assembly is performed by overlapping all reads into an overlap graph, which is traversed to produce a layout, leading to a final consensus calling step [76, 77]. These methods were first used to assemble Sanger sequencing data but began losing popularity after the development of De Bruijn graph-based methods. More recently, OLC methods have begun to have a resurgence due to the development of single-molecule long read data, which have higher error rates and cannot be assembled effectively with de Bruijn graph-based methods.

1.3.3.2.2 De Bruijn graph assembly

As mentioned, OLC methods require the computation of all read overlaps, which is typically an $O(n^2)$, where n is the number of reads. Short read sequencing had made such method intractable, thus, de Bruijn graphs are employed to the reduce time complexity to $O(n)$. To construct this graph, reads are broken up into k-mers that can be queried for adjacent base extensions so there is no need to explicitly compute overlaps. Hash tables are often used to store the k-mers in the graph [78]. Generally speaking, methods to assemble short reads generally use de Bruijn graph-based methods, but we note that there are methods such as SGA [79] that efficiently use OLC

methodology to assemble short reads. Examples of de Bruijn graph-based assemblers are Velvet [80], ALLPATHS [81] and ABySS [82].

1.3.3.2.3 Targeted assembly

Targeted assembly at its core is simply the process of collecting of sequences from a bait sequence and assembling them. This was first was accomplished with TASR, which collected reads that had matching k-mer content and assembled them [83]. This method was followed by Mapsembler [84], which collects sequences with read alignments and performs a mini-assembly on less mappable sequences to increase the coverage for targeted assembly.

These initial targeted assembly tools, including the more recent aTRAM designed for assembling targets with from different species [85], have limited utility when an incomplete sequence is used to localize reads for assembly, such as using a transcript to reconstruct a whole genic region, and do not typically scale well when given a large (>1000) number of targets.

1.4 Objectives

The overall goals in my work are to explore and develop algorithms to improve the state-of-the-art methods of sequence analysis in the following ways:

1. Explore and evaluate methodologies for sequence analysis on new sequencing technologies such as linked reads and single-molecule long reads
2. Explore, design, and evaluate applications of probabilistic data structures to improve the computational scalability and performance of sequence analysis tasks

3. Explore, design, and evaluate new types of applications of probabilistic data structures in sequence analysis

Chapter 2: Sequence classification using Bloom filters

2.1 Publication note

Most of the work described in this chapter, including benchmarks, was previously published in Bioinformatics in 2014 [15]. The chapter also includes modifications and additions to reflect post-publication work. The tool has also been used in multiple publications and, of note, has been used for pathogen detection in multiple The Cancer Genome Atlas Projects (TCGA) [86-88]. The work described in this chapter is the main work of the author, under the supervision of Inanc Birol. I designed and implemented the tool described, performed the experiments and wrote the manuscript. The co-authors on this work helped test the software, provided valuable feedback during the development of the tool and helped in editing the manuscript.

2.2 Author summary

Large datasets can be screened for sequences from a specific organism, quickly and with low memory, by a data structure that supports time- and memory-efficient set membership queries. Bloom filters have such operations but require that false positives be controlled. We present a Bloom filter-based sequence-screening tool called BioBloom Tools that is faster than BWA, Bowtie 2 and FACS, delivers sensitivity and specificity comparable to these tools, controls false positives, and has low memory requirements. This chapter addresses objective 2 in section 1.4. The source code and user manual for BBT are available through www.bcgsc.ca/platform/bioinfo/software/biobloomtools. The software is released under the GNU General Public License 3.0 (GPL3).

2.3 Introduction

Pipelines that detect pathogens and contamination screen for host sequences so they do not interfere with downstream analysis [89-92]. The alignment-based algorithms these pipelines use provide mapping locations that are irrelevant for classification, so perform more computation than is needed. To address this, we have developed BioBloom Tools (BBT).

BBT utilizes Bloom filters – probabilistic, constant time access data structures that identify whether elements belong to a set [33]. Bloom filters are similar to hash tables, but do not store the elements themselves; instead, they store a fixed number of bits for every element into a common bit array. Thus, they use less memory, but queries to the filter may return false membership (hits) due to hash collisions in the common bit array. The false-positive rate (FPR) can be managed by increasing the size of the filter (Section A.1). Using Bloom filters for sequence categorization was pioneered by the program FACS [65]. Here, we describe a Bloom filter implementation that includes heuristics to control false positives and increase speed.

2.4 Methods

We first build filters from a set of reference sequences by dividing the sequences into all possible *k*-mers (sub-strings of length *k*). We compare the forward and reverse complement of every *k*-mer and include the alphanumerically smaller sequence in the filter. We calculate the bit signature of a *k*-mer by mapping the sequence to a set of integer values using a number of hash functions [93]. The bitwise union of the signatures of all the *k*-mers constitute a Bloom filter for the corresponding reference sequences.

To test whether a query sequence of length l is present in the target reference(s), we use a sliding window of k-mers. Starting at one end of the query sequence and shifting one base-pair at a time along this sequence, we check each k-mer against each reference's Bloom filter. When a k-mer matches a filter, we calculate a score:

$$s = \frac{1}{l-k}\sum_{i=1}^{c}(a_i - 0.5)$$

where a_i is the number of adjacent filter-matching k-mers and c is the current stretch of contiguous matches This heuristic penalizes likely false positive hits. We evaluate k-mers this way until we reach either a specified score threshold (s^*) or the end of the query sequence. If at any point we reach s^*, we categorize the query as belonging to the reference and terminate the process for that query. Further, we employ a jumping k-mer heuristic that skips k k-mers when a miss is detected after a long series of adjacent hits. This efficiently handles cases in which the query has a single (or a few) base mismatch(es) with the target.

2.5 Benchmarking

In our paper, we compared BBT against two widely-used Burrows-Wheeler Transform-based alignment tools that have low memory usage and high accuracy: BWA [58] and Bowtie 2 (BT2) [59], and against the C++ implementation of FACS (https://github.com/SciLifeLab/facs). The version of BBT used in these tests was 2.0.4. Unless otherwise stated, all tests were run on a Red Hat Enterprise Linux Server, Linux version 2.6.18-164.el5, using a 12×2.67GHz X5650 Intel Xeon processor with 48GB of memory. We used BWA version 0.7.5a-r428, Bowtie2 version 2.1.0, and FACS version 2.0 for all tests. Because BWA aln requires multiple processes for paired end reads, each process was run sequentially, and the CPU times were totalled afterwards.

2.5.1 Benchmarking on simulated data

We used dwgsim (https://github.com/nh13/DWGSIM) to generate simulated Illumina reads from human, mouse and *E. coli* reference genomes. For each genome, we generated 1 million 2×150bp paired-end (PE) reads and 1 million 100bp single-end (SE) reads. We used *E. coli* because it is a common contaminant and is genetically distant from human. With the mouse dataset, which is commonly used in xenograft studies., we tested categorization accuracy for species that are closely related genetically.

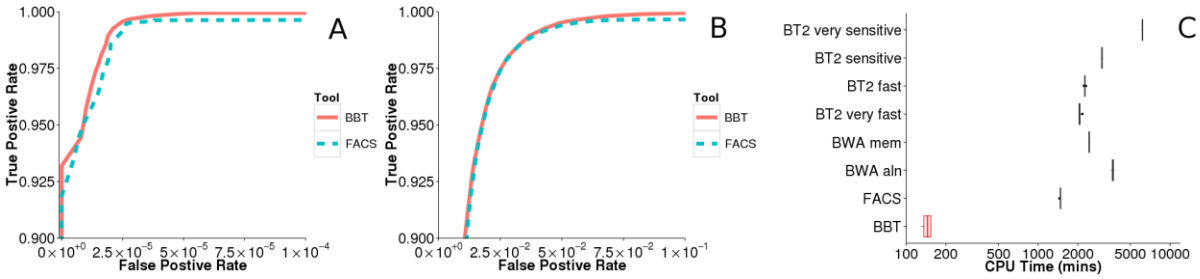


Figure 2.1 Performance comparisons of BBT against FACS, BWA and BT2. Receiver operator characteristic curves between BBT and FACS using simulated *H. sapiens* and (A) *E. coli* and (B) *M. musculus* 100bp SE reads filtered against a *H. sapiens* Bloom filter using a *k*-mer size of 25bp; (C) CPU time benchmark comparing BT2 (for a range of built-in settings), BWA (using aln and mem settings) and BBT (using *s = 0.1), using one lane of human 2×150bp PE Illumina HiSeq 2500 reads.**

2.5.1.1 Sensitivity and specificity comparison to FACS

Because FACS does not support PE reads we used the 100bp SE reads, we tested a range of scoring thresholds for both FACS and BBT. Using a *k*-mer size of 25bp, BBT generally matched or outperformed FACS (Figure 2.1 A-B). For both tools, increasing the *k*-mer length gave higher specificity but lower maximum sensitivity (Figure A.1 and A.2).

2.5.1.2 Sensitivity and specificity comparison to BWA and BT2

We used the 2×150bp PE reads for this test. Overall, BBT performed comparably to the aligners, and outperformed BT2 *fast* and BT2 *very fast* in both sensitivity (as 1 – sensitivity) and specificity (**Table 2.1**).

Tool & Settings	1 - Sensitivity (*H. sapiens*)	FDR (*M. musculus*)	FDR (*E. coli*)
BT2 very sensitive	1.40×10^{-5}	2.03×10^{-2}	**0**
BT2 sensitive	7.52×10^{-4}	9.08×10^{-3}	**0**
BT2 fast	1.26×10^{-2}	5.90×10^{-3}	**0**
BT2 very fast	1.34×10^{-2}	5.65×10^{-3}	**0**
BWA aln	3.26×10^{-3}	**8.14×10^{-4}**	**0**
BWA mem	**0**	1.92×10^{-1}	1.00×10^{-4}
BBT (*s**=0.1)	8.42×10^{-3}	3.78×10^{-3}	**0**
FACS^{+}	1.22×10^{-1}	9.88×10^{-3}	**0**

Table 2.1 Benchmarking results of BBT against BWA and BT2 using simulated paired end 2×150bp reads.

$^{+}$All reads were treated as SE reads.

2.5.2 Benchmarking on experimental data

We used a single lane of 2×150bp PE human DNA reads (https://basespace.illumina.com/run/716717/2x150-HiSeq-2500-demo-NA12878) generated with an Illumina HiSeq 2500 sequencer to benchmark computational performance. For a controlled

comparison, we ran at least 8 replicates for each tool, we measured CPU time, and set all applications to use a single thread.

2.5.2.1 Performance comparison to BWA and BT2

We ran BBT with s*=0.1 and compared it to BWA and BT2, using a range of run modes for the latter two tools. BBT was faster than the fastest aligner/settings combination (BT2 very fast) by at least an order of magnitude (Figure 2.1C). The mapping rates (categorization rates for BBT and FACS) of each tool were comparable, at 96.69% (BT2 very sensitive), 96.57% (BT2 sensitive), 96.18% (BT2 fast), 95.97% (BT2 very fast), 99.76% (BWA mem), 95.12% (BWA aln), 95.81% (FACS) and 97.27% (BBT).

2.5.2.2 Memory usage

For categorization, using the human reference and simulated reads, the peak memory usage for each tool was 3.8GB (BBT), 4.8GB (FACS), 3.1GB (BWA aln), 5.2GB (BWA mem) and 3.4GB (BT2). Note that these figures are for categorization only, and do not include the memory usage for creating the FM-indexes and Bloom filters. Unless slower disk-based methods are used, creating an FM-index takes at least O($n$log(n)) bits of memory, where n is the size of the reference sequence [94]. In contrast, Bloom filter memory usage for both creation and categorization are O(log(1/f)n), where f is the FPR and n is the number of input sequences. We used 5182 bacterial sequences (6×10^{10} unique 25-mers) to demonstrate the storage capacity of a Bloom filter, resulting in a 6.8 GB filter with an FPR of 0.75%.

We created filters with sizes of 3.2 GB for both FACS and BBT. Assuming optimal numbers of hash functions are used, filters with the same size should have similar FPRs. However, we observed that we had to use different FPR settings in creating these filters (FPR of 0.5% for FACS and 0.75% for BBT). We hypothesize that the tools differed from theoretical estimates because of implementation-specific calculation differences.

2.6 Discussion

The use of this tool is justified not only by its computational performance but the reproducibility and simplicity of mapping. Unlike mappers like BWA, which will be affected by the concept of mapping quality, a metric based on the probability of a match given the uniqueness of sequences in an indexed reference, BBT only considers the score when assigning mapping results, and only concerns itself with the count of k-mers relative to the length. This means that if the same filter is used with multiple targets the mapping results will be the same regardless. In the case of BWA, adding more reference will change the mapping quality, and may lead to different mapping results even if the same reference was used, when mapping to multiple references. In some regards, if a single best match is desired, the concept of mapping quality makes sense, and will likely yield better results; however, for tasks relating to classification mapping quality often leads to confusions in mapping result interpretations and reproducibility. The scoring metric used here is a heuristic attempted to balance classification sensitivity with likely false positives and homology. However, more principled probability-based classification is possible, as we will see in Chapter 4.

Chapter 3: Kollector: transcript-informed, targeted *de novo* assembly of gene loci

3.1 Publication note

Most of the work described in this chapter was previously published in Bioinformatics in 2017 [95] The publication was a joint effort between myself and Erdi Kucuk (co-first authors), under the supervision of Inanc Birol. I designed and implemented the machinery needed to perform read collection and progressive Bloom filter creation, aided in experimental design, and wrote and revised much of the manuscript. Erdi Kucuk implemented the overarching Kollector scripts, performed the experiments, and co-wrote the initial draft of the manuscript. Other co-authors on this work provided valuable feedback during the development of the tool and helped in editing the manuscript.

3.2 Author summary

Despite considerable advancements in sequencing and computing technologies, *de novo* assembly of whole eukaryotic genomes is still a time-consuming task that requires a significant amount of computational resources and expertise. A targeted assembly approach to perform local assembly of sequences of interest remains a valuable option for some applications. This is especially true for gene-centric assemblies, whose resulting sequences can be readily utilized for more focused biological research. Here we describe Kollector, an alignment-free targeted assembly pipeline that uses thousands of transcript sequences concurrently to inform the localized assembly of corresponding gene loci. Kollector robustly reconstructs introns and novel sequences within these loci and scales well to large genomes – properties that make it especially

useful for researchers working on non-model eukaryotic organisms. We demonstrate the performance of Kollector for assembling complete or near-complete *Caenorhabditis elegans* and *Homo sapiens* gene loci from their respective, input transcripts. In a time- and memory-efficient manner, the Kollector pipeline successfully reconstructs respectively 99% and 80% (compared to 86% and 73% with standard *de novo* assembly techniques) of *C. elegans* and *H. sapiens* transcript targets in their corresponding genomic space using whole-genome shotgun sequencing reads. We also show that Kollector outperforms both established and recently released targeted assembly tools. Finally, we demonstrate three use cases for Kollector, including comparative and cancer genomics applications. This chapter addresses objective 2 and 3 in section 1.4.

3.3 Introduction

Production of high-quality reference genome sequences for non-model organisms remains a challenging undertaking, especially for large (>1 Gbps) genomes. For such projects, *de novo* whole-genome assembly typically requires billions of sequencing reads from several different types of DNA libraries. Processing these large volumes of data, and using them to assemble a genome, usually necessitates access to a high-performance computing environment, significant expertise, and specialized software [96]. An attractive alternative for generating reference sequences can be achieved through the targeted assembly of gene/transcript sequences of interest. Even for species with scant transcriptomic sequence information, there are likely existing sequences that could be used to aid *de novo* assembly, such as homologous gene sequences from a related organism. Utilization of these data helps to localize the assembly problem and ensures that the desired sequences (e.g. genic regions) are fully reconstructed. A favourable consequence of this localization is a reduction of the complexity and computational

cost relative to that of a whole-genome assembly. In practice, however, the computational cost of identifying reads related to target sequences has remained challenging due variation and novel sequences not found within a target.

The first solution for the reconstruction of specific targets was accomplished with TASR, an alignment-free targeted *de novo* assembly software [83]. This method was followed by Mapsembler [84], which uses read alignments to guide the process, and presented a more memory-efficient and faster alternative. These pioneering targeted assembly technologies were originally designed to reconstruct specific transcript variants, fusion transcripts or genes from large consortium shotgun data, and have now found applications in human health research [18, 19]. These methods are notably very efficient as they pair sequence recruitment with built-in *de novo* assembly algorithms that utilize internal data structures directly. Unfortunately, these targeted assembly tools have limited utility when an incomplete sequence bait is used to localize reads for assembly, such as using a transcript to reconstruct a whole genic region or the use of homologous, yet divergent, sequences as bait.

To reconstruct incomplete regions, most modern methods utilize an iterative read recruitment process to fill in gaps. MITObim [97], GRAbB [98], and aTRAM [85] recruit reads based on sequences derived from reads recruited in earlier iterations to extend novel regions of previously incomplete sequences. MITObim was designed to assemble mitochondrial sequences and works by recruiting reads that share a 31-mer (subsequence of length 31) in common with the target, cycling through the read set multiple times until the target is reconstructed. GRAbB works in a similar fashion, however, it is designed to recruit reads for multiple targets at a time, and thus

had been shown to computationally out-perform MITObim [98]. Finally, aTRAM is designed for assembling orthologs from a related genome and utilizes BLAST [99] to index portions of the sequence reads so multiple iterations do not require multiple passes through the raw reads during the recruitment process at the cost of higher memory usage. For each of these tools, after each cycle of recruitment, an assembly using established assembly tools (e.g. Velvet; [80]) is performed and fed into later iterations to extend the bait sequence.

Advances in RNA-Seq technology and *de novo* assembly tools [100-102] have made high-quality transcriptomes from non-model organisms increasingly available, which can provide a valuable resource to inform the targeted assembly of genic regions. In this chapter we describe Kollector, an alignment-free targeted assembly pipeline that can use whole transcriptome assemblies to filter and classify whole-genome shotgun sequencing reads, thereby localizing the *de novo* assembly of corresponding genic loci. The pipeline collects genomic reads related to target loci using a novel data structure called progressive Bloom filters implemented within BioBloom Tools (BBT) [15] and assembles them with ABySS [82], a de Bruijn graph [4] short read *de novo* assembler. Kollector is able to expand intronic regions iteratively, but in practice requires a fewer number of iterations than previous methods by greedily populating progressive Bloom filters. We demonstrate efficient targeted assembly of *Caenorhabditis elegans* and *Homo sapiens* genes by Kollector, and its relative effectiveness compared to four published targeted assembly tools. We also show applications of Kollector in comparative and cancer genomics use cases.

3.4 Methods

3.4.1 Progressive Bloom filters

Bloom filters are memory-efficient probabilistic data structures with tunable false positives rates [33]. They are effective in storing sequences for use in fast, specific and sensitive sequence classification [15, 65]. This works by shredding (*k*-merizing) a reference sequence, such as transcript sequences, into its constituent *k*-mers (subsequences of a uniform length *k*) and seeding a Bloom filter with these *k*-mers. This seeding sequence is also referred to as a bait sequence. One can then query the Bloom filter using *k*-mers derived from whole-genome sequence reads, and test for sequence similarity at a given threshold. Within this chapter, our match criteria are based on a similarity score threshold (a value between 0-1) described in the BioBloom Tools (BBT) manuscript [15]. Reads/queries are then categorized as match or no match.

Implemented within BBT, we introduce efficient read recruitment using a novel data structure called a *progressive Bloom filter*. In a progressive Bloom filter, we scan a read set and greedily add additional *k*-mers from matching reads (and their pairs, when available) after each filter query. Because the sequence similarity/*k*-mer overlap threshold (*r* parameter) can result in partial overlaps, it allows for the addition of new *k*-mers in the Bloom filter. Consequently, the contents of our Bloom filter will expand into genomic regions not found in our original seed sequences, such as intronic regions. This method works particularly well when read pairs are used, by incorporating the entire *k*-mer content of a read when its pair registers a positive match (Algorithm B.1). After the *k*-mer space is expanded sufficiently, the resulting Bloom filter can be used to again scan through the reads to recruit all relevant reads within the expanded regions of interest.

3.4.2 Kollector pipeline

In Kollector we first use BBT in progressive Bloom filter mode seeded with input transcript (target) sequences. We scan a set of genomic reads for reads pairs that share a user-defined amount of *k*-mer overlap based on the (*r* parameter) referred to as the tagging stage of Kollector. The threshold parameter *r* should remain high (0.5 to 0.9) to minimize off-target *k*-mer recruitment, but in the case of high read error rates, divergent or low coverage regions, a low value of *r* (0.2 to 0.5) might be used instead. Read tagging continues until the progressive Bloom filter reaches a user-defined maximum number of *k*-mers (*n* parameter), or until all the genomic reads are processed. The *n* parameter determines the maximum number of *k*-mers the progressive Bloom filter can contain to maintain a maximum FPR before the second stage mentioned below. The maximum FPR is determined by the *f* parameters in BBT, set to 0.001 by Kollector. In practice, *n* should be set to the maximum expected size of the total genic space being reconstructed. At this FPR the memory usage of the progressive Bloom filter will not exceed 15*n* bits.

A real-data illustration of this process for a single *C. elegans* transcript C17E4.10 is shown in Figure 3.1. As one would expect, initially tagged reads are derived from exonic regions, but as those reads are added to the filter it allows for reads from intronic regions to be tagged as well (from recruiting their pair) until reads spanning the entire gene have been added to the filter. This progressive Bloom filter is used to recruit reads that share a *k*-mer overlap with the filter based on the length of the read (*s* parameter, defined in BBT [15]) within the entire read set in the second stage of the pipeline (all recruited reads in Figure 2.1). The threshold parameter *s* uses the

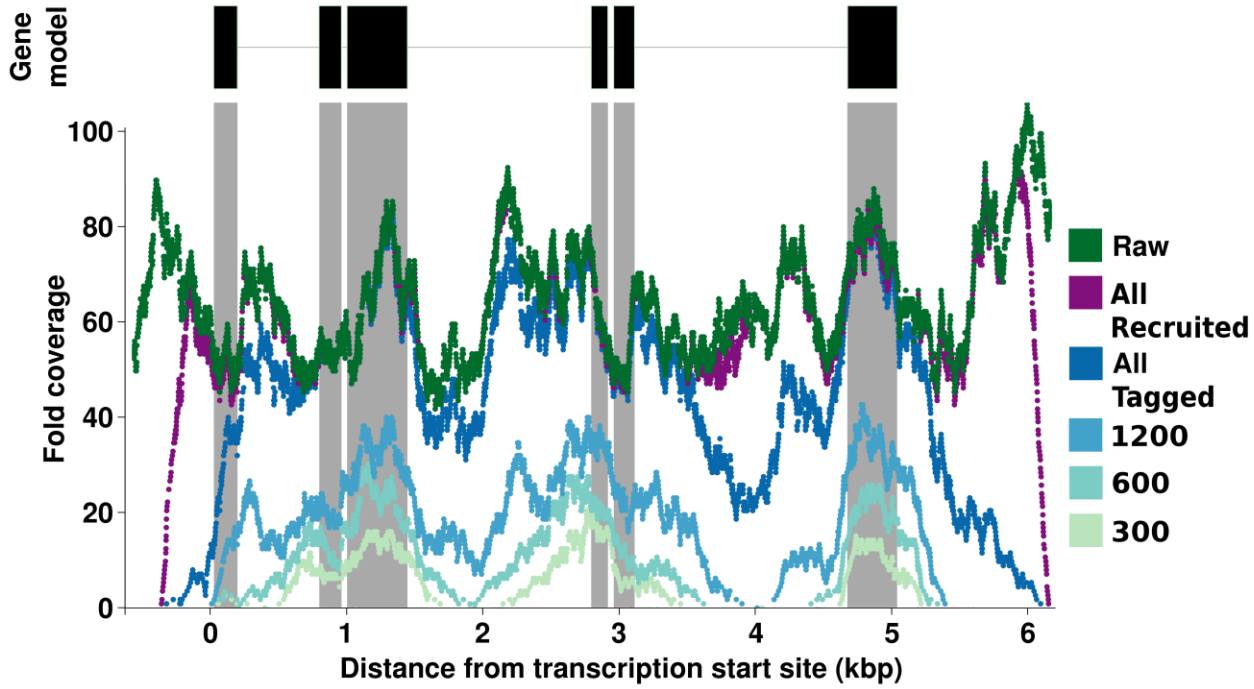


Figure 3.1 Genomic read tagging by progressive Bloom filter. Fold coverage of tagged reads is shown at points in time (first 300, 600 and 1200 and all tagged reads) during a single run with the *C. elegans* C17E4.10 gene as the target. The raw read coverage is indicated solely as a baseline, to show the tagging process of the progressive Bloom filter. The final recruited coverage is shown in purple after the second stage of Kollector has collected all the reads. The fold coverage was calculated after read alignment to the reference transcript. In the Gene Model track, the black rectangles depict the exons and the connecting grey line depicts the introns.

same similarity metric as *r* when evaluating reads, however, *s* is not as critical since off-target read classifications will not propagate and can remain safely set to values between 0.2 to 0.5. In the third stage, the recruited reads are assembled with ABySS (v1.5.2; np=12 CPUs, k=96), and finally, the input transcripts are aligned to the assembly with GMAP (version 2016-05-01; t=10) [103] to report assembled scaffolds that contain the targeted loci. The peak memory of

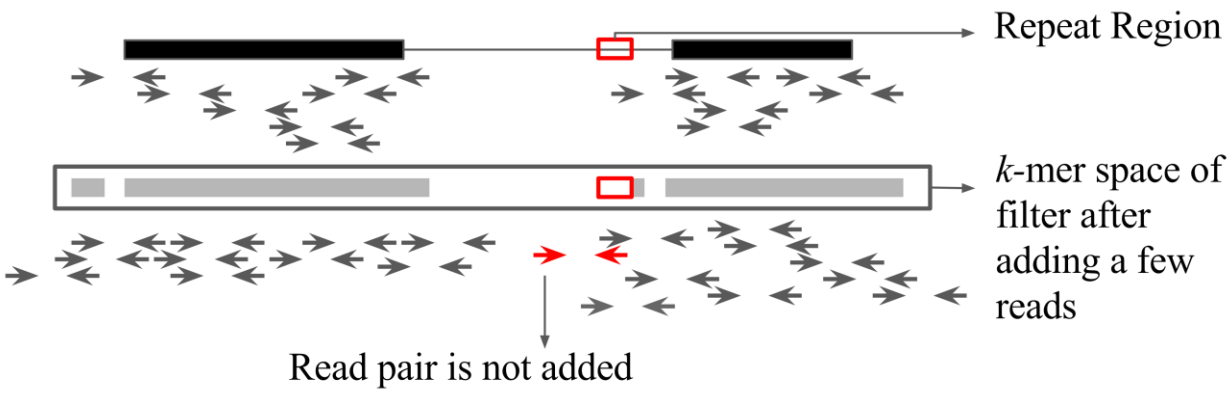


Figure 3.2 Illustration of read recruitment using a Progressive Bloom Filter. The black bars depict *k*-mers derived from the input transcripts. The grey bars depict the *k*-mer space after a few reads have been tagged and their constituent *k*-mers added to the Bloom filter.

Kollector is dominated by the downstream assembly algorithm used. By default, our pipeline uses ABySS 1.5.2, but in principle, other assembly algorithms may be used as well.
This pipeline can also be run in an iterative mode. Un- or partially assembled targets from an earlier iteration may be selected as the input, along with other targets, and fed back to the pipeline, as the read localization process and the resulting de Bruijn graph complexity are dependent on the context represented in the Bloom filters. Therefore, iterations may result in successful reconstructions for targets that had failed previous attempts.

To improve the performance of Kollector in complex genomes, BBT can also take a Bloom filter of repeat sequences as an additional input and use it to tag repeats while extending the progressive Bloom filter. Sequences that are tagged as repeats are not used for the expansion of the *k*-mer space within the filter, thus preventing the recruitment of off-target regions (Figure 3.2).

#	Species	Datatype	Read Lengths	Total Bases	Raw Cov.	Source
1	*C. elegans*	WGS	110bp	7.5Gbp	75x	SRA Accession: DRR008444
	C. elegans	Transcripts	-	27Mbp	-	RefSeq mRNA (>1kb)
2	*H. sapiens*	WGS	250bp	229Gbp	70x	SRA Accession: ERR309932
	H. sapiens	Transcripts	-	138Mbp	-	TCGA barcode: 22-4593-01 assembled using Trans-ABySS (v1.5.1; k = 42)
3	*P. glauca*	WGS	150-300bp	1.2Tbp	48x	SRA Accession: SRR1982100
	P. glauca	Transcripts	-	23Mbp	-	Genome Annotation: GCA_000966675.1 (high confidence genes)
4	*P. schaeffi*	WGS	100bp	12.8Gbp	128x	SRA Accession: SRX390495
	P. Humanus	Transcripts	-	2.44Mbp	-	Dryad DOI: http://dx.doi.org/10.5061/dryad.9fk1s
5	*M. musculus*	WGS	150bp	116Gbp	41x	SRA Accession: SRX1595526
	H. sapiens	Transcripts	-	57Mbp	-	Ensembl bioMART
6	*H. sapiens*	WGS	100bp	13.8Gbp	4x	TCGA barcode: TCGA-BA-4077 (subset)
	HPV 16	Ref. Genome	-	8Kbp	-	Papillomavirus Episteme

Table 3.1 Datasets used in Kollector targeted assembly experiments and comparisons

3.5 Results

We tested our tool on a variety of datasets (Table 2.1), and within different targeted assembly contexts to show the utility and efficacy of Kollector. All benchmarks described below were

obtained using a single high-performance computer with 48 GB of RAM and 12 dual Intel Xeon X5650 2.66 GHz CPUs.

3.5.1 Targeted gene assembly using transcript sequences

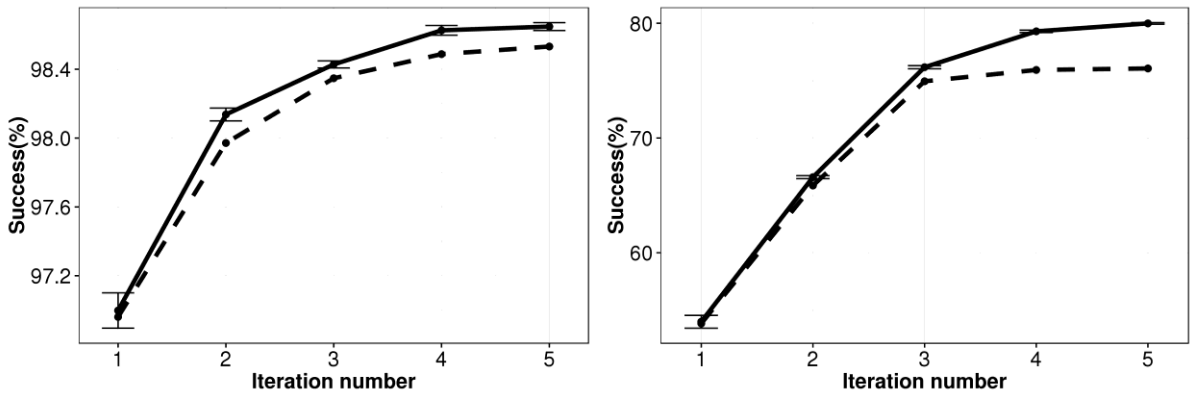
3.5.1.1 Kollector assemblies

Kollector performs incrementally better when used iteratively, where genes not assembled in initial stages are provided as input for the next iteration. After each iteration, the target transcript sequences are aligned to the Kollector output with GMAP [103], and those that have a unique match to a single genomic contig along a certain sequence identity (default = 90%) are deemed to have been successfully reconstructed. Transcripts that do not satisfy this criterion are re-tried in the subsequent iteration by setting a lower specificity for sequence selection during the read collection phase. This is achieved by lowering the *r* parameter in each iteration (e.g. 0.90, 0.70, 0.50, 0.30, 0.20), while keeping the other parameters constant (s = 0.50, n = 100,000,000 in our experiments).

Before running Kollector on *H. sapiens*, we randomly divided the transcriptome (Table 3.1, #2) into five bins of ~10k transcripts each to prevent the memory usage of ABySS v1.5.2 from exceeding 48GB of RAM. Each bin is used to initiate a single progressive Bloom filter in a separate Kollector run with a *k*-mer cap (-n) of 100,000,000. Each bin took 29 GB of RAM and completed within 12 h. Isoforms split across different bins may assemble the same genic region multiple times. Similarly, isoforms within the same bin will help reinforce the assembly of their common target locus from the improved read tagging. Highly similar transcripts originating from

multi-copy genes may be collapsed into a single sequence, however, the extent of this depends on the assembly algorithm used by Kollector.

Because progressive Bloom filters are also sensitive to the order of reads in the input file, Kollector has the option to shuffle the genomic reads before each iteration to reduce any potential bias created by read order (Figure 3.3). In our tests, read shuffling led to significant gains in the overall assembly success for both species, reaching 98.7% in *C. elegans* (13,378 out of 13,556 assembled) and 80.1% (41,631 out of 52,006 assembled) in *H. sapiens*.



(A) **(B)**

Figure 3.3 Performance of Kollector for assembling target sequences in (A) *C. elegans* and (B) *H. sapiens*. Genomic reads were randomly shuffled prior to being provided as input to the pipeline (solid lines, mean of 10 independent experiments). For comparison, runs with no shuffling are also depicted (dashed lines). The success rate was calculated as the proportion of target genes successfully assembled over five iterations.

To determine the efficacy of our targeted approach compared to a regular *de novo* assembly in reconstructing the genic space, we performed a whole-genome assembly of the *C. elegans* and *H. sapiens* datasets discussed above. We used ABySS v1.5.2 to assemble both genomes, utilizing

the same assembly *k*-mer size used within our targeted assemblies. This value of *k* was used because it yielded the highest N50 in a set of *k*-mer sweeps on the whole-genome assembly. We found that only 85.7% and 72.9% of all genic regions were completely captured (within a single contig) in the *C. elegans* and *H. sapiens* whole-genome *de novo* assemblies, respectively. Compared to 98.7% and 80.1% with Kollector for *C. elegans* and *H. sapiens*, respectively, these results show that performing targeted assembly increases reconstruction contiguity in genic regions. In addition, each complete whole-genome assembly required more computational resources than Kollector, requiring 3 hours with 8 GB (single node) and 41 hours with 857 GB (18 nodes) for the *C. elegans* and *H. sapiens* assemblies, respectively.

We investigated the failed reconstruction of 199 *C. elegans* transcripts (unshuffled experiment, Figure 3.3A dashed line), and found that the failed targets had on average a longest intron length significantly larger than the successfully assembled ones ($p=1.5x10^{-8}$; Student's t-test; Figure 3.4A), indicating that the failure was due in part to the lengths of the longest introns. For targets with maximum intron lengths of approximately 20 kbp, we expect 50% of the reconstructions to fail. However, we note that these large intron genes make-up a very small proportion of the total dataset (Figure 3.4B). Although the distributions suggest substantial overlap, and there were many long targets with successful assembly, with lengths comparable to the failed targets, our statistical test suggests that Kollector has a bias towards assembling smaller genes, likely due to the challenge of identifying enough genomic reads to connect exons separated by long introns. We expect that the use of longer reads and insert sizes (possibly mate-pair reads) could help alleviate some of these issues.

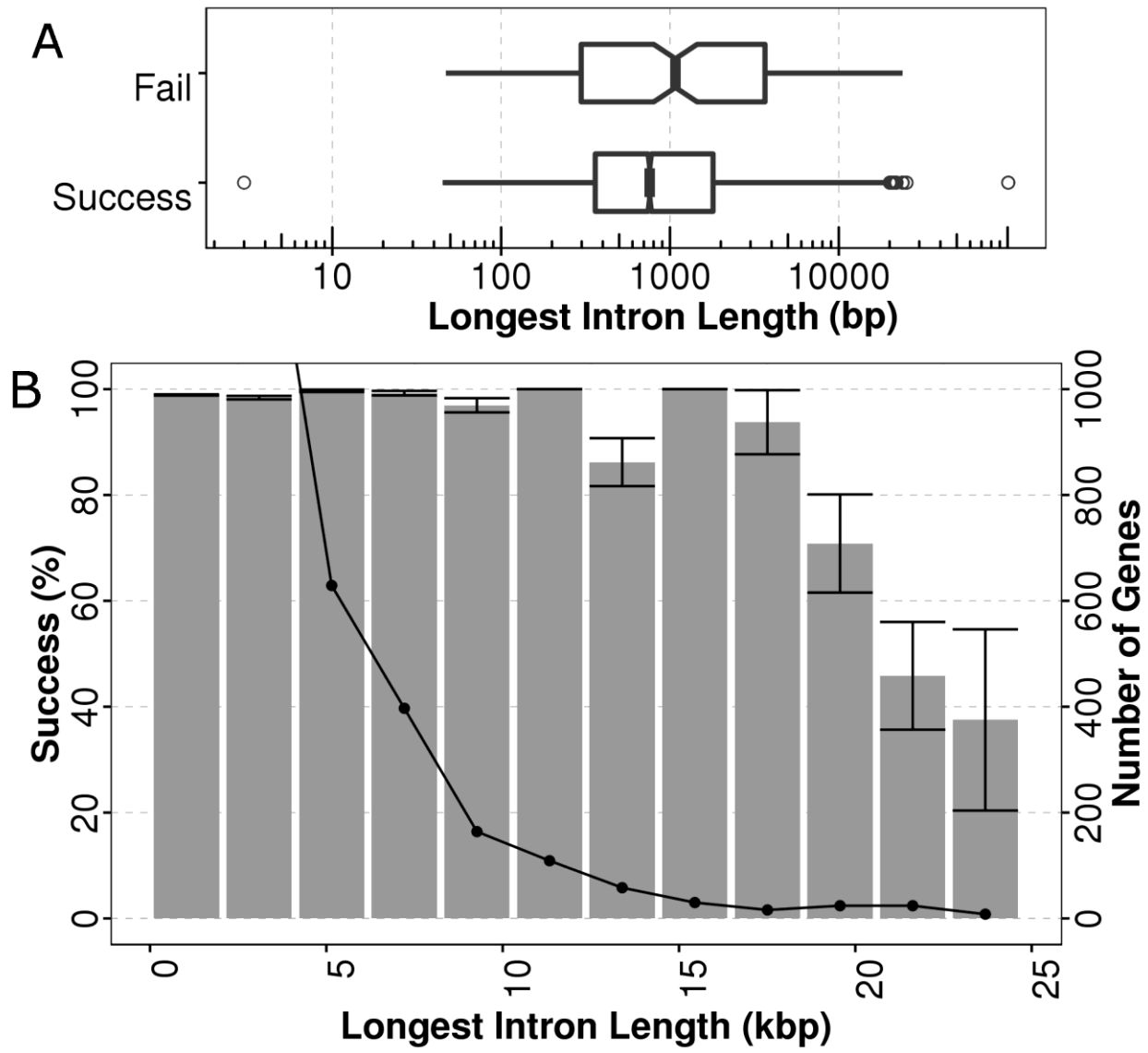


Figure 3.4 A) Longest intron length comparison between the *C. elegans* target genes that are successfully assembled (top) vs. those that failed (bottom). Notches in the boxes represent a 95% confidence interval around the median. The length difference between the two groups, represented on a logarithmic scale on the x-axis, is found to be statistically significant by t-test ($p=1.5x10^{-8}$). This analysis yields the same result for gene length comparison (Figure B.2) B) Proportion of successful gene assemblies vs longest introns (bars) with the number of gene in each bin (lines). The first and second bins make up 10,237 and 1,496 genes respectively and make up 86% of all genes in the dataset (Figure B.1).

Species	Number of Genes Assembled	Number of Genes Aligned	Success Rate
C. elegans	13,378	13,356	99.8
H. sapiens	41,631	41,525	99.7

Table 3.2 Accuracy of Kollector-assembled genes

To evaluate the accuracy of the genomic contigs produced by Kollector, we aligned the output of a Kollector run from both species to their respective reference genome with BLASTn [104], calling a correct alignment at a threshold of 95% query coverage and 95% sequence identity. Doing so, we find that 99.7% or more of the assembled genes satisfy these criteria in both species (Table 3.2).

Due to the greedy nature of our algorithm, it is possible to recruit off-target regions. The primary source of error is due to repeats, with minor contributions from false positives and read errors. We investigated these off-target events by aligning *H. sapiens* transcripts to the genome. We extracted the regions based on the start and end coordinates of all contigs produced by Kollector that aligned. We found that for contigs >500bp, only 40% were on target. Though this may seem low, this assembly still represents a significantly smaller subset of the whole-genome, and this analysis does not take into account contigs generated directly upstream of each gene.

3.5.1.2 Comparisons against Mapsembler, TASR, aTRAM and GRAbB

Although not originally and specifically designed for reconstructing genomic loci from transcript sequences, we tested Mapsembler2 (v2.2.4) and TASR (v1.5.1) using the *C. elegans* genomic dataset and input transcripts. For Mapsembler2, we used the default *k*-mer size ($k = 31$) and the

consensus sequence mode (-i 2), which means extensions of the target sequence are collapsed. We ran TASR using the independent, *de novo* assembly mode (-i 1) with default parameters, meaning targets are only used for read recruitment and not for seeding the assembly. We evaluated the results of these tools with the same GMAP alignment criteria that we reported for Kollector. In our tests, Mapsembler2 and TASR assembled 28% and 18% of the targets, respectively (Table 3.3). In contrast, Kollector was able to assemble 97% of the target gene set in the first iteration alone. This is because Mapsembler2 and TASR are designed for re-assembling input targets within their respective sequence boundaries, with limited extension capability in the flanking regions, limiting their utility beyond single-exon genes. Kollector's progressive filtering algorithm, unlike the former approaches, can incorporate reads derived from intronic regions for assembly, as discussed.

We also tested iterative read recruitment methods aTRAM and GRAbB on a random subset of 1000 transcripts from our *C. elegans* dataset. A smaller subset was chosen as the methods proved intractable on larger datasets. To complete the computation of GRAbB, it requires stopping conditions such as a minimum length of assembly for each target. We provided GRAbB with the exact genomic lengths specified by the reference annotations. Despite this, we were unable to finish computation of GRAbB in a tractable amount of time, and instead took the intermediate results of the method after running GRAbB for 48 hours. We found that on these read subsets, aTRAM and GRAbB assembled 73.1% and 42.5% of the targets respectively. Despite the smaller size of the subset, Kollector outperformed both tools in speed, while utilizing a comparable amount of memory.

Method	Number of targets attempted	Number of targets assembled	Percentage of targets assembled	Wall Clock Time (h)	Peak Memory (GB)
Kollector	13,556	13,144	96.96	2	4.8
Mapsembler2	13,556	3,742	27.60	2	9.3
TASR	13,556	2,418	17.83	3	15.6
aTram	1,000	731	73.1	38	2.4
GRAbB	1,000	425	42.5	48*	3.1

Table 3.3 Comparison with Mapsembler2 and TASR

3.5.1.3 Scaling to large genomes: gene assembly in white spruce

Assembling complex and very large eukaryotic genomes is a computationally demanding process. Therefore, a targeted assembly approach for retrieving biologically important gene sequences remains an attractive option for most researchers. We have tested such a use case for Kollector using *Picea glauca* (white spruce), which has a genome size in the 20 Gbp range [105, 106]. For the species, Warren and colleagues derived a high confidence set of 16,386 gene predictions and associated transcripts. Using these transcript sequences, Kollector was able to reconstruct 13,277 (80.4 %) of the original target genes in a single iteration, requiring five processes, each using 43.3 GB of RAM and running for 24 hours.

Researchers are often also interested in the regions immediately upstream and downstream of genes, which contain promoters and other regulatory elements. Due to the nature of the progressive filtering algorithm, Kollector assemblies may extend into these regions. In order to

demonstrate this, we aligned the aforementioned high-quality transcript models to the resulting Kollector assemblies and quantified the amount of sequence upstream and downstream with respect to the transcript. We show that, in addition to a gene's exonic and intronic sequences, Kollector typically reconstructs approximately 1 kb of sequence beyond the 5' and 3' ends of the target transcript (Figure 3.5). Such extensions would enable characterization of the regulatory factors and complexes in the proximal promoter of genes of interest by chromatin immunoprecipitation and would be especially empowering to studies of non-model organisms without available reference genome sequences.

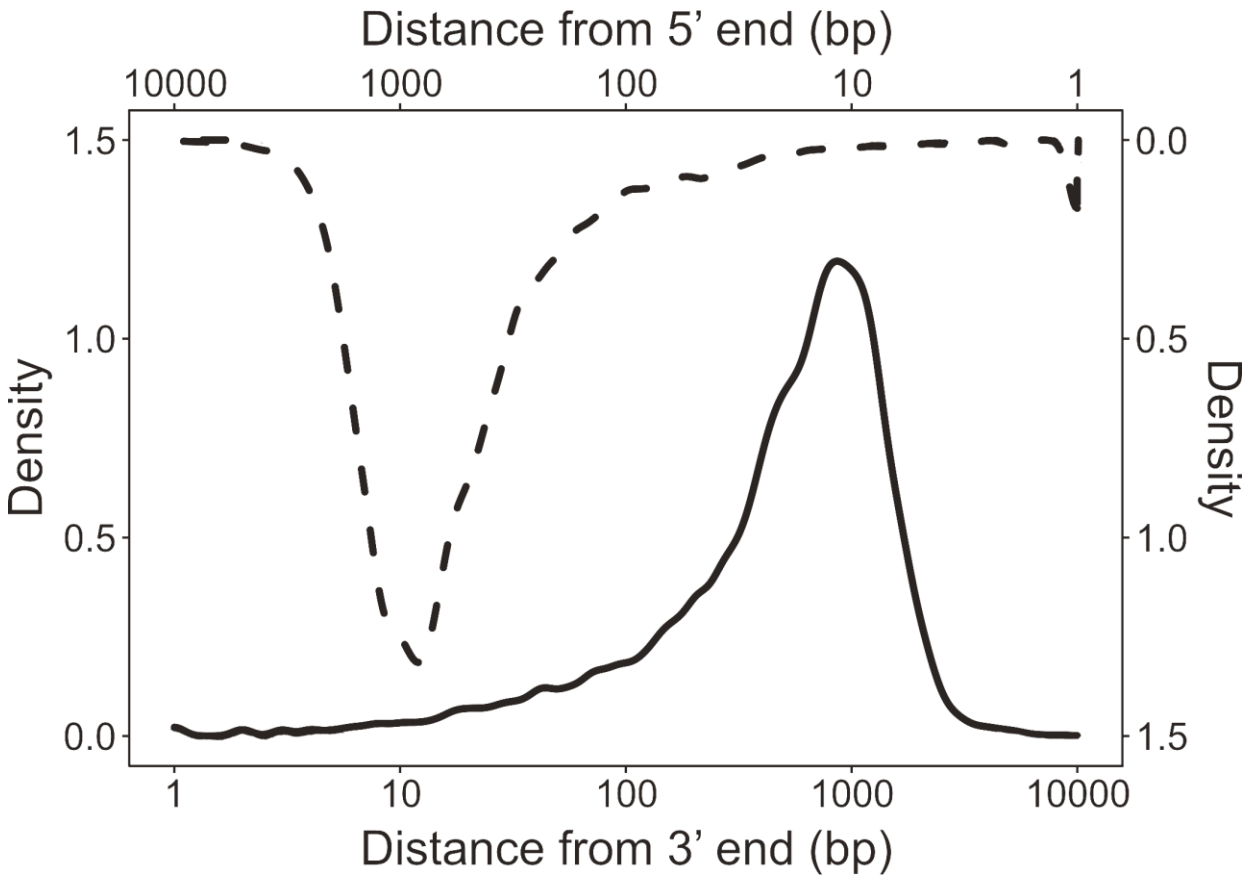


Figure 3.5 Length distribution of flanking regions, after Kollector assembly of *P. glauca* genes. In order to

define the flanking regions, we aligned the high-confidence transcript models of white spruce to our Kollector assemblies and quantified the length of the sequence upstream (dashed line, the upper x-axis and right-side y-axis) and the downstream (solid line, the lower x-axis and left y-axis) of the input transcript alignment.

3.5.1.4 Targeted cross-species gene assembly

Non-model organisms might not have extensive and well-annotated transcriptomes available to researchers. In such cases, Kollector can use transcript sequences from one species to reconstruct the genic regions of a related species.

3.5.1.5 Cross-species assembly using Kollector

We tested Kollector using *H. sapiens* transcript sequences as bait to assemble orthologous (>70% sequence identity) *Mus musculus* genes (Table 2.1, #4). Despite being separated by approximately 90 million years of evolution (http://timetree.org), Kollector was able to assemble 3,295 of 4,025 target genes in a single iteration (r = 0.90, assembly k = 96), corresponding to an 81.9% success rate, as assessed using the orthologous *M. musculus* transcripts. This single iteration took 4 hours using 18.1 GB of RAM.

3.5.1.6 Comparisons to aTRAM

We assessed Kollector's performance against aTRAM [85], which to our knowledge, is the only tool designed to assemble entire genes (including introns) guided by an input protein or DNA gene sequence with or without introns. In their study, Allen and co-workers (2015) used a dataset of 1,534 conserved proteins from human head lice (*Pediculus humanus*) [107] to assemble

orthologous genes in the chimpanzee lice (*Pediculus schaeffi*) genome. We ran Kollector using the same whole-genome shotgun reads and corresponding cDNA sequences for each orthologous gene, and compared the results using two metrics: the proportion of the target gene that aligned to the assembled scaffold, and a number of assembled genes that passed a reciprocal best BLASTn hit test with each target gene.

In our tests, Kollector slightly outperformed aTRAM on both metrics (on average 99.3% to aTRAM's 98.6%, and 1,552 genes passing the reciprocal BLASTn test compared with 1,530 in aTRAM). Kollector achieved this task in less than one-tenth of aTRAM's run time (Table 3.4). The markedly greater speed of Kollector is mainly due to its use of alignment-free classification with *k*-mers and Bloom filters, allowing it to process thousands of transcripts in a single run. Whereas aTRAM relies on iterative alignments to individual targets to recruit enough reads for their assembly. For applications on this relatively small scale, the progressive read filtering algorithm used by Kollector eliminates the need for iterations.

For these experiments we parameterized Kollector with $r = 0.9$, s = 0.5, $k = 48$, and $n =$ 10,000,000, values similar to our other experiments. Our success rate suggests that this method is robust in capturing relatively divergent sequences. We think this is due to our progressive filtering algorithm, which recruits reads from more conserved regions first and then uses them to extend the sequence in more divergent regions. Furthermore, the sensitivity of reconstruction of more divergent sequences may be increased by using a low *r* parameter (the specificity required for read tagging), which may be preferable if the evolutionary distance between two species is considerable.

Method	Proportion	Reciprocal Best-BLASTn	Time (h)
Kollector	99.3	1,532	2
aTRAM	98.6	1,530	25

Table 3.4 Comparison with aTRAM

3.5.2 Whole-genome targeted assembly

A prominent application of *de novo* assembly is the detection of specific, yet novel, sequences, where a mapping approach can introduce bias [108]. However, the large sample size of many studies, such as those of cancer genomics consortia (e.g. ICGC, TCGA), can put a strain on computational resources when *de novo* whole-genome assembly is required. Therefore, a fast and reliable targeted assembler, like Kollector, might be an attractive option for most researchers, especially when considering its ability to extend incomplete input sequences.

In order to demonstrate the extent of its utility, we have used Kollector for the targeted assembly of Human Papilloma Virus (HPV) in a cancer sample. The Cancer Genome Atlas consortium has profiled 279 head and neck squamous cell carcinomas (HSNCs) and detected many HPV positive samples, which were experimentally confirmed with immunohistochemistry and *in situ* hybridization [86]. We ran Kollector on the genomic data from one of the confirmed samples [TCGA-BA-4077] with HPV type -33. Kollector does not need the bait to match the exact target sequence, as demonstrated in Section 3.2, so we used HPV type-16 reference genome as bait, and were able to re-assemble the complete HPV type-33 genome sequence in a single iteration

with less than 15 minutes runtime and using 1 Gb of memory. We also used genomic reads from the matched normal sample as the negative control, and, as expected, Kollector did not yield any assembled HPV sequences. Because Kollector uses only sequences contributed by the reads, the assembled strain remains unbiased relative to our bait sequence.

3.6 Discussion and onclusions

We have described Kollector, a targeted *de novo* genome assembly application that takes transcript sequences and whole-genome sequencing reads as input and assembles the corresponding gene loci. Input transcripts are used to seed progressive Bloom filters, a greedy database that can efficiently expand into intronic regions using sequence overlaps. Due to our alignment-free approach, we demonstrate that Kollector can scale well up to large genomes, such as those of human and spruce.

When assembling genes from transcripts, we show that Kollector successfully assembles more gene loci than iterative read recruitment methods aTRAM and GRABb in less time and assembles more genes than both non-iterative read recruitment methods Mapsembler2 and TASR. However, we note in the latter case that this was expected since these tools were not designed for targeted gene loci assembly using RNA transcripts and are thus unable to fill in large intronic gaps. Kollector also successfully assembles more gene loci than aTRAM when assembling the genic space of a related species.

After our evaluations, we showcased three additional use cases for Kollector. The first one showed that Kollector was able to effectively assemble gene sequences in *P. glauca* (white

spruce), despite its large 20-Gbp genome. These gene assemblies typically included 1 kb of upstream and downstream sequence with respect to the input transcript, illustrating the utility of the approach to be able to examine promoters and other regulatory elements around genes for downstream applications. This demonstrates that for gene-centric investigations, Kollector can be a robust substitute for *de novo* whole-genome assembly, which remains computationally challenging at large scales.

The second use case concerned comparative genomics, where Kollector assembled *M. musculus* genes using orthologous *H. sapiens* transcripts as input. This comparative genomics approach is particularly valuable to researchers working on non-model organisms, which might not have extensive and well-annotated transcript sequences available.

Our final demonstration involved the whole-genome targeted assembly of HPV in a head and neck cancer sample. Kollector solves this problem by *de novo* assembling reads of interest, without the risk of introducing artifacts, as typically is the case when aligning reads to a reference genome. Because Kollector can fill in missing sequences by recruiting reads with a progressive Bloom filter, it only requires a limited amount of sequence homology within the bait sequence to fully reassemble a viral sequence. We note that the extent of divergence of the seed sequence that Kollector can use was not fully explored, and thus may be interesting to investigate in future studies.

In conclusion, we expect Kollector to be a valuable addition to the current suite of targeted assembly tools. Not only does it scale to large datasets, but it can also be used to reconstruct

orthologous sequences in non-model organisms and would find utility in the reconstruction of large regions *de novo,* using only transcript sequences.

Chapter 4: Mismatch-tolerant, alignment-free sequence classification using multiple spaced seeds and multi-index Bloom filters

4.1 Publication note

Most of the content of this Chapter has been submitted to the Proceedings of the National Academy of Science, and an early draft [109] is available as a preprint in BioRxiv. In addition, preliminary work for this project was presented as a talk at HitSeq 2016. Versions of this tool are featured as part of the Targeted Assembly Pipeline (TAP) [20] published in BMC Medical Genetics in 2018. The work described in this chapter is the main work of the author, under the supervision of Inanc Birol. The author designed and implemented the tool described performed the experiments and wrote the manuscript. The co-authors on this work helped test the software, provided valuable feedback during the development of the tool and helped in editing the manuscript.

4.2 Author summary

Alignment-free classification tools have enabled high-throughput processing of sequencing data in many bioinformatics analysis pipelines due to their advantages in computational efficiency. Originally *k*-mer based, such tools lack sensitivity in the face of sequencing errors and polymorphism, so some tools have been augmented with spaced seeds, which are capable of tolerating mismatches. However, the use of spaced seeds has not seen much practical use in classification because they are often accompanied by increased computational and memory costs compared to their *k*-mer based counterparts. These limitations have also caused the design and length of spaced seeds used in these tools to be limited as storing spaced seeds can be costly. To

address this, we have designed a novel probabilistic data structure called a multi-index Bloom Filter (miBF), which can store and query multiple spaced seed sequences to thousands of reference genomes, with low memory costs that remain static regardless of seed length. We formalize how to minimize false-positive rates, providing a framework applicable to future probabilistic data structures using multiple spaced seeds on multiple classification targets. Available as a part of BioBloom Tools, we demonstrate the utility of our tool in two applications: read binning for targeted assembly and taxonomic read assignment. Our tool shows higher sensitivity and specificity for read-binning than sequence alignment-based methods, also executing in less time. For taxonomic classification, we show higher sensitivity than a conventional spaced seed-based approach in an order of magnitude less time, while using half the memory. This chapter addresses objectives 2 and 3 in section 1.4.

4.3 Significance

There is sustained growth in data throughput in modern DNA sequencing platforms, surpassing the capacity-growth in computing infrastructure. For many biological studies, faster alignment algorithms alleviated some of the burden brought forth by rapidly increasing data volumes, but not all applications in genomics projects require sequence alignments. Accordingly, recent alignment-free methods are finding application in a wide range of tasks from transcript expression analysis, to metagenome characterization, to read filtering for localized de novo sequence assembly. Although these alignment-free classification algorithms are faster than alignment, they are often limited in sensitivity and may have high memory requirements. We demonstrate that by using spaced-seeds and probabilistic data structures, the sensitivity of alignment-free classification methods can be enhanced without impacting memory usage.

In addition to its practical value, this chapter also introduces a theoretical framework for using multiple spaced seeds in a multi-index Bloom filter. The concepts would be of interest to a wide audience, including researchers in various fields of computer science and data structures in particular. Ideas and concepts towards minimizing false-positive rates in massive volumes of queries would be of particular interest to researchers studying probabilistic data structures, which power the Internet. In these aspects, contribution to the literature of miBF stands out: while computational biology and bioinformatics usually adapt concepts from the computer sciences for effective analysis of large datasets, this work may be generalized to benefit the wider computer science community.

4.4 Introduction

In computational biology, sequence classification, the process of binning of biological sequences into categories such as genome of origin, is a common task with many applications such as contamination screening [15], pathogen detection [63, 64, 91], metagenomics [63, 64, 110], and targeted assembly from shotgun sequence data [83, 95]. Though this problem is addressable via sequence alignment [14], the scale of modern datasets (both of the query and the reference sequences), has spurred the development of faster alignment-free hashed based similarity methods [110], as exact genomic coordinates are often unnecessary and lead to more computation than necessary. We have developed a novel probabilistic data structure based on the Bloom Filter (BF) [33], a data structure that implicitly stores data to reduce memory usage. Employing multiple spaced seeds to represent sequences, it can better handle sequence polymorphisms and errors compared to other hash-based sequence classification methods.

The most common hash-based, alignment-free indexing methods are *k*-mer based. These methods work by breaking a reference sequence into sub-sequences of length *k* base pairs (bp) and indexing them (often in a hash table). To query sequences, they are broken into *k*-mers and interrogated against the index for shared *k*-mers. If a significant number of *k*-mers are found, the read is then classified. The *k*-mers must be long enough such that they are unlikely to be the same between indexed targets, especially if there is substantial sequence similarity. However, *k*-mers cannot compensate for differences between references and queries that occur within *k* base pairs of each other. This limitation has motivated us to use spaced seeds [66] (also called gapped q-grams [111]).

Spaced seeds are a modification to the standard *k*-mer where some wildcard positions are added to allow for approximate sequence matching. They were originally proposed in PatternHunter in 2002 [66], and have been used since to improve the sensitivity and specificity of homology search algorithms [112-116]. Employing multiple spaced seeds together can increase the sensitivity of homology searches [117]. Spaced seeds have also been employed in metagenomics studies to improve the sensitivity of classification [67, 68].

Probabilistic data structures are a class of data structures that focus on representing data approximately, so query operations can sometimes produce a false positive. The use of probabilistic data structures in bioinformatics has expanded in recent years, owing to their speed (hash table-like) and low memory usage. Different mitigation strategies to reduce the false positive rate (FPR) of these data structures have been proposed, however, no matter the methods, the core strategies are the reduction of base probabilities and the use of joint probability from

independent events. For example, from the first probabilistic data structure, the BF [33], false positives are reduced by lowering the occupancy of the BF (decreasing the base probability of a false positive) or by increasing the number of hash functions used (exploiting the joint probability of multiple independent events). These principles have not changed; however, there may be aspects of the datatype being indexed that may be exploited but are overlooked. For instance, some methods that utilize probabilistic data structures for key-value associations in sequence analysis consider every inserted key as an independent event [42, 48], assigning a single FPR for each key query. However, in the biological sciences, unlike in some other applications of these data structures in other situations, each key is a decomposition of the same sequences (i.e. multiple groups of keys originate from the same sequence) and are thus not independent and can be exploited to reduce the FPR when querying [118].

The use of BFs for sequence-based classification was originally developed in the tool Fast and Accurate Classification of Sequences [65]. We later developed BioBloom Tools (BBT) [15], which used heuristics to optimize the runtime and reduce the effect of false positives, as described in Chapter 2. Though BBT proved effective when using a small number of references, determining the specific reference of a queried sequence requires multiple BFs, which can lead to an O(n) time complexity, where n is the number of references. Here, we have extended the functionality of BFs for the classification against multiple references (key-value association) in a novel data structure called a multi-index Bloom Filter (miBF). While it shares similarities with existing data structures, it has properties that allow it to synergize with spaced seeds.

There are similar data structures that can be considered for the sequence classification problem, such as Bloomier filters [41], sequence Bloom trees [43], quotient filters [44, 51], the quasi-dictionary [42, 46], the Pufferfish data structure [53], the Othello data structure [47, 48], and interleaved BFs [49] with different time and memory complexities (See Section 1.3.3.2 for their comparison). The miBF data structure belongs in the same family of these data structures, with similar theoretical performance and properties, but is specifically designed to work with multiple spaced seeds as opposed to *k*-mers. At suggested parameterization, miBF requires around 20 bits per key, assuming 16-bit values and that every single spaced seed in a set of multiple spaced seeds is considered its own key. Look-up time is constant and will require up to 2 cache misses per key lookup. The FPR for a single key lookup of miBF may differ depending on the value it classifies to, and thus a direct comparison to the FPR of other methods is difficult. To most effectively query miBFs, they must be used with multiple lookups over the same sequence, amortizing the FPR to a small value. Because of this, we consider the FPR to be a function of both the parameters of the miBF and length of the sequence being queried.

In this chapter, we describe the miBF data structure, showing how we integrate multiple spaced seeds in this data structure, explain how it is constructed, and how it is queried. We present a formulation for calculating the FPR of queries to the miBF for any sequences of a given length. We implemented a generic sequence classifier using the miBF, and present in two use cases, read-binning for targeted assembly and metagenomic classification compared against state-of-the-art tools BWA-MEM and CLARK/CLARK-S respectively, showcasing the increased sensitivity and runtime performance the miBF provides. The data structure and the tool used here are provided through our software repository at https://github.com/bcgsc/biobloom.

4.5 Results

4.5.1 Filtering reads for targeted assembly

Targeted assembly can improve the throughput and reduce the complexity of assembly for applications such as clinical diagnostics for structural variants or other mutations. A typical procedure when performing targeted assembly is the extraction of sequencing reads in the target loci, before using these reads in a *de novo* assembly pipeline. This can be done via alignment or sequence classification since exact genomic coordinates are not necessary for a *de novo* assembly. For this application, we compared the binning of reads with BWA-MEM [119] with our method on a set of simulated reads. We simulated Illumina reads with a depth of 100x (2,303,019 read pairs) using pIRS (v1.1.1) [120] from a gene set composed of 580 COSMIC (v77) genes [121], and an equal number of non-COSMIC genes randomly selected from RefSeq [122]. We indexed the set of 580 genes into a miBF using a set of four spaced seeds (Section C.1). Because BWA-MEM unfairly suffers poor specificity when only the 580 genes are indexed (Figure C.1), we indexed all the genes in the human genome (Figure 4.1) rather than only the indexed set of COSMIC genes to prevent off-target alignments. The seeds used in the read binning experiments were 80 bp in length and have Shannon entropies of 2.953 and 2.958 (using a word size of 3), with mirrored templates having the same entropy.

Compared to BWA-MEM, BBT obtained a higher overall sensitivity (99.996% vs 98.687%) and lower overall FPR (0.400% vs 0.421%). On a per gene basis, BBT outperforms BWA-MEM in terms of F1 score, except for RANBP2 (Figure 4.1). This is due to additional reads that seem to originate from a similar homologous gene, that is not part of the gene set indexed in the miBF. If

BWA-MEM is used without a full indexed set it also suffers the same false positives (Figure C.1). We note that in our evaluation we did not penalize BWA-MEM for multi-mapping to a gene that was not in the target set so that it was comparable to BBT (which indexed only 580 genes). We compared the runtime of each tool, finding that BBT runs at least 2x faster than BWA-MEM, and scales better on more threads (Figure C.2). Memory usage of the classification stage of BBT on this set of 580 genes was 20MB. These tests were performed on the same machine (Intel Xeon E7-8867 2.5GHz).

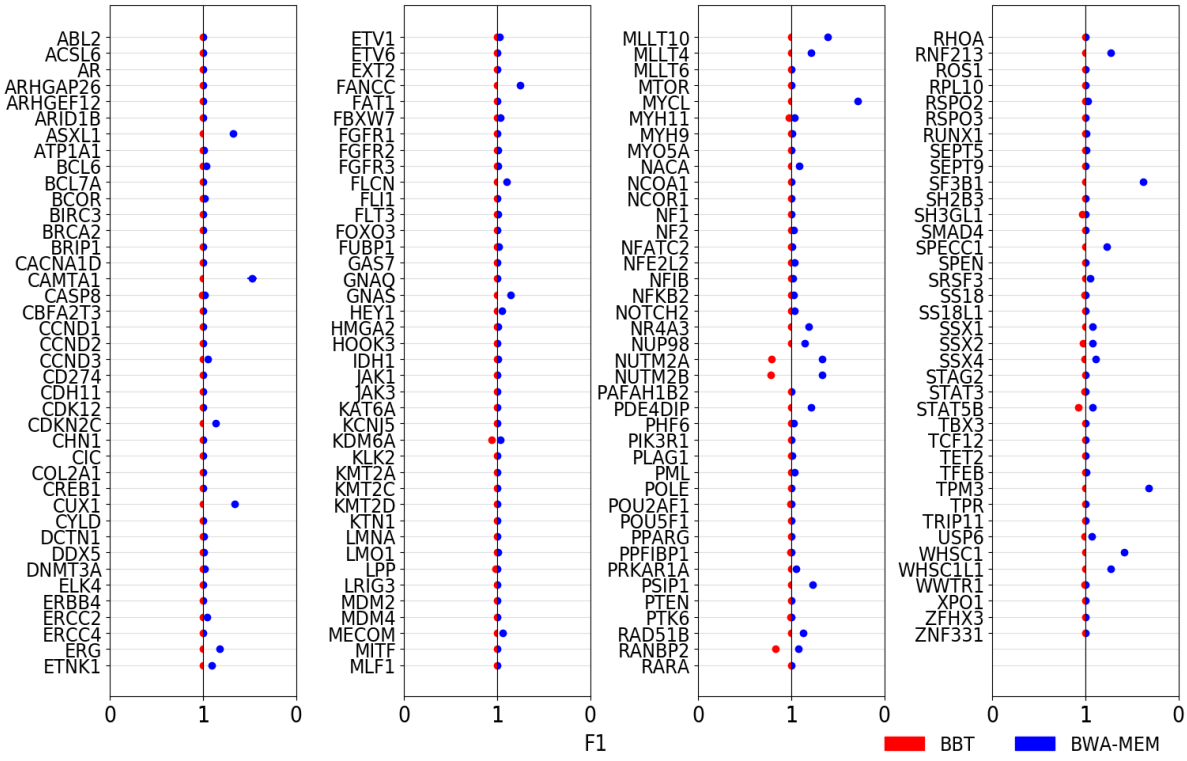


Figure 4.1 Per-gene comparison of classification performance by BBT vs BWA-MEM. F1 scores for both methods using a simulated dataset calculated for each gene and plotted on the same horizontal line. Only genes with F1 scores less than one is shown here (For full set *cf.* Figure C.3). The scale on the X-axis for BWA-MEM on the right is reversed for easy visual comparison such that higher scores for both methods localize to the middle.

We also compared the memory and time usage of indexing, but because the indexed set of COSMIC genes was small, it does not offer a fair comparison for the indexing stages of the two methods. Instead, to compare the scalability of indexing we indexed a 3.5G fasta file consisting of ~1000 bacterial sequences, using the same set of seeds and default parameters. BWA-MEM indexing algorithm does not offer parallelization and it took 1.5 hours to index in the entire file. Using a single thread BBT took 6.7 hours to index the same file, which dropped to only 0.7 hours using 16 threads. BWA-MEM only took 5GB of memory to index this file, whereas BBT used 33GB for the task.

4.5.2 Metagenomic classification

Although BBT is a generic classification tool, when given the proper reference sequences it is possible for it to be used as the workhorse for a metagenomics classification pipeline. To demonstrate this, we compared BBT with CLARK [64] and CLARK-S [68] (the spaced seed variant of CLARK). We compared CLARK-S because it is the only metagenomic classification tool that we know of that supports multiple spaced seeds to improve classification sensitivity. We also compare the method to CLARK, the predecessor to CLARK-S, because it is well characterized against other tools [123].

We first generated a standard database (bacterial + viral genomes, constructed May 2018) for both CLARK-S and CLARK. So that the comparisons were comparable, we used the same references sequences and taxon IDs that CLARK uses to construct the miBF (Section C.2). To index the genomes, CLARK took 24.0h, CLARK-S took 24.5h and BBT took 8.5 hours to

generate an index. We requested 64 threads for each tool, but CLARK generally did not use more than 1 CPU at a time, while BBT used around 13 CPUs on average, suggesting there is room to optimize parallelism when indexing for both tools, though more so for CLARK. Memory usage when indexing was 170GB for CLARK, 206GB for CLARK-S and 190GB for BBT. The seeds used in the metagenomic experiments were 42 bp in length and have Shannon entropies of 2.939 and 2.950 (using a word size of 3), again with mirrored templates having the same entropy.

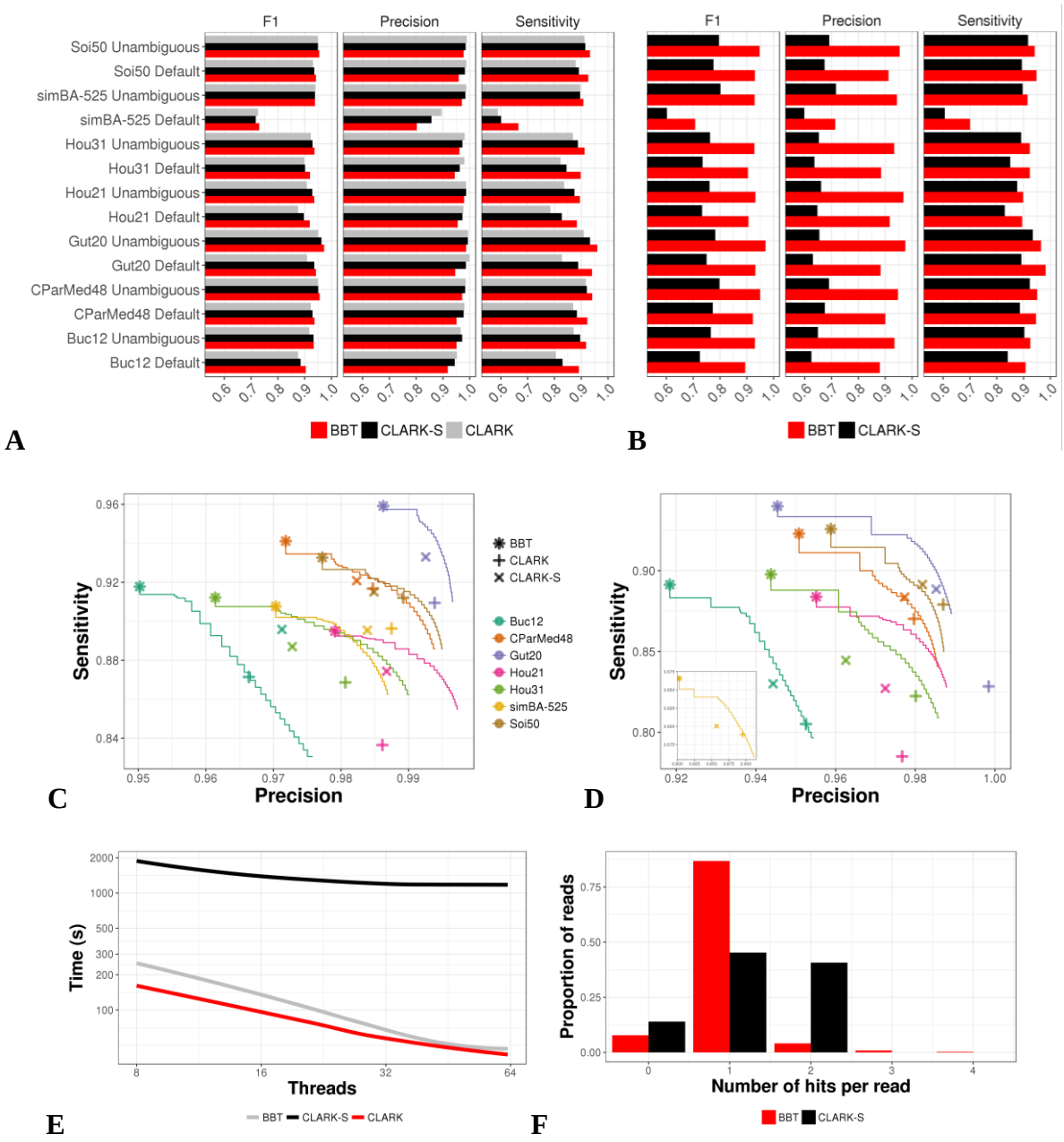


Figure 4.2 Comparison of CLARK, CLARK-S, BBT A: Precision and sensitivity considering only the best hit of a classification. B: Precision and sensitivity considering all multimaps. CLARK is omitted as it produces a single hit. C: ROC-like plot considering best hits on an unambiguous dataset. Lines indicate BBT runs parameterized to yield a higher precision (Section C.2). D: ROC-like plot considering best hits on default dataset. E: Runtime comparison of CLARK, CLARK-S, BBT at 8 to 62 threads. Both axes are in log scale, and under the situation of perfect scaling, the trend should follow a linear slope. F: Number of hits per record in all CLARK-S datasets for BBT and CLARK-S. Again, CLARK is not included because it does not multimap sequences.

There are differences between the miBF index and CLARK/CLARK-S databases beyond our implicit representation of seeds. First, unlike CLARK-S and CLARK, seed sequences shared between different taxa are not removed but distributed between taxa and if repetitive will be set as saturated (see Section 4.7). Also, though CLARK-S and BBT both use multiple spaced seeds, the miBF did not use the same set of seeds as CLARK-S because of our restrictions on seed designs (see Section 4.7). In addition, because our seed design does not affect memory usage of the miBF, we were also free to use longer seeds (Section C.1).

In our benchmarks, we used the simulated metagenomic datasets in the CLARK-S paper [68], however, because the NCBI databases have changed since the original CLARK-S publication, we had to omit read simulated from genomes that no longer have a corresponding species taxon in the database from the comparison. Nevertheless, since we omit the same reads in all runs and use the same reference sequences, our results still yield a fair comparison. We generated two sets of simulated reads as outlined in the CLARK-S paper; the difference between the "Default" and "Unambiguous" sets is the Unambiguous set does not have reads with all 32-mers shared between any two taxa IDs, but we note that, because the database has changed, this distinction may no longer hold completely true. The default datasets are more representative of real datasets, and the unambiguous datasets are more idealized for CLARK and CLARK-S.

The default output of CLARK only produces a single best match, but CLARK-S produces a secondary hit. Like CLARK-S, BBT produces secondary hits, with the difference being that BBT can produce more than one secondary hit. To this end, we compared the performance of each tool when it came to only the best hit in addition to comparing the results when multiple hits

were considered**.** For default BBT, only 5.4% have multi-hits, and of that, a majority (75.2%) only hit two targets (Figure 4.2B). CLARK-S provides a secondary hit 40.7% of the time thus penalizing the precision when considering multiple hits.

As expected, CLARK-S has higher sensitivity than CLARK in almost all cases, reproducing the results found in the original CLARK-S paper (Figure 4.2A, Table C.1, C.2). The trend shows that default BBT has the highest sensitivity allowing it to yield the highest F1 score in all but one case (Unambiguous simBA-525 dataset). In addition, we tested BBT with parameterizations to increase precision, by filtering ambiguous elements (Figure 4.2C, D and Section C.2). In most datasets, BBT will outperform CLARK and CLARK-S in sensitivity at the same precision, except for the unambiguous simBA-525 dataset.

Included with the CLARK-S datasets are three negative control datasets totalling in 3 million 100bp pair reads. Unexpectedly, some of the reads in the negative control datasets had mapped in both CLARK-S (six reads) and default BBT (one read). This contrasts with the original CLARK-S paper where no reads were mapped in any of the negative controls. In addition, in BBT, we expected no false positives because the minimum FPR parameter (-s) was specified to be less than 10^{-10} (default). Thus, we hypothesize this may be due to the difference in the databases, adding some new reference genomes that happen to have sequence similarity to those found in the negative control. Overall, we think this is not a cause for concern, as this represents a very small minority of the negative control.

We tested the runtime of each method at a differing number of threads (Figure 4.2E). We show that CLARK is the fastest tool, followed by BBT and finally CLARK-S. This trend follows when more threads are used, with the exception that BBT seems to scale better than CLARK, rivalling it at a higher number of threads. BBT and CLARK stay within an order of magnitude of each other, while CLARK-S takes an order of magnitude longer. CLARK used 87GB, BBT used 89GB, and CLARK-S used 175GB. Database loading speed was not considered in this comparison, but was dependent on I/O, and was quite comparable between the methods.

4.6 Discussion

We have described the miBF, a probabilistic sequence classification data structure that can classify to multiple value-types and synergizes well with multiple spaced seeds. Like BFs, the memory usage of miBF does not depend on the size of the seed indexed. With a BF, querying for the set of origin between multiple reference sets requires the construction and use of multiple BFs leading to O(n) time complexity when querying, where n is the numbers of reference sets/filters. In contrast, querying for the set of origin for an element in miBF requires only one instance of the data structure, and is performed in constant time. Packaged and implemented in BBT we have shown it to be a practical sequence classification tool, especially if sensitivity is a concern.

To optimize the available memory for a target FPR, BFs can use multiple hash functions to describe a given k-mer. We modified this concept in miBF: instead of using multiple hash values for the same k-mer, we hash multiple spaced seed templates across the input sequence. Naively, one may simply insert each spaced seed as its own element (using multiple hash functions for

each insertion); yet, since the seeds in the same "frame" of the sequence are dependent we can instead use a set of spaced seeds in the place of multiple hash functions. Also, by allowing for some spaced seeds in a frame to miss, we can tolerate mismatches when classifying sequences.

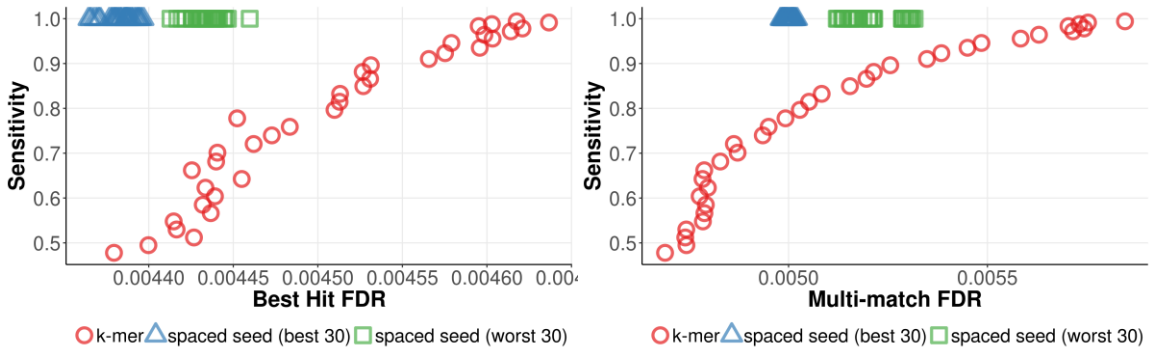


Figure 4.3 ROC-like plot investigating sets of multiple spaced seeds (60 designs) against *k*-mers (20 to 60) on classification on a miBF generated on 580 genes from the COSMIC database using reads 2x150 bp reads simulated with a 0.4% error rate on the same set of genes. Each set of spaced seeds has five seeds with the same weight of 20 and same length of 60. The *k*-mers used five hash functions to make the miBFs comparable.

Seed size has no impact on memory usage on this data structure, expanding the potential for specialized, highly sensitive and specific spaced seed designs. Optimal seed sensitivity computation is NP-hard [124], and although faster approximations exist [70], the problem remains difficult to optimize as our seeds can be any length. As a result, there is a vast opportunity for further research into multi spaced seed design with the miBF. Optimal seed design for sequence classification is a function of the sequencing error rate, homology detection tolerance, sequence length, and mutation/error types. Additionally, there are methods to hash multiple spaced seeds more efficiently [125], and some seed designs can be hashed more efficiently than others.

We investigated the impact of seed design on classification performance by randomly generating spaced seeds. In practice, using completely random seeds, it was difficult to create poor seeds with high overlap complexity [74]. To generate both well and poorly performing seeds, we used a generative Markov process [126] that purposely created seeds with high and low Shannon entropy [127]. We tried all possible combinations of these seeds into sets of 5 seeds and computed their overlap complexity. Our results show that seed design does matter, but even poorly generated spaced seeds tend to perform better compared to *k*-mers (Figure 4.3), suggesting one can expect gains on sensitivity relative to *k*-mers without extensive work on multiple spaced seed design. Thus, the seeds used throughout the paper were randomly generated with a script provided as part of BBT after picking a weight and seed length. Users of our tool can specify custom designs though there are a few restrictions as to which seed design can be used (see Section 4.7). As seed design improves, we expect the performance of our tool to improve.

To perform key-value lookups in the miBF, we store an ID for each hashed position. However, due to shared sequence or hash collisions, a loss of key-value associations can occur. If enough collisions occur, we can denote elements as saturated to let us know if key-value association information within a frame is lost, without destroying key-value pairs that are important for the classification of other sequences. In practice, the rate of saturation can be mitigated to 1% (depending on the number and relative repetitiveness of references) by using only 4 spaced seeds and a 50% BF occupancy (Figure C.5). Also, though largely unbiased, the representative counts

can vary between IDs, but the variance can be minimized by using more spaced seeds (Figure C.6).

When querying, we determine if a sequence is a false positive by using the frame counts (Section 4.7.6). We have formulated a model for calculating the FPR given a sequence (rather than a single element query), which also leverages the frequency of which an ID is retrieved. Interestingly, this means miBFs with more IDs will result in lower FPRs. The largest gains of having more IDs will be seen if there is low sequence sharing between references. Our formulation scales to the length of the input sequence and is thus more robust and reliable than hard thresholds based on number or proportion of hits. Finally, a lower FPR will benefit our runtime performance, because it will allow our early classification termination heuristics to activate more frequently. Frame counts are also generally a reliable metric for ranking multi-matches but can be distorted if the sequence contains repetitive sequences. When disambiguating multi-matches, we found the non-saturated frame counts (Section C.3) to be the best metric as it considers frames that may be repetitive yet is still able to tolerate mismatches. Sources of error when classifying and ranking multi-matches come from repetitive sequences, homologous references, sequencing errors, polymorphism, and random hash collisions during miBF construction.

We showcased our classification tool in two use cases. The first use case was the recruitment of reads for targeted assembly. We show superior sensitivity and FPR to BWA-MEM in less time. In addition, the use of spaced seeds affords a certain amount of specificity that allows BBT to index only the gene of interest, rather than the entire gene set, which BWA-MEM needs to

prevent off-target classification. The overall specificity of the matches was higher than BWA-MEM, but we note that in one target gene BWA-MEM outperformed BBT because it had access to the entire gene set. This suggests that a spaced seed approach may be superior in terms of overall specificity when using incomplete or missing reference sequences. The practical speed and sensitivity of miBFs have prompted use of miBFs in the targeted transcriptome assembly pipeline called TAP [20] and might be a good fit for other pipelines performing read binning. The second use case was the classification of metagenomic sequences to the species level. Under default parameterization, we showed higher sensitivity than both CLARK-S and CLARK. To illustrate that the gains to sensitivity were not necessarily at the cost of precision, we also ran BBT with parameters to allow for more precise classifications (by filtering ambiguous matches from the output). Under these parameters, we showed generally higher sensitivity at the same precision to CLARK and CLARK-S, though the power of this method lies in the improved overall sensitivity (at default parameters). Our maximum sensitivity gains were likely due to the use of a slightly sparser set of seeds with more seeds (four vs three), and because we do not filter out seed sequences that are shared between species. Despite the slight increase in sparseness, the specificity of the classification was maintained by using longer seeds (42 bp, longer than what CLARK-S can currently use).

Both CLARK-S and BBT can provide multi-matches. Although BBT may get the best hit incorrect in some cases, it is very likely that one of the secondary hits is the correct classification. When considering these multi-matches, BBT still outperformed CLARK-S with regards to sensitivity, and notably also had a much higher relative precision. However, this comparison to CLARK-S may be unfair since CLARK-S produces alternative hits liberally

(penalizing precision), whereas BBT only produces multi-hits when there is high ambiguity. In most cases, CLARK-S will often get the best hit correct, whereas BBT will have fewer reads with multi-hits, though these classifications will be much more likely to be true multi-maps. The runtime of BBT remained around two times slower than CLARK and generally scales better than CLARK when more threads are used. When parameterized with more specific settings BBT will run slower (depending on the parameters), so BBT better suits applications where sensitivity is important. Due to the implicit representation of the spaced seeds, BBT used half the memory CLARK-S and a similar amount of memory to CLARK despite using more spaced seeds. The runtime of CLARK-S was more than an order magnitude slower than CLARK and BBT, suggesting that computation using multiple spaced seeds can be quite expensive if not carefully optimized.

In conclusion, despite the complexity of using a tool based on a probabilistic data structure and spaced seeds, we expect this data structure and tool to be a valuable addition to existing classification methods due to its impressive computational performance as well as gains to classification sensitivity at scalable memory usage. The ideal application of our tools is in situations where high sensitivity is desired, and perhaps unknown or unindexed sequences may be present in the dataset. In addition, we hope that our work will invigorate research into spaced seed design because the length and weight of the seeds no longer have an impact on memory usage. Finally, our novel FPR formulations featured here may be widely applicable to any probabilistic data structures when classifying sequencing data and should help in keeping classification robust in the face of different length read sets and a lowered reliance on hard thresholds common to these methods.

4.7 Methods

4.7.1 Multiple spaced seeds

We hash multiple spaced seeds across a sequence broken into frames the size of the spaced seeds, similar to breaking a sequence into *k*-mers. Some spaced seeds in a single frame can miss, thus tolerating mismatches when classifying. By default, we will accept all but one seed in a frame to miss, but this can be changed (-*a*) to help decrease the FPR, at the cost of sensitivity. There is no restriction on length or weight (required match positions) for spaced seeds, however, the seeds must either share mirrored match positions to another seed or be palindromic, to enable us to store only one complement of the sequence. This allows us to save memory by storing each seed only once and not both forward and reverse-complements, analogous to storing canonical *k*-mers (comparing the forward and reverse-complement of a *k*-mer and consistently using one). The seeds used in our experiments were randomly generated subject to the following three conditions. We required that (1) they have 50% occupancy, (2) in a set of four seeds, there are exactly two seeds with a wildcard in each position, and (3) two of the seeds are mirrored to templates of the other two seeds. This design ensures that if we have a single mismatch in a query sequence at least two of the seeds report hits.

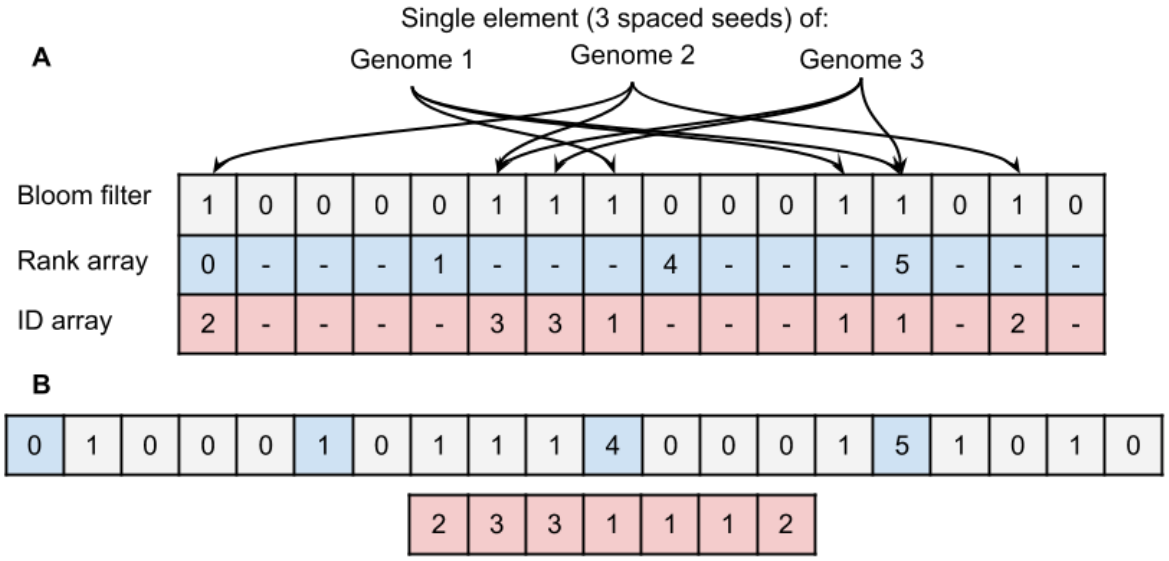
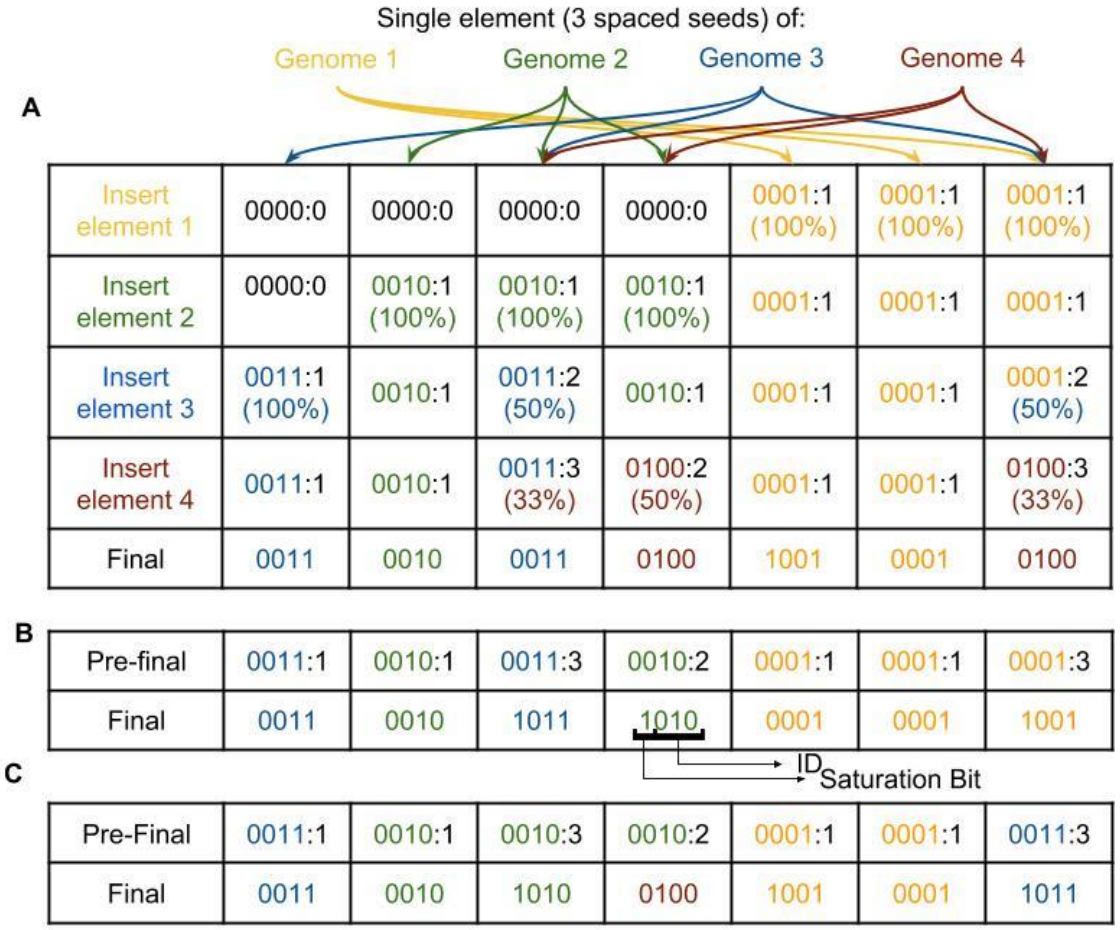


Figure 4.4 A: A visualization of the multi-index BF data structure. Three tables are used to represent the miBF and how they interact. B: The true form of the data with an interleaved form of the bit vector. The interval for the rank array is much larger than shown here (512 vs 4 bits), reducing its overhead to 64/512 = 0.125 bits per position.

4.7.2 Multi-index BF structure

The miBF can be thought of as three separate arrays. The first is a BF bit array, storing the presence or absence of an element in stored the miBF (with possible false positives) (Figure 4.4A). The next one stores the rank information of the bit array at specific intervals; this allows for constant time rank information access [57] to positions on the bit array. The third ID array stores integer identifiers for each element in the bit array. These integer IDs can be used to represent any arbitrary classification category such as strain or genome information. The rank array in conjunction with the BF is used to determine the rank of the value to index the integer identifier in the ID array. To improve cache performance the BF and rank arrays are interleaved into a single data vector [128] (Figure 4.4B).



Insert element 1	0000:0	0000:0	0000:0	0000:0	0001:1 (100%)	0001:1 (100%)	0001:1 (100%)
Insert element 2	0000:0	0010:1 (100%)	0010:1 (100%)	0010:1 (100%)	0001:1	0001:1	0001:1
Insert element 3	0011:1 (100%)	0010:1	0011:2 (50%)	0010:1	0001:1	0001:1	0001:2 (50%)
Insert element 4	0011:1	0010:1	0011:3 (33%)	0100:2 (50%)	0001:1	0001:1	0100:3 (33%)
Final	0011	0010	0011	0100	1001	0001	0100

Pre-final	0011:1	0010:1	0011:3	0010:2	0001:1	0001:1	0001:3
Final	0011	0010	1011	1010	0001	0001	1001

Pre-Final	0011:1	0010:1	0010:3	0010:2	0001:1	0001:1	0011:3
Final	0011	0010	1010	0100	1001	0001	1011

Figure 4.5 An illustration of a miBF data vector construction using a 3 bit ID. Values after the colons are the number of times an ID was considered to be inserted into that bucket, needed for reservoir sampling. The percentage in the parenthesis represents the chance of insertion of a different element into that position. A: Insertions into miBF of different keys randomized with reservoir sampling. Before the final pass, each element has at least one representative spaced seed inserted so no changes are needed. B: An example set of insertions causing saturation of some of the key-value association in the miBF. No replacement locations for locus 4 can be found, so the values are saturated. C: An example where locus 4 is not able to be represented before the final pass. Rather than saturation, a value of a duplicate of locus 2 is replaced by locus 4, the value with a smaller count is replaced first.

4.7.3 Multi-index BF construction

Construction of the miBF consists of multiple passes through the sequence set being indexed. To minimize memory usage overhead, all data is streamed, but multiple threads can be used on different parts of the data (so long as they are part of different IDs) to improve runtime. Three passes through the data are needed to construct a miBF. The first pass populates the BF, the second pass populates the ID array, and a final pass recovers colliding IDs and set saturation. Due to shared sequence or hash collisions, inserted values into the ID array may collide causing a loss of key-value association information. We compensate by flagging any loss of data and bias towards any ID across sets of sequences. To minimize bias, IDs are inserted into the data vector using reservoir sampling [129]. To do so a temporary count vector is needed, the same size of the data vector, updated analogously to a counting BF (Figure 4.5A). In the final pass, we set a saturation bit (Figure 4.5B) denoting a loss of information when an ID has no representative seeds for a single element (an element is considered a set of spaced seeds to one sequence position). This saturation bit indicates if key-value information within a frame is lost, without destroying key-value pairs that are important for the classification of other sequences. Finally, if no representative seeds exist for that element, but a duplicate ID is found (Figure 4.5C) in the same frame, we can prevent saturation by inserting the value into one of the duplicate positions. Duplicate positions happen most often when sequences are repetitive. More spaced seeds will help decrease any saturation (Figure C.5), but this will be at the cost of increased memory usage. Our construction method shows very little sequence bias in practice, and higher seed counts will decrease the variance of each count (Figure C.6).

4.7.4 Sequence classification

We perform classification in two stages. First, we determine if the matches to an ID are enough to determine the read is not a false positive match determined by a preset minimum FPR ($1/10^{10}$ by default). Then we rank the significant candidate IDs and filter in additional candidates that match strongly enough that they can be considered a multi-match. A lookup of the data structure is performed by hashing a set of spaced seeds in a frame and confirming if the value(s) are present, and if present using the rank value to retrieve the ID from the ID array. In this scheme, one can end up with multiple IDs for the query, with a high likelihood of one of them being a false positive. To query reliably, we use adjacent frame matches to the same ID to reduce the effective FPR, because matches to adjacent frames are independent events if they occur due to hash collisions. This is similar to other methods is shown to reduce the effective FPR in BFs [130], with the difference that multiple IDs with different frequencies in the filter need to be considered.

The chance of a false positive depends on the length of the sequence being queried, the length of the seed used, the FPR of the BF, and the frequency of each ID in the ID array (Figure C.7). When classifying a sequence, the FPR for each frame is a series of *n* independent Bernoulli trials (number of frames) so we can model the overall chance of a false positive using a binomial distribution. Our miBF FPR formulation is based on the BF FPR [33] as follows:

$$f = b^h = \sum_{x=h}^{h} \binom{h}{x} b^x (1-b)^{h-x}$$

where *b* is the occupancy of the BF and *h* is the number of multiple spaced seeds (traditionally number of hash functions) used for a single frame in the sequence. As we are also allowing some

misses due to our use of multi-spaced seeds, the formulation becomes:

$$f = \sum_{x=h-a}^{h} \binom{h}{x} b^x (1-b)^{h-x}$$

where *a* is the number of allowed misses for the set of spaced seeds in a frame. For a BF the chance of falsely classifying a sequence is easily determined by computing the cumulative density function (CDF) of the number of matches *m* - 1 and inverting it:

$$P(n,m) = 1 - \sum_{x=0}^{m-1} \binom{n}{x} (f)^x (1-f)^{n-x}$$

Computing the FPR of miBFs incorporates the fact that you can use multiple indexes to help further reduce the FPR, though it should be noted that each index *i* is an additional test. Before we compute the overall probability over an entire sequence for given index *i*, we must first formulate the probability of falsely matching a frame of classification:

$$f_i = \sum_{x=h-a}^{h} \binom{h}{x} b^x (1-b)^{h-x} (1-(1-s_i)^x)$$

where s_i is the frequency of index *i* in the data array of the miBF, relative to the frequency of all indices. Thus, the overall probability for false classification for index *i* is:

$$P(n,i) = 1 - \sum_{x=0}^{m_i-1} \binom{n}{x} (f_i)^x (1-f_i)^{n-x}$$

Out of all our tests for each index *i* we can take the best candidate (lowest p-value) and perform a multiple testing correction. In our implementation, we simply perform the Bonferroni correction [131]:

$$P(n,i)' = n \times P(n,i)$$

We explored different corrections using simulated data (Figure C.4) and showed that these correction methods generally result in similar corrected values for smaller critical values. In

practice we show that they are overly conservative in practice, potentially lowering the statistical power of the method. We hypothesize that this is caused by the fact that our values originate from a binomial distribution (which has a uniform cumulative probability density function but is also discrete), whereas these correction methods are formulated for a true non-discrete continuous uniform random distribution.

We filter in only matches that pass a minimum FPR threshold during classification. When the sequence classified is of a fixed length, we compute a fixed significant match threshold for each index by applying our Bonferroni-corrected critical p-value with a quantile function [132]. The classification will terminate early to improve throughput. We require several unambiguous significant matches (-r, default is 3) to terminate early. This heuristic has no effect on the FPR, and only affects the accuracy of multi-matches.

4.7.5 Ranking multi-mapping hits

Classified sequences can be associated with more than one candidate ID. To rank these candidate hits, we use the following hierarchy of match counts:

1. Non-saturated frame counts: ID counts to frames that do not have any saturated IDs in the same frame, only counted once per frame
2. Non-saturated solid frame counts: ID counts to frames without saturation, and only if the frame contains all seeds needed for a match, only counted once per frame
3. Frame counts: ID counts to frames regardless of saturation status, only counted once per frame

4. Non-saturated frame counts: ID counts that are not saturated, only counted once per frame allowing for frames to have some saturated IDs
5. Total non-saturated counts: all ID counts that do not have any saturated IDs
6. Total counts: all ID counts regardless of saturation status

These metrics should, in an ideal situation, agree with each other (i.e. always be higher if in the better match) and generally do. However, we also provide an option (-b) to filter classification of these cases where the metrics do not agree, improving the specificity, at the cost of a minor loss of sensitivity. We consider matches to be too close to call if any of their counts are within a threshold number of standard deviations (square-root of the count) of each other (-m, default of 3). The remaining candidates are returned, ranked by the hierarchy above.

4.7.6 Implementation details

The miBF data structure is implemented as part of BioBloom Tools (BBT) (https://github.com/bcgsc/biobloom), implemented in C++ with components from the BTL BF utility (https://github.com/bcgsc/btl_bloomfilter). For space seed hashing we use a modified version of ntHash[133], a recursive rolling hash specialized for nucleotide sequences. Finally, we use components and algorithms from the C++ Boost libraries [134] and the Succinct Data Structures Library [128].

Chapter 5: Innovations and challenges in detecting long read overlaps: an evaluation of the state-of-the-art

5.1 Publication note

Most of the work described in this chapter was previously published in Bioinformatics in 2017 [135]. The publication is a review/benchmarking paper featuring a list of published overlap algorithms, describing how they work and their bottlenecks, and evaluating their performance on two long read technologies. I wrote the manuscript, designed and implemented the experiments within the paper. Hamid Mohamadi helped refine the section on spaced seeds and created Figure 5.3 within the manuscript. Chen Yang aided in the simulation of ONT reads within the manuscript. Inanc Birol supervised the project. All co-authors helped edit the manuscript.

5.2 Author summary

Identifying overlaps between error-prone long reads, specifically those from Oxford Nanopore Technologies (ONT) and Pacific Biosciences (PB), is essential for certain downstream applications, including error correction and *de novo* assembly. Though akin to the read-to-reference alignment problem, read-to-read overlap detection is a distinct problem that can benefit from specialized algorithms that perform efficiently and robustly on high error rate long reads. Here, we review the current state-of-the-art read-to-read overlap tools for error-prone long reads, including BLASR, DALIGNER, MHAP, GraphMap, and Minimap. These specialized bioinformatics tools differ not just in their algorithmic designs and methodology, but also in their robustness of performance on a variety of datasets, time and memory efficiency, and scalability. We highlight the algorithmic features of these tools, as well as their potential issues and biases

when utilizing any particular method. To supplement our review of the algorithms, we benchmarked these tools, tracking their resource needs and computational performance, and assessed the specificity and precision of each. In the versions of the tools tested, we observed that Minimap is the most computationally efficient, specific and sensitive method on the ONT datasets tested; whereas GraphMap and DALIGNER are the most specific and sensitive methods on the tested PB datasets. The concepts surveyed may apply to future sequencing technologies, as scalability is becoming more relevant with increased sequencing throughput. This chapter addresses objective 1 in section 1.4.

5.3 Introduction

As today's lion share of DNA and RNA sequencing is carried out on Illumina sequencing instruments (San Diego, CA), most *de novo* assembly methods have been optimized for short read data with an error rate less than 1% [31, 136]. However, their associated short read length and GC bias sometimes bring significant challenges for downstream analyses [136, 137]. For instance, short read lengths make it difficult to assemble entire genomes due to repetitive elements [138]. The development of paired-end and mate-pair sequencing library protocols has helped mitigate this, but they do not completely resolve the issues inherent to short sequences [139]. Co-localization of short reads is a potential strategy to increase the contiguity of assemblies, using technologies such as Illumina TruSeq synthetic long reads [140] and 10X Genomics Chromium (Pleasanton, CA) [7]; however, tandem repeats in the same long single DNA fragment will continue to confound assembly methodologies. Also, synthetic and single-molecule long reads differ in the quality and quantity of their output; hence they require using different bioinformatics approaches.

Long read sequencing holds great promise and has proved useful in resolving long tandem repeats [141]. Still, the appreciable error rates associated with technologies offered by Oxford Nanopore Technologies (Oxford, UK; ONT) and Pacific Biosciences (Menlo Park, CA; PB) pose new challenges for the *de novo* assembly problem.

Read-to-read overlap detection is typically the first step of *de novo* Overlap-Layout-Consensus (OLC) assembly, the dominant approach for long read assembly [142, 143]. A read-to-read overlap is a sequence match between two reads and occurs when local regions on each read originate from the same locus within a larger sequence. OLC is an assembly process that uses these overlaps to generate an overlap graph, where each node is a read and each edge is an overlap connecting them. This graph is traversed to produce a layout of the reads, which is then used to construct a consensus sequence [76, 77]. Overlap detection has been identified as a major efficiency bottleneck when using OLC assembly methodology [144] for large genomes.
In addition to the importance of overlaps in OLC, the first *de novo* assembly methods for long reads employed error correction as their initial pipeline step [142, 143, 145], which often requires read-to-read overlaps. These error-corrected reads can then be overlapped again with higher confidence and ease due to the lowered error rate. Alternatively, one can forgo the error correction stages of assembly in favour of overlap between uncorrected raw reads [146]. The benefits of an uncorrected read-to-read overlap paradigm for assembly include the potential for lower computation cost [146], and repressing artifacts that may arise from read correction, such as collapsed homologous regions. On the other hand, for these methods, correctness of the initial

set of overlaps are even more critical, and because this assembly methodology may not perform any error correction at all [146] post-assembly polishing may be necessary.

At present, multiple tools are capable of overlapping error-prone long read data at varying levels of accuracy. These methods differ in their methodology but have some common aspects, such as the use of short exact subsequences (seeds) to discover candidate overlaps. Here we provide an overview of how each tool addresses the overlap detection problem, along with the conceptual motivations within their design. We also provide an evaluation of their performance on PB and ONT reads.

5.4 Background

5.4.1 Current challenges when using PB sequencing

PB sequencing uses a DNA polymerase anchored in a well small enough to act as a zero-mode waveguide (ZMW) [147]. The polymerase acts on a single DNA molecule incorporating fluorophores labelled nucleotides, which are excited by a laser. The resulting optical signal is recorded by a high-speed camera in real-time [5]. Base-calling errors on this platform occur at a rate of around 16% [31] and are dominated by insertions [136, 148], which are possibly caused by the dissociation of cognate nucleotides from the active site before the polymerase can incorporate the bases. Mismatches in the reads are mainly caused by spectral misassignments of the fluorophores used [5]. Deletions are likely caused by base incorporations that are faster than the rate of data recording [5]. The errors seem to be non-systematic and also show the lowest GC coverage bias as compared to other platforms [136].

In addition to Phred-like quality values (QV), the instrument software reports three error-specific QVs (insertion, deletion, mismatch) [149]. As with other next-generation sequencing technologies [150], the total QV score consists of Phred-like values, which does not necessarily match the expected Phred quality score [148, 151]. We note that, other than BLASR, the current state-of-the-art overlap algorithms, tested herein, do not take quality scores into consideration. The error rate of PB sequencing can be reduced through the use of circular consensus sequencing (CCS) [152]. In CCS, a hairpin adaptor is ligated to both sides of a linear DNA sequence. During sequencing, the polymerase can then pass multiple times over the same sequence (depending on the processivity of the polymerase). The multiple passes are called into consensus and collapsed, yielding higher-quality reads. The use of CCS reads simplifies error correction and prevents similar, but independent, genomic loci from correcting each other [153]. However, these reads are shorter and also result in lower overall throughput, so many PB datasets generated do not utilize this methodology. Because of this trend, the methods for overlap detection outlined in this paper have thus been designed for non-CCS reads.

5.4.2 Current challenges when using ONT sequencing

ONT sequencing works by measuring minute changes in ionic current across a membrane when a single DNA molecule is driven through a biological nanopore [6]. Currently, signal data is streamed to a cloud-based service called Metrichor that, at the time of performing this work, uses hidden Markov models (HMM) with states for every possible 6-mer to render base calls on the data. Metrichor also provides quality scores for each base call, however, like other next-generation sequencing technologies [150], the values do not follow the Phred scale [154].

In the current HMM base calling methodology, if one state is identical to its next state, no net change in the sequence can be detected. This means that homopolymer states longer than six cannot be captured as they would be collapsed into a single 6-mer. It has also been observed that there are some 6-mers, particularly homopolymers, underrepresented in the data [142, 155] when compared to the 6-mer content of the reference sequence, suggesting that there may be a systematic bias to transition in some states over others. In addition, there is some evidence suggesting GC biases within this type of data [154, 156]. We note that the base calling problem is under active development, with alternative base-calling algorithms such as Nanocall [157] and DeepNano [158], recently made publicly available.

One can mitigate error rates in ONT data by generating two-direction (2D) reads. Similar to CCS for the PB platform, 2D sequencing involves ligating a hairpin adaptor, and allowing the nanopore to process both the forward and reverse strand of a sequence [155]. Combining information from both strands was shown to decrease the error rate from 30-40% to 10-20% [15, 155], similar to the error rates of non-CCS PB sequencing. For the comparisons presented in this paper, we only consider 2D reads, as we expect investigators to prefer using higher quality ONT data. While subsequent to our work 2D protocol was deprecated because of a patent dispute by PB, the quality of the regular ONT reads improved enough, such that our analysis here still remains relevant.

5.5 Definitions and concepts

In the context of DNA sequencing, an *overlap* is a broad term referring to a sequence match between two reads due to local regions on each read that originate from the same locus within a

larger sequence (e.g., genome). A read-to-read overlap can be depicted at varying levels of detail that has implications on both the downstream processing and computational costs associated with overlap computation, as discussed below.

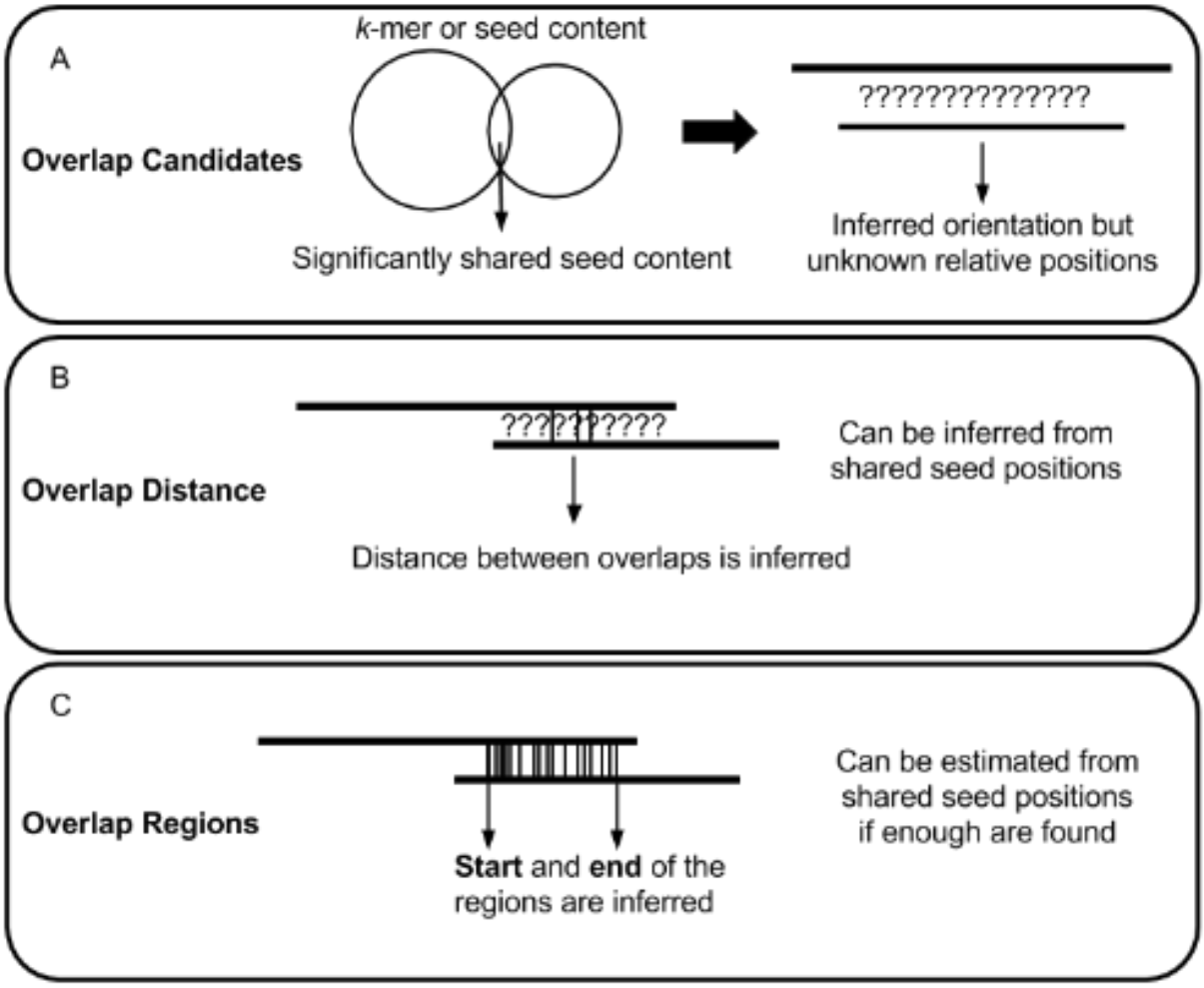


Figure 5.1 An overview of possible outcomes from an overlap detection algorithm. Each level has a computational cost associated with it, with the general trend being A<B<C. The common seeds-based comparison methods are not the only way to obtain these overlaps, but it is the most popular method used.

5.5.1 Definitions

The task of determining *overlap candidates* (Figure .51A) is usually the first step in an overlap algorithm, and it refers to a simple binary pairing of properly oriented reads. To find overlap candidates on error-prone long reads, most methods look for matches of short sequence seeds (*k*-mers) between the sequences.

Overlap distance (Figure 5.1B) refers to the relative positions between two overlapping reads. These distances provide directionality to the edges of the overlap graph. Theoretically, if the sequences are free of insertions or deletions (indels), then a correct overlap distance would be sufficient to produce a layout and build a consensus from the reads. However, even a single indel error in one of the reads will cause a shift of coordinates, which would complicate consensus calling. Also, one cannot distinguish between partial and complete overlaps just with the distance information alone. Overlap distance can be estimated without a full alignment, based on a small number of shared seeds.

Overlap regions (Figure 5.1C) refer to relative positions between overlapping reads, with the added information of start and end positions of the overlap over each read. If no errors are present, the sizes of the regions on both reads should be identical. In practice, due to high indel errors in long reads, this is rarely the case. Nevertheless, one can use this information to distinguish between partial and full overlaps. Similar to overlap distance, overlap regions can be estimated without a full alignment, but typically, more shared seeds are required for confident estimations.

Figure 5.2 Visualization of partial and full overlaps in dovetail or contained (containment) forms. The grey portion between the reads indicates the range of the overlap region, note that partial overlaps do not extend to the end of the reads.

Overlap regions between two reads may be *full* (complete) or *partial* and may *dovetail* each other or one may be *contained* in the other (Figure 5.2). Full overlaps are overlaps that cover at least one end of a read in an overlap pair, whereas partial overlaps cover any portion of either read without the ends (Figure 5.2). Sources contributing to observed partial overlaps include false positives due to near-repeats, chimeric sequences, or other artifacts [146]. Partial overlaps may also be a manifestation of read errors or haplotypic variations or polymorphisms, where mismatches between reads prevent the overhangs to be accounted for in the other reads. Disambiguating the source of the overhang in partial overlaps may be important to downstream applications, especially when using non-haploid, metagenomic, and transcriptomic datasets.

5.5.2 Alignments versus overlaps

There are many similarities between methods for local alignment and methods for overlap detection since their use of seeds to find regions of local similarity are common to both problems. Somewhat similar to the discovery of partial overlaps, it may be important to find local alignments, as they may help discover repeats, chimeras, undetected vector sequences and

other artifacts [144]. However, although a local aligner can serve as a read overlap tool [159, 160], overlaps are not the same as local alignments.

Unlike a local alignment tool, at a minimum, a read overlapper tool may simply indicate overlap candidates, and will typically only provide overlap regions, rather than full base-to-base alignment coordinates. In addition, local alignment algorithms require a reference and query sequence, typically indexing the reference in a way that query sequence can be streamed against it. An overlap algorithm does not require a distinction between query and reference, leading to novel indexing strategies that facilitate efficient lookup and comparison between reads without necessarily streaming reads. Finally, although, it is possible for an overlap detection algorithm to produce a full account of all the bases in overlapping reads, doing so would typically require costly algorithms like Smith-Waterman [161]. Indeed, though many tools presented in this review can produce full local alignments, some tools provide an option for computing overlap regions and local alignments separately (e.g. GraphMap [160]). Alternatively, other tools only provide overlap regions and do not provide any additional alignment refinement (such as MHAP [143] and Minimap [146]).

5.6 Long read overlap methodologies

Sequence overlap algorithms look for shared seeds between reads. Due to the higher base error of PB and ONT sequence reads, these seeds tend to be very short (Figures D.1 and D.2). The core differences between algorithms (Figure 5.3) relate to not only how shared seeds are found, but in the way, the seeds are used to determine an overlap candidate. After a method finds candidates, it will validate the overlaps and compute the estimated overlap regions usually by

comparing the locations of each shared seed, ensuring that they are collinear and have a consistent distance relative to other shared seeds. Each method produces a list of overlap candidates and provides an overlap region between reads. In some pipelines, the majority of the computation time is spent on realigning overlapping reads for error correction after candidates are found [162]. In others, precise alignments may not be needed [146]. Thus, the output of each overlap algorithm contains, at minimum, the overlap regions, and often with some auxiliary information for downstream applications (Table 5.1).

Software	Algorithm features	Associated assembly tools	Output	Availability
BLASR	FM-Index, anchor clusters	PBcR	SAM alignment, other proprietary formats (overlap regions)	https://github.com/PacificBiosciences/blasr
DALIGNER	Cache efficient *k*-mer sort (radix) and merge	DAZZLER, MARVEL, FALCON	Local Alignments, LAS format (alignment tracepoints)	https://github.com/thegenemyers/DALIGNER
MHAP	MinHash	PBcR, Canu	MHAP output format (overlap regions)	https://github.com/marbl/MHAP
GraphMap	Gapped q-gram (spaced seeds), colinear clustering	Ra	SAM alignment, MHAP output format (overlap regions)	https://github.com/isovic/GraphMap
Minimap	Minimizer colinear clustering	Miniasm	PAF (overlap regions)	https://github.com/lh3/Minimap

Table 5.1 Summary of overlap tools output formats, associated pipelines, and availability.

5.6.1 BLASR

BLASR was one of the first tools developed specifically to map PB data to a reference [159]. It utilizes methods developed for short read alignments but is adapted to long read data with high

indel rates and combines concepts from Sanger and next-generation sequencing alignments. BLASR uses an FM-index [163] to find short stretches of clustered alignment anchors (of length *k* or longer), generating a shortlist of candidate intervals/clusters to consider. A score is assigned to the clusters based on the frequency of alignment anchors. Top candidates are then processed into a full alignment.

Although BLASR was originally designed for read mapping, it has since been used to produce overlaps for *de novo* assembly of several bacterial genomes [145]. Still, to use the method for overlap detection one needs to carefully tune its parameters. For example, to achieve high sensitivity, BLASR needs prior knowledge of the read mapping frequency to parameterize nBest and nCandidates (default 10 for both) to a value higher than the coverage depth. The runtime of the tool is highly dependent on these two parameters [143], which may be due to the cost of computing a full alignment, the added computational cost per lookup to obtain more anchors or a combination of the two.

What slows down this method is the choice of data structure, and its search for all possible candidates (not only the best candidates) for each lookup performed. The theoretical time complexity of a lookup in an FM-index data structure is linear with respect to the number of bases queried [163], albeit not being very cache efficient [144]. Thus, if one maps each read to a unique location, this would only take linear time with respect to the number of bases in the dataset. However, since only short (and often non-unique, cf. Figures D.1, D.2) contiguous segments can be queried due to the high error rate, extra computation is required to consider all additional candidate anchor positions. Finally, BLASR computes full alignments rather than just

overlap regions, thus, possibly utilizing more computational resources than needed for downstream processes.

5.6.2 DALIGNER

DALIGNER was the first tool designed specifically for finding read-to-read overlaps using PB data [144]. This method focuses on optimizing the cache efficiency, in response to the relatively poor cache performance of the FM-index suffix array/tree data structure. It works by first splitting the reads into blocks, sorting the *k*-mers in each block, and then merging those blocks. The theoretical time complexity of DALIGNER when merging a block is quadratic in the number of occurrences of a given *k*-mer [144].

To optimize speed and mitigate the effect of merging, DALIGNER filters out or decreases the occurrences of some *k*-mers in the dataset. Using a method called DUST [164], DALIGNER (-mdust option) masks out low complexity regions (e.g. homopolymers) in the reads before the *k*-mers are extracted. Using a second method, it filters out *k*-mers in each block by multiplicity (-t option), increasing the speed of computation, decreasing memory usage, and mitigating the effects of repetitive sequences. However, these options also carry the risk of filtering out important *k*-mers needed for overlaps.

To use DALIGNER efficiently on larger datasets, splitting of the dataset into blocks is necessary. The comparisons required to perform all overlaps is quadratic in time relative to the number of blocks. DALIGNER provides a means to split input data based on the total number of base pairs and read lengths (using the DBsplit utility). DALIGNER optionally outputs full overlaps but will

first output local alignment *tracepoints* to aid in computing a full alignment in later steps, producing large auxiliary files.

5.6.3 MHAP

MHAP [143] is a tool that uses the MinHash algorithm [165] to detect overlaps based on the *k*-mer similarity between any two reads. MinHash computes the approximate similarity between two or more sets by hashing all the elements in the set with multiple hash functions and storing the elements with the smallest hashed values (minimizers) in a *sketch* list. Using the minimum hash value is a form of locality-sensitive hashing since it causes similar elements in a set to hash to the same value. In MHAP, overlap candidates are simply two *k*-mer sets that have a Jaccard index score above a predefined threshold. After the overlap candidates are found, overlap regions are computed using the median relative positions of the shared minimizers. These overlaps are validated by using the counts of the second set of shared minimizers that may be of smaller size *k* (for accuracy) within 30% (--max-shift) of each overlap region [143].

The time complexity of computing a single MinHash sketch is O(*khl*), where *l* is the number of *k*-mers in the read set for a sketch size *h*. Evaluating *n* reads for all resemblances traditionally takes $O((hn)^2)$ time [165], though, MHAP further reduces its time complexity by storing *h* min-mers in *h* hash tables to use for lookups to find similar reads [143]. Because the sketch size used for each read is the same, MHAP may unnecessarily use more memory, and lose sensitivity if reads vary widely in length [146].

Like DALIGNER, MHAP functions best when repetitive elements are not used as seeds. MHAP supports the input of a list of *k*-mers, ordered by multiplicity, obtained by using a 3rd party *k*-mer counting tool, such as Jellyfish [166].

MHAP's computational performance may be confounded by its implementation. While most high-performance bioinformatics tools utilize C/C++ for their performance benefits, MHAP is implemented in Java. Another method called Minlookup [167], written in C utilizes a similar algorithm to MHAP, and it is designed with ONT datasets in mind. The authors demonstrate improved performance associated with their implementation. However, Minlookup was not evaluated here as it was in early development, at the time of performing this work, and could not use multiple CPU threads.

5.6.4 GraphMap

GraphMap, like BLASR, was designed primarily as a read mapping tool [160], but for ONT data. It specifically addresses the overlap detection problem, notably producing full alignments. GraphMap also provides an option to generate overlap regions exclusively.

In GraphMap the "-owler" option activates a mode specifically designed for computing overlaps. Like its standard mapping algorithm, it first creates a hash table of seeds from the entire dataset. The seeds it uses are not *k*-mers, but rather gapped q-grams [168] – *k*-mers with wild card positions, also called spaced seeds [169]. It is not clear what gapped q-grams work optimally with ONT or PB data; more research is needed to determine the optimal seeds to cope with high error rates. The current implementation uses a hardcoded seed that is 12 bases long with an

indel/mismatch allowed in the middle (6 matching bases, 1 indel/mismatch base, followed by 6 matching bases). GraphMap then collects seed hits, using them for finding the longest common subsequence in k-length substrings [170]. The output from this step is then filtered to find collinear chains of seeds (private correspondence with Ivan Sović). The bounds of these chains are then returned, using the MHAP output format.

5.6.5 Minimap

Minimap [146] is an overlapper/mapping tool that combines concepts from many of its predecessors, such as DALIGNER (*k*-mer sorting for cache efficiency), MHAP (computing minimizers) and GraphMap (clustering collinear chains of matching seeds). Minimap subsamples the hashed *k*-mer space by computing minimizers and compiles the corresponding *k*-mers along with their location on their originating reads.

Like MHAP, the use of repetitive *k*-mers as the min-*k*-mer can degrade the performance of overlap detection. To minimize the effect of repetitive elements, Minimap uses an invertible hash function when choosing min-*k*-mers. This is similar to DALIGNER's use of DUST; it works by preventing certain hash values that correspond to low complexity sequences. Similar to DALIGNER, Minimap automatically splits datasets into batches (blocks) to reduce the memory usage on a large dataset.

Also similar to DALIGNER, Minimap was designed with cache efficiency in mind. It stores its lists of minimizers initially in an array, which is later sorted for the seed merging step. Though the computational cost incurred by sorting the list can negatively impact performance compared

with the constant cost of insertion in a hash table, its cache performance outperforms a conventional hash table. All hits between two reads are then collected using this sorted set and are clustered together into approximately collinear hits. The overlap regions for each pair of overlaps are then finally outputted in pairing mapping format (PAF) [146].

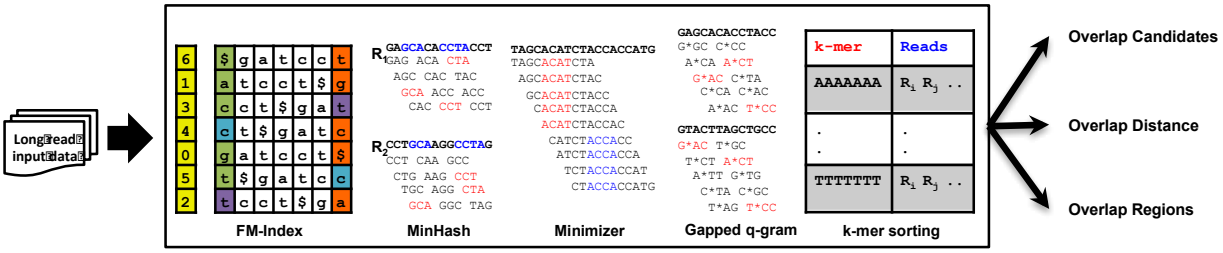


Figure 5.3 Visual overview of overlap detection algorithms. At the least, each method produces overlap regions. They may also generate auxiliary information, such as alignment tracepoints or full alignments. We show different seed identification approaches from leading overlap detection tools in the central box. BLASR utilizes the FM-Index data structure for seed identification. MHAP employs the MinHash sketch for seed selection. Minimap takes Minimizer sketch, a similar sketch approach used by MHAP. GraphMap uses gapped q-grams for finding the seeds. DALIGNER takes advantage of a cache-efficient k-mer sorting approach for rapid seed detection.

5.7 Benchmarking

We profiled and compared results from BLASR, DALIGNER, MHAP, GraphMap, and Minimap, using publicly available long read datasets with the newest chemistries available at the time of the study (Table D.1). We simulated *E. coli* datasets for the PB and ONT platforms using PBSim [171] and NanoSim [172], respectively (Section D.3), and simulated ONT *C. elegans* reads using NanoSim (Table D.1). We used experimental PB *E. coli* (P6-C4) and *C. elegans* whole-genome shotgun sequencing datasets and experimental ONT *E. coli* (SQK-MAP-006)

dataset. In-depth evaluations of specificity and sensitivity required a comprehensive parameter sweep, thus only the *E. coli* datasets were investigated in this section, as the larger *C. elegans* dataset proved to be intractable when used with some of the tools.

5.7.1 Sensitivity and FDR

We profiled the sensitivity and false discovery rate (FDR = 1 - precision) on the experimental PB P6-C4 *E. coli* and the ONT SQK-MAP-006 *E. coli* datasets. We also evaluated the tools on simulated data generated based on these datasets. Our ground truth for the real dataset was determined via BWA mem alignments to a reference, using -x pacbio and ont2d options, respectively [58].

We note that the ground truth may have missing or false alignments. In addition, Minimap was originally validated using BWA mem alignments, which may bias the performance measurements of this tool. In the same vein, it is possible for BLASR and BWA to share similar biases since they both use suffix arrays and have a similar algorithmic approach. However, these alignments can still serve as a good estimate for ground truth comparisons, since mismatch rate to a reference is much lower than the observed mismatch between overlapping reads. In the latter case, reads that are, say, 80% accurate will have a mutual agreement of 64% on average. In addition, due to our reference-based approach, our metrics are resilient against false overlaps caused by repetitive elements. Further, all tools are compared against the same alignments; hence we expect our analysis to preserve the relative performance of tools. Finally, there is no ambiguity for ground truth in the simulated datasets, as each simulation tool reports exactly

where in the genome the reads were derived from, allowing us to calculate the exact precision and sensitivity of each method.

To produce a fair comparison, we used a variety of parameters for each tool (Section D.1). These parameters were chosen based on tool documentation, personal correspondence with the authors, as well as our current understanding of their algorithms. We ran MHAP with a list of *k*-mer counts derived from Jellyfish [166] for each value of *k* tested to help filter repetitive *k*-mers. Unfortunately, GraphMap could not be parameterized when running in the "owler" mode and had only one set of running parameters. In this regard, our results for GraphMap may not be impartial; the more the parameters we evaluated for an algorithm, the better the chance of this algorithm to outperform the others in the Pareto-optimal results.

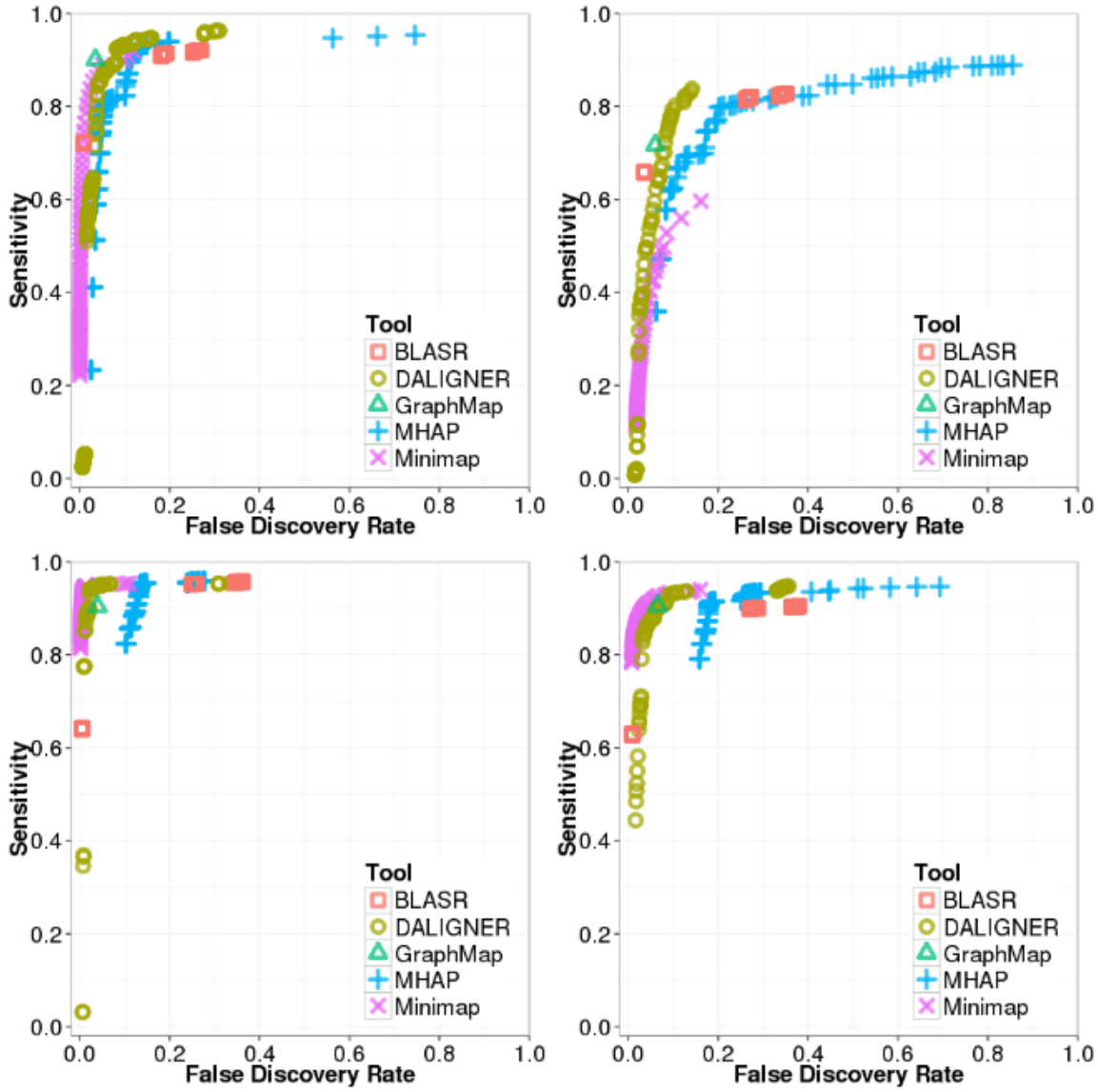


Figure 5.4 ROC-like plot using BLASR, DALIGNER, GraphMap, MHAP, GraphMap, and MHAP. Top left: PB P6-C4 *E. coli* simulated with PBsim. Top right: PB P6-C4 *E. coli* dataset. Bottom left: ONT SQK-MAP-006 *E. coli* simulated with Nanosim. Bottom right: ONT SQK-MAP-006 *E. coli* dataset.

We counted an overlap as correct when the overlapping pair was present in our ground truth with the correct strand orientation. With these metrics, we did not take into account reported lengths of overlap (Figure D.4) but note that this information may be important (e.g. to improve the

performance of realignment). For each tool we plotted these results on receiver operating characteristic (ROC)-like plots (featuring FDR rather than the traditional false positive rate) (Note D.1, Figure D.3). For ease of these comparisons, we computed the skyline, or Pareto-optimal results (the points with the highest sensitivity for a given FDR) (Figure 5.4.).
We can see that although many tools have similar sensitivity and FDR depending on the parameterization, the overall trends reveal differences in sensitivity and FDR on each specific datatype. For instance, MHAP can achieve high sensitivity on all datasets but lacks precision compared to most other methods on the ONT datasets. The only other tool that may have less precision on the ONT datasets is BLASR. DALIGNER proves to have a high sensitivity and precision, but it is not always the winner, especially on the ONT dataset. Minimap has high sensitivity and precision on the ONT datasets but does not maintain such performance on the PB dataset. Finally, the results for GraphMap were competitive despite using a single parameterization.

These plots reveal that the selection of operating parameters very much depends on the balance of project-specific importance attributed to sensitivity and precision, as expected. For instance, the importance of sensitivity is clear as it provides critical starting material for downstream processing. On the other hand, low sensitivity can be tolerated if the downstream method employs multiple iterations of error correction because as errors are resolved within each iteration, the sensitivity is expected to increase. However, these downstream operations, of course, may come with a high computing cost.

The F1 score (also F-score or F-measure) represents a common way to combine the two scores we used. It is the harmonic mean between sensitivity and precision. To better compare these methods, we computed F1 scores for each using a range of parameters and considered the highest value for each method to be representative of its overall performance. We calculated confidence intervals for the F1 scores using three standard deviations around the observed values, which revealed that reported F1 values were statistically significantly different from each other.

Tool	Simulated PB *E. coli*			Simulated ONT *E. coli*			PB P6-C4 *E. coli*			ONT SQK-MAP-006 *E. coli*		
	Sens. (%)	Prec. (%)	F1 (%)	Sens. (%)	Prec. (%)	F1 (%)	Sens. (%)	Prec. (%)	F1 (%)	Sens. (%)	Prec. (%)	F1 (%)
BLASR	91.0	81.9	86.2	**95.2**	75.1	84.0	66.0	**96.5**	78.3	89.9	73.0	80.6
DALIGNER	**92.4**	91.9	92.1	94.9	97.6	95.9	**83.8**	85.8	**84.8**	**92.9**	91.0	91.9
MHAP	91.5	88.0	89.8	95.1	86.5	90.6	79.8	79.8	79.8	91.2	82.0	86.3
GraphMap	90.1	**96.5**	**93.1**	90.4	96.0	93.1	71.7	94.0	81.4	90.6	93.4	92.0
Minimap	88.9	94.8	91.8	94.6	**99.0**	**96.7**	59.6	83.8	69.7	91.2	**95.4**	**93.2**

Table 5.2 An overview of sensitivity and precision on simulated and real error-prone long read datasets. In both the PB and ONT simulated datasets, the best values, shown in boldface, are statistically significantly better than the other values. We derived these values from the best settings of each tool (according to the best F1 score) after a parameter search. We calculated confidence intervals for the sensitivity, specificity and F1 scores using three standard deviations around the observed values. In the worst case, the error never exceeded ±0.1%, ±0.1% and ±0.2% respectively.

For the simulated PB data, GraphMap has the highest F1 score (despite being designed for ONT data and not PB data) followed by DALIGNER, Minimap, MHAP, and BLASR (Table 5.2). For the real PB data, DALIGNER has the highest F1 score followed by GraphMap, MHAP, and BLASR. For both the simulated and real ONT datasets, Minimap was the best method, yielding the highest F1 score, followed by GraphMap, MHAP and BLASR (Table 5.2).
Overall, these results suggest that some tools may perform substantially differently on data from different platforms. We hypothesize that differences in the read length distributions and error type frequencies could be responsible for this behaviour.

5.7.2 Computational performance

To measure the computational performance of each method, we ran each tool with default parameters (with some exceptions see Section D.2), as well as another run with optimized parameters yielding the highest F1 score (Table 2 and Section D.1) obtained after a parameter sweep on the simulated datasets. We note that GraphMap's owler mode could not be parameterized, except for choosing the number of threads, so there was no difference in the settings for default and highest F1 score parameterization runs. We ran our tests serially on the same 64-core Intel Xeon CPU E7-8867 v3 @ 2.50GHz machine with 2.5TB of memory. We measured the peak memory, CPU and wall clock time across read subsets to show the scalability of each method.

We investigated the scalability of the methods, testing them using 4, 8, 16 or 32 threads of execution on the *E. coli* datasets (Figures D.5-12). Despite specifying the number of threads,

each tool often used more or fewer threads than expected (Figures D.5,6,13,14). In particular, MHAP tended to use more threads than the number we specified.

On all tested *E. coli* datasets in our study, we observe that Minimap is the most computationally efficient tool, robustly producing overlap regions at least 3-4 times faster than all other methods, even when parameterizing for optimal F1 score (Figure D9,10). Determining the next fastest method is confounded by the effect of parameterization. For instance, when considering only our F1 score optimized settings, the execution time of DALIGNER was generally within an order of magnitude or less of the execution time of Minimap. On the other hand, DALIGNER can be 2-5 times slower than MHAP on some datasets under default parameters.

With default settings, DALIGNER performs up to 10 times slower than with F1 score optimized settings. This primarily occurs because the *k*-mer filtering threshold (-t) in the F1 optimized parameterization not only increases specificity but also reduces runtime. In contrast, our parameterization to optimize the F1 score in MHAP decreases the speed (by a factor of 3-4). In this case, the culprit was the sketch size (--num-hashes) used; larger sketch sizes increase sensitivity at the cost of time.

Finally, GraphMap is generally the least scalable method, the slowest when considering default parameters only, and only 1-2 times faster than BLASR when considering F1 optimized settings. BLASR is also able to scale better, using more threads than GraphMap (Figures D.9, 10).
In addition to its impressive computational performance, Minimap uses less memory than almost all methods on tested *E. coli* datasets (Figures D.11, 12), staying within an order of magnitude of

BLASR on average, despite the latter employing an FM-index. Memory usage in GraphMap seems to scale linearly with the number of reads at a rate nearly 10 times that of the BLASR or Minimap, likely owing to the hash table it uses. The memory usage characteristics of DALIGNER and MHAP are less clear, drastically changing depending on the parameters utilized. Overall MHAP has the worst memory performance even when using default parameters. The cause of the memory increase between optimized F1 and default setting in MHAP is again due to an increase in the sketch size between runs. Because of *k*-mer multiplicity filtering, DALIGNER's memory usage is 2-3 times lower when parameterized for an optimized F1 score. Many of the trends from the *C. elegans* datasets mirror the performance on the smaller *E. coli* dataset. Again, computational performance on the larger *C. elegans* datasets is still dominated by Minimap (Figure 5.5, Figures D13, 14), being at least 5 times faster than any other method. DALIGNER's performance seems to generally scale well, especially when *k*-mer filtering is performed (within an order of magnitude of Minimap). With default settings, MHAP is 2-3 times faster than DALIGNER, but is several orders of magnitude slower, when the F1 score is optimized. The performance of GraphMap shows that it does not scale well to a large number of reads (>100,000), and its calculations take an order of magnitude longer than BLASR.

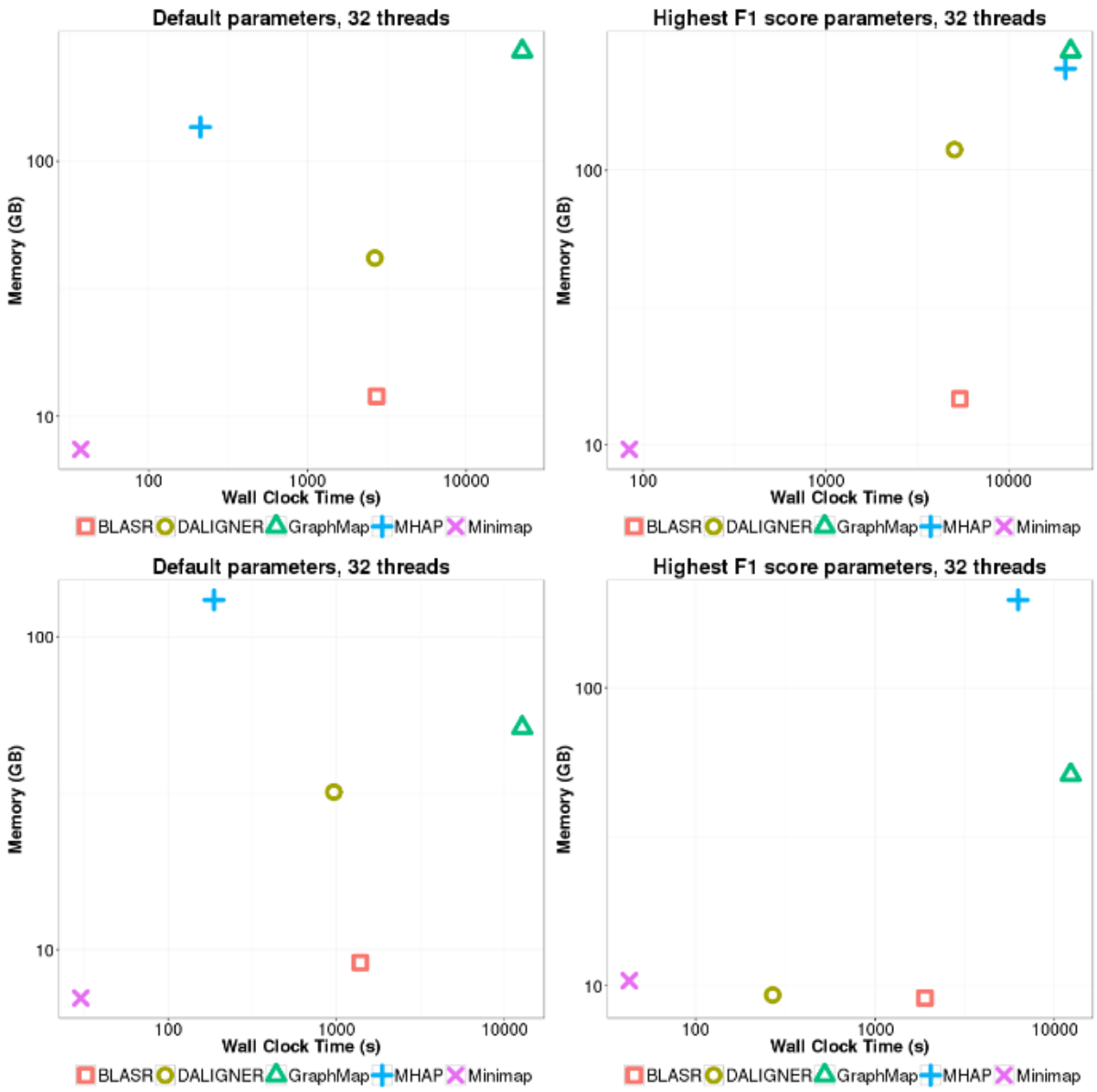


Figure 5.5 Wall clock time and memory on the PB P6-C4 (top) and simulated ONT (bottom) *C. elegans* dataset on 100000 randomly sampled reads. Each tool was parameterized using default settings (left) or using settings from runs yielding the highest F1 score on the simulated *E.*coli datasets (right).

Of note is the performance of BLASR when ran on PB and ONT datasets with default settings, which is roughly on par with that of DALIGNER (Figure 5.5, Figures D.13, 14). When using optimized parameters, BLASR is also at least twice as fast as MHAP (Figure 5.5, Figures D.13,

14). We had expected the computational time for DALIGNER and MHAP to scale better than BLASR on the large *C. elegans* datasets, as any upfront costs (e.g. MinHash sketch computation) would be amortized on larger datasets. We note that these results seemingly contradict the results found in previous studies [143, 144]. This may be due to different datasets and technology versions used by the two studies but to a greater extent this likely highlights the importance of careful parameterization of each tool. Specifically, for DALIGNER, it is important to filter the occurrences of *k*-mers (-t) in each batch to maintain not only specificity but also computational performance. For MHAP, increasing the sketch size increases the sensitivity of the initial filter but increases computation.

The trends in memory performance on the *C. elegans* datasets are generally consistent with *E. coli* datasets (Figure D.13, 14). A notable exception, however, is the memory usage of DALIGNER, which begins levelling off with increased number of reads. Unlike with the *E. coli* dataset, this dataset is large enough that DALIGNER begins to split the data into batches, reducing its memory usage.

5.8 Discussion

Our study highlights that there are important considerations to factor in while developing new tools or improving existing ones.

5.8.1 Modularity

A tool that can report intermediate results may help reduce computation in downstream applications. For example, modularizing overlap candidate detection, overlap validation, and

alignment can provide flexibility when used in different pipelines. GraphMap's owler mode is an example of this, enabling users to generate MHAP-like output for overlap regions, rather than a more detailed alignment on detected regions. Further, compliance with standardized output is highly recommended, including for generating intermediate results. Doing so would not only allow one to perform comparative performance evaluations on a variety of equivalent metrics but also allow for flexibility in creating new pipelines. Examples of emergent output standards include the Graphical Fragment Assembly (GFA) (https://github.com/pmelsted/GFA-spec) format, PAF (Li, 2016), and the MHAP output format.

5.8.2 Cache efficiency

Given the concepts presented, and along with our benchmarks performed herein indicates that theoretical performance estimations based on time complexity analysis might not be enough to conclude on what works best. Traditional algorithm complexity analysis suffers from an assumption that all memory access costs are the same. However, on modern computers, intermediate levels of fast-access cache exist between the registers of the CPU and main memory. A failed attempt to read or write data in the cache is called a cache miss, causing delays by requiring the algorithm to fetch data from other cache levels or main memory.

Cache efficiency in algorithmic design has become a major consideration, and in some cases will trump many time complexity-based motivations for algorithmic development. For instance, though the expected time complexity of DALIGNER has a quadratic component based on the number of occurrences of a *k*-mer in the dataset, its actual computational performance seems to be much better empirically. The authors claim this is due to the cache efficiency of the method

(compared to using an FM-index) [144], and in practice, this also seems to be the case, as observed in our comparisons.

The basic concept of a cache efficient algorithm relies on minimizing random access whenever possible, by serializing data accesses in blocks that are small enough to fit into various levels of cache, especially at the levels of cache with the lowest latency. Algorithms that exploit a specific cache configuration utilize an I/O-model (also called the external-memory model) [173, 174]. Conceptually, these algorithms must have explicit knowledge of the size of each component of the memory hierarchy and will adjust the size of contiguous blocks of data to minimize data transfers from memory to cache.

In contrast to the I/O model, algorithms that are designed with cache in mind, but do not explicitly rely on known cache size blocks are called cache-oblivious [175]. Cache oblivious algorithms are beneficial, as they do not rely on the knowledge of the processor architecture; instead they utilize classes of algorithms that are inherently cache efficient such as scanning algorithms (e.g. DALIGNER's merging step of a sorted list).

5.8.3 Batching and batch/block sizes

For many of the methods surveyed in this paper, memory usage scales linearly to the number of reads in the set, sometimes exceeding 100 GB on only 100,000 long reads (Figure 5.5, Figure D.12,13). Thus, to perform all necessary comparisons on large datasets (i.e. to compute an upper triangular matrix of candidate comparisons) the data must be processed in batches. Generally, it is better to use as few blocks as possible, since the time required to perform all overlaps is

quadratic relative to the number of batches. Methods that have a very low memory usage overall will be able to have the computational benefit of splitting the data into fewer batches. Batching is handled in different ways depending on the tool. Some tools have built-in splitting (DALIGNER/DAZZLER database with DBsplit and -I with Minimap), and others have this process built into their associated pipelines (e.g. MHAP and PBcR). Though BLASR may have more scalable memory usage, it will eventually require splitting as the index it uses is limited to 4 GB in size. To encourage the adoption of a tool in different pipeline contexts, built-in batching is a desirable attribute to ensure that memory usage scales with the dataset size.

5.8.4 Repetitive elements and sequence filtering

Any common regions due to homology or other repetitive elements may confound read-to-read overlaps and may be difficult to disambiguate from true overlaps. Such repetitive elements may lead to many false positives in overlap detection and may increase the computational burden, leading to lower quality in downstream assembly. Thus, it is common for overlap methods to employ sequence filtering, by removal or masking of repetitive elements to improve algorithmic performance both in run time and specificity. Many of the methods compared utilize *k*-mer frequencies to filter highly repetitive *k*-mers using an absolute or percent *k*-mer multiplicity (e.g. MHAP). These filtering approaches can be affected by the batch size when filters are seeded based on their multiplicity in each batch, rather than in the entire dataset (e.g. DALIGNER, Minimap). Another common filtering strategy is to prevent the use of low complexity sequences (e.g. DALIGNER, Minimap).

Some overlaps caused by repetitive elements can be useful for some assembly pipelines, such as in the identification of repeat boundaries. Thus, provided that it is computationally tractable, it may be beneficial for future overlap algorithms to assess the likelihood that an overlap is repeat-induced; this metric would ideally be standardized to facilitate modularity between pipelines.

5.8.5 Tuning for sensitivity and specificity

As we have presented in our review, methods to compute read-to-read overlap vary in their designs and implementations, although they share similar concepts and parameters. For example, many of the methods are *k*-mer based and, as expected, behave similarly when that parameter is tuned; for instance, increasing *k* increases specificity but decreases sensitivity especially on sequencing data with the high base error. At the current error rates for both PB and ONT technologies, a *k* of 16 is optimal, but it can be expected to increase as error rates decrease. Another common theme amongst read overlap tools is that many methods employ an initial candidate discovery stage, followed with a more specific candidate validation stage. Increasing sensitivity at the cost of specificity at the initial stage is typically tolerated by the second stage. Although, because the second stage is often computationally expensive, the parameters (e.g. –threshold in MHAP or –nCandidates in BLASR) controlling the number candidates considered for the validation stage must be tuned accordingly.

Growing in popularity is the use of another prominent method – minimizers to generate a reduced representation of each sequence – helping speed-up comparisons (e.g. MinHash sketches). Tuning the corresponding parameters (e.g. –w in Minimap and --num-hashes in

MHAP) in a way that more closely tracks with the original data representation will result in higher sensitivity and sensitivity but at a greater computational cost.

5.9 Conclusions

There are many challenges in evaluating algorithms that function on error-prone long reads, such as those from PB and ONT instruments. Although both sequencing technologies have comparable error rates, characteristics of their errors as well as their read length distributions are substantially different. Also, within each technology there are rapid improvements in quality [154, 155], causing disagreement between datasets derived from the same technology. Despite these issues, we show that Minimap is currently the most computationally efficient method (in both time and memory) and is the most specific and sensitive method on the ONT datasets tested. We note that Minimap is not as sensitive or as specific as Graphmap, DALIGNER or MHAP on the PB datasets tested. Our results have shown that GraphMap and DALIGNER are most specific and sensitive method on PB datasets tested, though DALIGNER scales better computationally. PB is a more mature technology compared to ONT, it is not surprising to see several tools performing well on the platform.

Here, we have provided an overview of leading read-to-read overlap detection methods, comparing their concepts and performances. We think our results will guide researchers to make informed decisions when choosing between these methods. As well, our elucidation to open problems may help developers improve on existing overlap detection tools or build new ones.

Chapter 6: Conclusion

The main research question in this thesis is whether sequence analysis can be improved through the use of probabilistic data structures and to explore the landscape of methodologies could work on newer emerging sequencing technologies. This thesis has four major results:

- I have shown Bloom filter-based classification methods to require less memory than an FM-index to index sequences and to use similar amounts of memory during classification. Bloom filter-based approaches have higher cache efficiency and can process sequence data more quickly.
- Progressive Bloom filters tractably and efficiently recruit genomic read sequences for targeted assembly, even when sequences have missing sequences not found in the *bait* set, such as introns or promoter sequences or divergent sequences (e.g., from other species).
- Multi-index Bloom filters show high computational efficiency and extremely high sensitivity for multi-target sequences classification (with applications for targeted read binning and metagenomics), owing to their synergistic pairing with spaced seeds.
- Overlap detection methods for long reads differ from each other and from older methods because of the high error rates found in modern long read datasets, and currently, minimizer-based methods show the highest scalability and performance.

Though, I have succeeded in addressing my stated objectives within this work, the subject matter of my thesis is a highly active area of research. Broadly, this work can still be expanded on by exploring the utility of current Illumina centric algorithms on newer data types and exploring

other applications traditionally carried out by alignment-based algorithms, supplanting them with alignment-free alternatives.

6.1 Addressing limitations of hash-based probabilistic data structures

The obvious key insights into whether a probabilistic method should be used are the consideration that whether quantifiable benefits (i.e. lower memory and faster runtime) outweigh the drawbacks (i.e. false positives and lossy data storage). I have shown that for datatypes with relatively low error, such as Illumina sequencing, probabilistic data structures make a lot of sense; however, if the trend long read data with a high indel data persists, my work with probabilistic data structures may become less relevant as the benefits of probabilistic storage lie in being able to reduce the memory footprint of long *k*-mers and spaced seeds. These methods shine when longer more complex seeds are used (i.e. long *k*-mers, and spaced seeds). Probabilistic methods lose their efficacy on high error data such as long read data (with high rates of indels). However, there may be additional benefits as longer reads provide more real estate, so to speak, to allow for chances for a longer *k*-mer to be found, especially if error rates drop further. Indeed, fast classification using Bloom filters is something that can be done on long reads, and BBT could be potentially parameterized to do so. In addition, BBT could be potentially coupled with a minimizer-based subsampling strategy to reduce the memory usage further (in a way similar to co-occurrence filters [36]).

6.2 Applying probabilistic hashing methods on long read technologies

This thesis offers several novel and effective solutions to address common memory and time resource limitations in analyzing large sequence datasets, a technology landscape currently

dominated by Illumina. However, if the long read technologies become more widely used, and their data continue to have substantially higher error rates, especially in the form of indels, my work with probabilistic data structures in its current form will likely become less relevant. This is because the benefits of probabilistic storage lie in being able to reduce the memory footprint of long *k*-mers and spaced seeds, which do not work well with data with high indel rates. However, they may also remain relevant if indel error rates drop further, as longer reads provide more real estate, so to speak, to allow for chances for a longer *k*-mer to be found. Indeed, fast classification using Bloom filters is something that can be done on long reads, and BBT could be potentially modified to do so. For Bloom filter classification in BBT, this would likely require implementing a different scoring algorithm based on probability, like the formulation featured in Chapter 4, since this datatype features variable read lengths.

6.3 Multi-Index Bloom filters as an approximate read mapper

Theoretically, base-pair accurate mapping of sequences without alignment is possible using miBFs. After all, miBFs can already index arbitrary IDs, so one can imagine that one can simply add index a sequence with not just a genome ID but position. However, to reduce the number of IDs (and thus, potentially reduce memory), one may want to index tiles of sequence rather than each position. More precisely, if mapping short reads, exact base pair positions would simply require that index positions in tiles to be shorter than the read length. The junction seeds between 2 tiles can be used to anchor the sequence.

Currently, this would work if DNA were not double-stranded, and would require further modifications to presented algorithms to allow for strand mapping. The hashing scheme used in

BBT is ntHash [133], which hashes only the canonical sequence (forward or reverse sequence, whichever returns the lower hash value). To obtain direction of the read sequence, a record of what strand was picked as the canonical sequence is needed. This can be stored as part of the ID, similar to the way the saturation bit is set. A potential scheme could be an ID with the first two bits reserved, one for saturation, and the other for the sign bit (relative to the reference). When mapping, the sign of the frames can be compared to what is seen in the filter, and direction assigned accordingly.

However, some complications remain for full implementation for this use case. First, the false-positive formulation in Chapter 4 would be complicated, as adjacent frames may contain two or more IDs. Second, the presence of error may erase the junction sequence used for anchoring; in these cases, the algorithm would not be able to assign an exact position. This can be mitigated by reducing the size of each tile decreeing the range of positions the sequence can map to. Application of this possible algorithm would cater to situations where close but not exact alignments would be sufficient. An example of this would be in the creation of a paired end graph during genome assembly. The only information needed in this case would be if two contigs are linked by a set of paired reads, and this does not require an actual base pair to base-pair alignment.

6.4 Optimizing spaced seed design for alignment-free mapping

Though discussed extensively, notably absent from Chapter 4 was the extensive design of the spaced seeds used. The issue is that spaced seed design is still an uncharted territory when it comes to direct mapping applications, especially when the seeds used can be of any size when

represented by probabilistic data structures. Our results showed that seed design has a limited effect, suggesting that extensive effort in design may not be called for; that is, random seeds may work well enough compared to a theoretical optimal seed, and that only specifically designed poor seeds have notably poorer performance, although still potentially outperforming *k*-mers. Nevertheless, seed design can be considered a one-time cost depending on the application, thus the cost associated with exploring methods to optimize the seeds may be amortized and may be worth exploring in future work.

6.5 Broader outlook

This work has illustrated novel data types to help reduce the computational burden of sequence analysis and made possible analysis that would be intractable otherwise. As high-throughput sequence technologies advance, computational demands will only grow higher. At present, the methods described in this thesis have been applied in practical settings. BBT is used at the GSC for not only contamination screening of all sequencing data generated, but also in pathogen detection[86-88, 176]. Kollector has been used to enrich the genic space of our *de novo* spruce genome [95] and bullfrog [177] assemblies. As a more recent data type, it is currently unknown what applications miBF will enable. Potential direct applications would include adapting it as the engine for a full-fledged metagenomics classification pipeline. As a novel data structure with some interesting properties, such as reducing FPR for autocorrelated data, it may find applications outside of bioinformatics, in other computational settings.

Bibliography

1. *DNA Sequencing Costs: Data - National Human Genome Research Institute (NHGRI)*. Available from: https://www.genome.gov/27541954/dna-sequencing-costs-data/.

2. Moore, G.E., *Cramming More Components Onto Integrated Circuits.* Proc. IEEE, 1998. **86**(1): p. 82-85.

3. Ferragina, P. and G. Manzini, *Opportunistic data structures with applications*, in *Proceedings 41st Annual Symposium on Foundations of Computer Science*.

4. Pevzner, P.A., H. Tang, and M.S. Waterman, *An Eulerian path approach to DNA fragment assembly.* Proceedings of the National Academy of Sciences, 2001. **98**(17): p. 9748-9753.

5. Eid, J., et al., *Real-Time DNA Sequencing from Single Polymerase Molecules.* Science, 2009. **323**(5910): p. 133-138.

6. Stoddart, D., et al., *Single-nucleotide discrimination in immobilized DNA oligonucleotides with a biological nanopore.* Proc. Natl. Acad. Sci. U. S. A., 2009. **106**(19): p. 7702-7707.

7. Eisenstein, M., *Startups use short-read data to expand long-read sequencing market.* Nat. Biotechnol., 2015. **33**(5): p. 433-435.

8. Smith, T.F. and M.S. Waterman, *Identification of common molecular subsequences.* J. Mol. Biol., 1981. **147**(1): p. 195-197.

9. *The Cancer Genome Atlas - Data Portal*. Available from: https://tcga-data.nci.nih.gov/docs/publications/tcga/.

10. Warren, R.L., et al., *Improved white spruce (Picea glauca) genome assemblies and annotation of large gene families of conifer terpenoid and phenolic defense metabolism.* Plant J., 2015. **83**(2): p. 189-212.

11. Jamshidi, F., et al., *Diagnostic Value of Next-Generation Sequencing in an Unusual Sphenoid Tumor.* Oncologist, 2014. **19**(6): p. 623-630.

12. Sheffield, B.S., et al., *Personalized oncogenomics: clinical experience with malignant peritoneal mesothelioma using whole genome sequencing.* PLoS One, 2015. **10**(3): p. e0119689.

13. Durbin, R., *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. 1998: Cambridge University Press. 356.

14. Bazinet, A.L. and M.P. Cummings, *A comparative evaluation of sequence classification programs.* BMC Bioinformatics, 2012. **13**(1): p. 92.

15. Chu, J., et al., *BioBloom tools: fast, accurate and memory-efficient host species sequence screening using bloom filters.* Bioinformatics, 2014. **30**(23): p. 3402-3404.

16. Kostic, A.D., et al., *PathSeq: software to identify or discover microbes by deep sequencing of human tissue.* Nat. Biotechnol., 2011. **29**(5): p. 393-396.

17. Peabody, M.A., et al., *Evaluation of shotgun metagenomics sequence classification methods using in silico and in vitro simulated communities.* BMC Bioinformatics, 2015. **16**: p. 363.

18. Brown, S.D., et al., *Neo-antigens predicted by tumor genome meta-analysis correlate with increased patient survival.* Genome Res., 2014. **24**(5): p. 743-750.

19. Warren, R.L., et al., *Derivation of HLA types from shotgun sequence datasets.* Genome Med., 2012. **4**(12): p. 95.

20. Chiu, R., et al., *TAP: a targeted clinical genomics pipeline for detecting transcript variants using RNA-seq data.* BMC medical genomics, 2018. **11**(1): p. 79.

21. Valouev, A., et al., *A high-resolution, nucleosome position map of C. elegans reveals a lack of universal sequence-dictated positioning.* Genome Res., 2008. **18**(7): p. 1051-1063.

22. Canard, B., C. Bruno, and R.S. Sarfati, *DNA polymerase fluorescent substrates with reversible 3'-tags.* Gene, 1994. **148**(1): p. 1-6.

23. Zheng, G.X.Y., et al., *Haplotyping germline and cancer genomes with high-throughput linked-read sequencing.* Nat. Biotechnol., 2016. **34**(3): p. 303-311.

24. Sanger, F. and A.R. Coulson, *A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase.* J. Mol. Biol., 1975. **94**(3): p. 441-448.

25. Ronaghi, M., et al., *Real-time DNA sequencing using detection of pyrophosphate release.* Anal. Biochem., 1996. **242**(1): p. 84-89.

26. Rusk, N. and R. Nicole, *Torrents of sequence.* Nat. Methods, 2010. **8**(1): p. 44-44.

27. Bentley, D.R., et al., *Accurate whole human genome sequencing using reversible terminator chemistry.* Nature, 2008. **456**(7218): p. 53.

28. *Press Release*. Available from: http://www.illumina.com/company/news-center/press-releases/press-release-details.html?newsid=1676396.

29. Mullis, K.B., *The unusual origin of the polymerase chain reaction.* Scientific American, 1990. **262**(4): p. 56-65.

30. *Chromium Product Brochure*. Available from: http://www.10xgenomics.com/wp-content/uploads/2016/02/Chromium_Product_Brochure.pdf.

31. Laehnemann, D., A. Borkhardt, and A.C. McHardy, *Denoising DNA deep sequencing data-high-throughput sequencing errors and their correction.* Brief. Bioinform., 2016. **17**(1): p. 154-179.

32. Llc, G.B., *Probabilistic Data Structures: Bloom Filter*. 2010: General Books. 46.

33. Bloom, B.H., *Space/time trade-offs in hash coding with allowable errors.* Commun. ACM, 1970. **13**(7): p. 422-426.

34. Putze, F., et al., *Cache-, Hash- and Space-Efficient Bloom Filters,* in *Lecture Notes in Computer Science*. p. 108-121.

35. Mitzenmacher, M. and M. Michael, *Compressed bloom filters,* in *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing - PODC '01*. 2001.

36. Tirdad, K., et al., *COCA Filters: Co-occurrence Aware Bloom Filters,* in *Lecture Notes in Computer Science*. 2011. p. 313-325.

37. Cohen, S., C. Saar, and M. Yossi, *Spectral bloom filters*, in *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD '03*. 2003.

38. Cen, Z., et al., *A multi-layer bloom filter for duplicated URL detection*, in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*. 2010.

39. Lall, A. and M. Ogihara, *The Bitwise Bloom Filter*. 2007.

40. Deng, F., D. Fan, and R. Davood, *Approximately detecting duplicates for streaming data using stable bloom filters,* in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06*. 2006.

41. Chazelle, B., et al., *The Bloomier filter: an efficient data structure for static support lookup tables,* in *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*. 2004, Society for Industrial and Applied Mathematics. p. 30-39.

42. Marchet, C., et al., *A resource-frugal probabilistic dictionary and applications in bioinformatics.* Discrete Applied Mathematics, 2018.

43. Solomon, B. and C. Kingsford, *Fast search of thousands of short-read sequencing experiments.* Nature biotechnology, 2016. **34**(3): p. 300.

44. Bender, M.A., et al., *Don't thrash.* Proceedings VLDB Endowment, 2012. **5**(11): p. 1627-1637.

45. Pandey, P., et al., *Squeakr: an exact and approximate k-mer counting system.* Bioinformatics, 2017. **34**(4): p. 568-575.

46. Marchet, C., et al., *A resource-frugal probabilistic dictionary and applications in (meta) genomics.* arXiv, 2016.

47. Yu, Y., et al., *Memory-efficient and ultra-fast network lookup and forwarding using Othello hashing.* IEEE/ACM Transactions on Networking, 2018. **26**(3): p. 1151-1164.

48. Liu, X., et al., *A novel data structure to support ultra-fast taxonomic classification of metagenomic sequences with k-mer signatures.* Bioinformatics, 2018. **34**(1): p. 171-178.

49. Dadi, T.H., et al., *DREAM-Yara: An exact read mapper for very large databases with short update time.* bioRxiv, 2018: p. 256354.

50. Solomon, B. and C. Kingsford, *Improved search of large transcriptomic sequencing databases using split sequence bloom trees.* Journal of Computational Biology, 2018. **25**(7): p. 755-765.

51. Pandey, P., et al., *Mantis: A fast, small, and exact large-scale sequence-search index.* Cell systems, 2018. **7**(2): p. 201-207. e4.

52. Limasset, A., et al., *Fast and scalable minimal perfect hashing for massive key sets.* arXiv preprint arXiv:.03154, 2017.

53. Almodaresi, F., et al., *A space and time-efficient index for the compacted colored de Bruijn graph.* Bioinformatics, 2018. **34**(13): p. i169-i177.

54. Iqbal, Z., et al., *De novo assembly and genotyping of variants using colored de Bruijn graphs.* Nature genetics, 2012. **44**(2): p. 226.

55. Yu, Y., et al., *SeqOthello: querying RNA-seq experiments at scale.* Genome biology, 2018. **19**(1): p. 167.

56. Natarajan, A. and S. Subramanian. *Bloom filter optimization using cuckoo search*. in *2012 International Conference on Computer Communication and Informatics*. 2012. IEEE.

57. Jacobson, G., *Succinct Static Data Structures*. 1989. 101.

58. Li, H. and R. Durbin, *Fast and accurate short read alignment with Burrows-Wheeler transform.* Bioinformatics, 2009. **25**(14): p. 1754-1760.

59. Langmead, B., L. Ben, and S.L. Salzberg, *Fast gapped-read alignment with Bowtie 2.* Nat. Methods, 2012. **9**(4): p. 357-359.

60. Raman, R., et al., *Succinct indexable dictionaries with applications to encoding k -ary trees, prefix sums and multisets.* ACM Trans. Algorithms, 2007. **3**(4): p. 43-es.

61. Sadakane, K., S. Kunihiko, and G. Roberto, *Squeezing succinct data structures into entropy bounds*, in *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm - SODA '06*. 2006.

62. Higashi, S., et al., *Analysis of composition-based metagenomic classification.* BMC Genomics, 2012. **13 Suppl 5**: p. S1.

63. Wood, D.E. and S.L. Salzberg, *Kraken: ultrafast metagenomic sequence classification using exact alignments.* Genome Biol., 2014. **15**(3): p. R46.

64. Ounit, R., et al., *CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers.* BMC Genomics, 2015. **16**: p. 236.

65. Stranneheim, H., et al., *Classification of DNA sequences using Bloom filters.* Bioinformatics, 2010. **26**(13): p. 1595-1600.

66. Ma, B., J. Tromp, and M. Li, *PatternHunter: faster and more sensitive homology search.* Bioinformatics, 2002. **18**(3): p. 440-445.

67. Břinda, K., M. Sykulski, and G. Kucherov, *Spaced seeds improve k-mer-based metagenomic classification.* Bioinformatics, 2015. **31**(22): p. 3584-3592.

68. Ounit, R. and S. Lonardi, *Higher classification sensitivity of short metagenomic reads with CLARK-S.* Bioinformatics, 2016.

69. Kucherov, G., et al., *A UNIFYING FRAMEWORK FOR SEED SENSITIVITY AND ITS APPLICATION TO SUBSET SEEDS.* J. Bioinform. Comput. Biol., 2006. **04**(02): p. 553-569.

70. Hahn, L., et al., *rasbhari: Optimizing Spaced Seeds for Database Searching, Read Mapping and Alignment-Free Sequence Comparison.* PLoS Comput. Biol., 2016. **12**(10): p. e1005107.

71. Egidi, L. and G. Manzini, *Multiple seeds sensitivity using a single seed with threshold.* J. Bioinform. Comput. Biol., 2015. **13**(4): p. 1550011.

72. Noé, L. and G. Kucherov, *Improved hit criteria for DNA local alignment.* BMC Bioinformatics, 2004. **5**: p. 149.

73. Li, M., et al., *Patternhunter II: highly sensitive and fast homology search.* J. Bioinform. Comput. Biol., 2004. **2**(3): p. 417-439.

74. Ilie, L. and S. Ilie, *Multiple spaced seeds for homology search.* Bioinformatics, 2007. **23**(22): p. 2969-2977.

75. Pop, M., *Genome assembly reborn: recent computational challenges.* Brief. Bioinform., 2009. **10**(4): p. 354-366.

76. Myers, E.W., *A Whole-Genome Assembly of Drosophila.* Science, 2000. **287**(5461): p. 2196-2204.

77. Simpson, J.T. and P. Mihai, *The Theory and Practice of Genome Sequence Assembly.* Annu. Rev. Genomics Hum. Genet., 2015. **16**(1): p. 153-172.

78. Chapman, J.A., et al., *Meraculous: de novo genome assembly with short paired-end reads.* PLoS One, 2011. **6**(8): p. e23501.

79. Simpson, J.T. and R. Durbin, *Efficient de novo assembly of large genomes using compressed data structures.* Genome Res., 2012. **22**(3): p. 549-556.

80. Zerbino, D.R. and E. Birney, *Velvet: Algorithms for de novo short read assembly using de Bruijn graphs.* Genome Res., 2008. **18**(5): p. 821-829.

81. Maccallum, I., et al., *ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads.* Genome Biol., 2009. **10**(10): p. R103.

82. Simpson, J.T., et al., *ABySS: a parallel assembler for short read sequence data.* Genome Res., 2009. **19**(6): p. 1117-1123.

83. Warren, R.L. and R.A. Holt, *Targeted assembly of short sequence reads.* PLoS One, 2011. **6**(5): p. e19816.

84. Peterlongo, P. and R. Chikhi, *Mapsembler, targeted and micro assembly of large NGS datasets on a desktop computer.* BMC Bioinformatics, 2012. **13**: p. 48.

85. Allen, J.M., et al., *aTRAM - automated target restricted assembly method: a fast method for assembling loci across divergent taxa from next-generation sequencing data.* BMC Bioinformatics, 2015. **16**(1).

86. Cancer Genome Atlas, N., *Comprehensive genomic characterization of head and neck squamous cell carcinomas.* Nature, 2015. **517**(7536): p. 576-582.

87. Zheng, S., et al., *Comprehensive pan-genomic characterization of adrenocortical carcinoma.* Cancer cell, 2016. **29**(5): p. 723-736.

88. Robertson, A.G., et al., *Comprehensive molecular characterization of muscle-invasive bladder cancer.* Cell, 2017. **171**(3): p. 540-556. e25.

89. Tang, K.-W., et al., *The landscape of viral expression and host gene fusion and adaptation in human cancer.* Nature communications, 2013. **4**.

90. Xu, G., et al., *RNA CoMPASS: A Dual Approach for Pathogen and Host Transcriptome Analysis of RNA-Seq Datasets.* PloS one, 2014. **9**(2): p. e89445.

91. Kostic, A.D., et al., *PathSeq: software to identify or discover microbes by deep sequencing of human tissue.* Nature biotechnology, 2011. **29**(5): p. 393.

92. Castellarin, M., et al., *Fusobacterium nucleatum infection is prevalent in human colorectal carcinoma.* Genome research, 2012. **22**(2): p. 299-306.

93. Broder, A. and M. Mitzenmacher, *Network applications of bloom filters: A survey.* Internet mathematics, 2004. **1**(4): p. 485-509.

94. Ferragina, P., T. Gagie, and G. Manzini, *Lightweight data indexing and compression in external memory.* Algorithmica, 2012. **63**(3): p. 707-730.

95. Kucuk, E., et al., *Kollector: transcript-informed, targeted de novo assembly of gene loci.* Bioinformatics, 2017. **33**(12): p. 1782-1788.

96. Nagarajan, N. and M. Pop, *Sequence assembly demystified.* Nat Rev Genet, 2013. **14**(3): p. 157-167.

97. Hahn, C., L. Bachmann, and B. Chevreux, *Reconstructing mitochondrial genomes directly from genomic next-generation sequencing reads—a baiting and iterative mapping approach.* Nucleic acids research, 2013: p. gkt371.

98. Brankovics, B., et al., *GRAbB: selective assembly of genomic regions, a new niche for genomic research.* PLoS Comput Biol, 2016. **12**(6): p. e1004753.

99. Altschul, S.F., et al., *Basic local alignment search tool.* J. Mol. Biol., 1990. **215**(3): p. 403-410.

100. Grabherr, M.G., et al., *Full-length transcriptome assembly from RNA-Seq data without a reference genome.* Nat Biotechnol, 2011. **29**(7): p. 644-652.

101. Peng, Y., et al., *IDBA-tran: a more robust de novo de Bruijn graph assembler for transcriptomes with uneven expression levels.* Bioinformatics, 2013. **29**(13): p. i326-i334.

102. Robertson, G., et al., *De novo assembly and analysis of RNA-seq data.* Nature Methods, 2010. **7**(11): p. 909-912.

103. Wu, T.D. and C.K. Watanabe, *GMAP: a genomic mapping and alignment program for mRNA and EST sequences.* Bioinformatics, 2005. **21**(9): p. 1859-1875.

104. Altschul, S., *Basic Local Alignment Search Tool.* Journal of Molecular Biology, 1990. **215**(3): p. 403-410.

105. Warren, R.L., et al., *Improved white spruce (Picea glauca) genome assemblies and annotation of large gene families of conifer terpenoid and phenolic defense metabolism.* Plant J, 2015. **83**(2): p. 189-212.

106. Birol, I., et al., *Assembling the 20 Gb white spruce (Picea glauca) genome from whole-genome shotgun sequencing data.* Bioinformatics, 2013.

107. Johnson, K.P., et al., *Rates of genomic divergence in humans, chimpanzees and their lice.* Proc. Biol. Sci., 2014. **281**(1777): p. 20132174.

108. Alkan, C., B.P. Coe, and E.E. Eichler, *Genome structural variation discovery and genotyping.* Nat Rev Genet, 2011. **12**(5): p. 363-76.

109. Chu, J., et al., *Improving on hash-based probabilistic sequence classification using multiple spaced seeds and multi-index Bloom filters.* BioRxiv, 2018: p. 434795.

110. Peabody, M.A., et al., *Evaluation of shotgun metagenomics sequence classification methods using in silico and in vitro simulated communities.* BMC bioinformatics, 2015. **16**(1): p. 362.

111. Burkhardt, S. and J. Kärkkäinen. *One-gapped q-gram filters for Levenshtein distance*. in *Annual Symposium on Combinatorial Pattern Matching*. 2002. Springer.

112. Ghandi, M., et al., *Enhanced regulatory sequence prediction using gapped k-mer features.* PLoS Comput Biol, 2014. **10**(7): p. e1003711.

113. Noe, L. and D.E. Martin, *A coverage criterion for spaced seeds and its applications to support vector machine string kernels and k-mer distances.* J Comput Biol, 2014. **21**(12): p. 947-63.

114. Ilie, L., et al., *BOND: Basic OligoNucleotide Design.* BMC Bioinformatics, 2013. **14**: p. 69.

115. Kucherov, G., L. Noe, and M. Roytberg, *A unifying framework for seed sensitivity and its application to subset seeds.* J Bioinform Comput Biol, 2006. **4**(2): p. 553-69.

116. Ilie, L., S. Ilie, and A.M. Bigvand, *SpEED: fast computation of sensitive spaced seeds.* Bioinformatics, 2011. **27**(17): p. 2433-4.

117. Qi, J., H. Luo, and B. Hao, *CVTree: a phylogenetic tree reconstruction tool based on whole genomes.* Nucleic Acids Res., 2004. **32**(Web Server issue): p. W45-7.

118. Pellow, D., D. Filippova, and C. Kingsford, *Improving Bloom filter performance on sequence data using k-mer Bloom filters.* Journal of Computational Biology, 2017. **24**(6): p. 547-557.

119. Li, H., *Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM.* arXiv [q-bio.GN], 2013.

120. Hu, X., et al., *pIRS: Profile-based Illumina pair-end reads simulator*. Bioinformatics, 2012. **28**(11): p. 1533-1535.

121. Forbes, S.A., et al., *COSMIC: somatic cancer genetics at high-resolution.* Nucleic Acids Res., 2017. **45**(D1): p. D777-D783.

122. O'Leary, N.A., et al., *Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation.* Nucleic Acids Res., 2016. **44**(D1): p. D733-45.

123. Lindgreen, S., K.L. Adair, and P.P. Gardner, *An evaluation of the accuracy and speed of metagenome analysis tools.* Sci. Rep., 2016. **6**: p. 19233.

124. Li, M., B. Ma, and L. Zhang, *Superiority and complexity of the spaced seeds*, in *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm - SODA '06*. 2006.

125. Girotto, S., M. Comin, and C. Pizzi, *Fast Spaced Seed Hashing*, in *17th International Workshop on Algorithms in Bioinformatics (WABI 2017)*. 2017, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. p. 14.

126. Markov, A., *Extension of the limit theorems of probability theory to a sum of variables connected in a chain.* Dynamic probabilistic systems, 1971(1): p. 552-577.

127. Shannon, C.E., *A mathematical theory of communication.* Bell system technical journal, 1948. **27**(3): p. 379-423.

128. Gog, S. and M. Petri, *Optimized succinct data structures for massive data.* Softw. Pract. Exp., 2013. **44**(11): p. 1287-1314.

129. Vitter, J.S., *Random sampling with a reservoir.* ACM Trans. Math. Softw., 1985. **11**(1): p. 37-57.

130. Pellow, D., D. Filippova, and C. Kingsford, *Improving Bloom Filter Performance on Sequence Data Using $$k$$ -mer Bloom Filters*, in *Lecture Notes in Computer Science*. 2016. p. 137-151.

131. Dunn, O.J., *Estimation of the Medians for Dependent Variables.* Ann. Math. Stat., 1959. **30**(1): p. 192-197.

132. Alefeld, G.E., F.A. Potra, and Y. Shi, *Algorithm 748; enclosing zeros of continuous functions.* ACM Trans. Math. Softw., 1995. **21**(3): p. 327-344.

133. Mohamadi, H., et al., *ntHash: recursive nucleotide hashing.* Bioinformatics, 2016.

134. Schäling, B., *The boost C++ libraries*. 2011: Boris Schäling.

135. Chu, J., et al., *Innovations and challenges in detecting long read overlaps: an evaluation of the state-of-the-art.* Bioinformatics, 2016. **33**(8): p. 1261-1270.

136. Ross, M.G., et al., *Characterizing and measuring bias in sequence data.* Genome Biol., 2013. **14**(5): p. R51.

137. Smith, D.R., et al., *Rapid whole-genome mutational profiling using next-generation sequencing technologies.* Genome Res., 2008. **18**(10): p. 1638-1642.

138. Alkan, C., et al., *Limitations of next-generation genome sequence assembly.* Nat. Methods, 2010. **8**(1): p. 61-65.

139. Treangen, T.J. and S.L. Salzberg, *Repetitive DNA and next-generation sequencing: computational challenges and solutions.* Nat. Rev. Genet., 2012. **13**(1): p. 36-46.

140. McCoy, R.C., et al., *Illumina TruSeq synthetic long-reads empower de novo assembly and resolve complex, highly-repetitive transposable elements.* PLoS One, 2014. **9**(9): p. e106689.

141. Ummat, A. and A. Bashir, *Resolving complex tandem repeats with long reads.* Bioinformatics, 2014. **30**(24): p. 3491-3498.

142. Loman, N.J., J. Quick, and J.T. Simpson, *A complete bacterial genome assembled de novo using only nanopore sequencing data.* Nat. Methods, 2015. **12**(8): p. 733-735.

143. Berlin, K., et al., *Assembling large genomes with single-molecule sequencing and locality-sensitive hashing.* Nat. Biotechnol., 2015. **33**(6): p. 623-630.

144. Myers, G., *Efficient Local Alignment Discovery amongst Noisy Long Reads*, in *Algorithms in Bioinformatics*. 2014, Springer Berlin Heidelberg. p. 52-67.

145. Chin, C.-S., et al., *Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data.* Nat. Methods, 2013. **10**(6): p. 563-569.

146. Li, H., *Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences.* Bioinformatics, 2016.

147. Levene, M.J., et al., *Zero-mode waveguides for single-molecule analysis at high concentrations.* Science, 2003. **299**(5607): p. 682-686.

148. Carneiro, M.O., et al., *Pacific biosciences sequencing technology for genotyping and variation discovery in human data.* BMC Genomics, 2012. **13**: p. 375.

149. Jiao, X., et al., *A benchmark study on error assessment and quality control of CCS reads derived from the PacBio RS.* Journal of data mining in genomics & proteomics, 2013. **4**(3).

150. O'Donnell, C.R., H. Wang, and W.B. Dunbar, *Error analysis of idealized nanopore sequencing.* Electrophoresis, 2013. **34**(15): p. 2137-2144.

151. Ewing, B., et al., *Base-calling of automated sequencer traces usingPhred. I. Accuracy assessment.* Genome research, 1998. **8**(3): p. 175-185.

152. Travers, K.J., et al., *A flexible and efficient template format for circular consensus sequencing and SNP detection.* Nucleic Acids Res., 2010. **38**(15): p. e159.

153. Richards, S. and S.C. Murali, *Best practices in insect genome sequencing: what works and what doesn't.* Current opinion in insect science, 2015. **7**: p. 1-7.

154. Laver, T., et al., *Assessing the performance of the Oxford Nanopore Technologies MinION.* Biomol Detect Quantif, 2015. **3**: p. 1-8.

155. Jain, M., et al., *Improved data analysis for the MinION nanopore sequencer.* Nat. Methods, 2015. **12**(4): p. 351-356.

156. Goodwin, S., et al., *Oxford Nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome.* Genome Res., 2015. **25**(11): p. 1750-1756.

157. David, M., et al., *Nanocall: An Open Source Basecaller for Oxford Nanopore Sequencing Data*. 2016.

158. Boža, V., B. Brejová, and T. Vinař, *DeepNano: deep recurrent neural networks for base calling in MinION nanopore reads.* PloS one, 2017. **12**(6): p. e0178751.

159. Chaisson, M.J. and G. Tesler, *Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory.* BMC Bioinformatics, 2012. **13**: p. 238.

160. Sović, I., et al., *Fast and sensitive mapping of nanopore sequencing reads with GraphMap.* Nat. Commun., 2016. **7**: p. 11307.

161. Waterman, M.S., *Dynamic Programming Alignment of Two Sequences*, in *Introduction to Computational Biology*. 1995. p. 183-232.

162. Sović, I., et al., *Evaluation of hybrid and non-hybrid methods for de novo assembly of nanopore reads.* Bioinformatics, 2016.

163. Ferragina, P., F. Paolo, and M. Giovanni, *Indexing compressed text.* J. ACM, 2005. **52**(4): p. 552-581.

164. Morgulis, A., et al., *A fast and symmetric DUST implementation to mask low-complexity DNA sequences.* Journal of Computational Biology, 2006. **13**(5): p. 1028-1040.

165. Broder, A.Z., *On the resemblance and containment of documents*, in *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*. 1997, IEEE. p. 21-29.

166. Marçais, G. and C. Kingsford, *A fast, lock-free approach for efficient parallel counting of occurrences of k-mers.* Bioinformatics, 2011. **27**(6): p. 764-770.

167. Wang, J.R. and C.D. Jones, *Fast alignment filtering of nanopore sequencing reads using locality-sensitive hashing*, in *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 2015.

168. Burkhardt, S., B. Stefan, and K. Juha, *One-Gapped q-Gram Filters for Levenshtein Distance*, in *Lecture Notes in Computer Science*. 2002. p. 225-234.

169. Keich, U., et al., *On spaced seeds for similarity search.* Discrete Appl. Math., 2004. **138**(3): p. 253-263.

170. Benson, G., et al., *Longest Common Subsequence in k Length Substrings*, in *Lecture Notes in Computer Science*. 2013. p. 257-265.

171. Ono, Y., K. Asai, and M. Hamada, *PBSIM: PacBio reads simulator--toward accurate genome assembly.* Bioinformatics, 2013. **29**(1): p. 119-121.

172. Yang, C., et al., *NanoSim: nanopore sequence read simulator based on statistical characterization*. 2016.

173. Aggarwal, A., et al., *The input/output complexity of sorting and related problems.* Commun. ACM, 1988. **31**(9): p. 1116-1127.

174. Demaine, E.D., *Cache-oblivious algorithms and data structures.* Lecture Notes from the EEF Summer School on Massive Data Sets, 2002. **8**(4): p. 1-249.

175. Frigo, M., et al. *Cache-oblivious algorithms*. in *Foundations of Computer Science, 1999. 40th Annual Symposium on*. 1999. IEEE.

176. Network, C.G.A.R., *Integrated genomic and molecular characterization of cervical cancer*. Nature, 2017. **543**(7645): p. 378.

177. Hammond, S.A., et al., *The North American bullfrog draft genome provides insight into hormonal regulation of long noncoding RNA.* Nature communications, 2017. **8**(1): p. 1433.

Appendices

Appendix A Chapter 2 Supplementary material

A.1 Bloom filter false positive rate in relation to the number of elements, size of filter and hash functions

The size of the filter per element with a given the false positive rate (FPR) is [93]:

$$f = \left(1 - \left(1 - \frac{1}{m}\right)^{hn}\right)^{h}$$

where *f* is the FPR, *m* is the size of the filter, *n* is the number of elements and *h* is the number of hash functions. This can be approximated as:

$$f = \left(1 - e^{-hn/m}\right)^{h}$$

The optimal number of hash functions [35] for a Bloom filter with respect to size is:

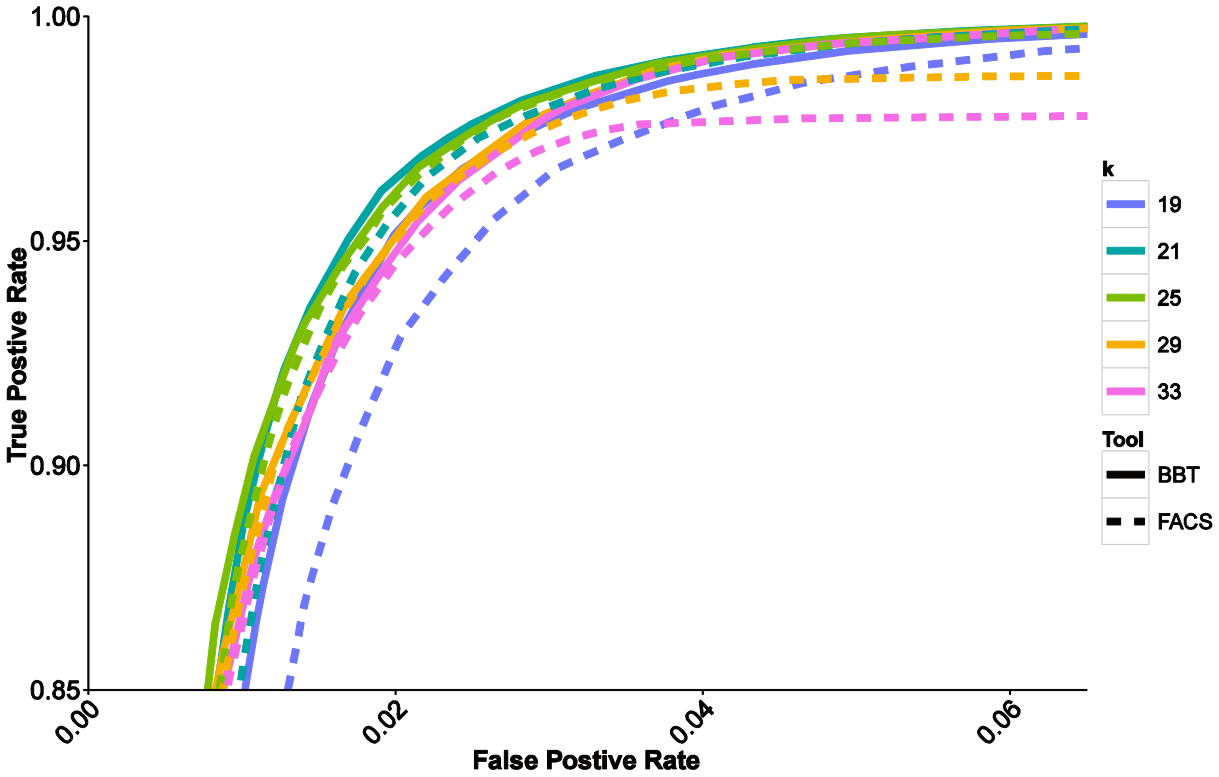
$$h = \frac{n}{m}\ln 2$$

Since there is a relationship between the filter size per element, we can also derive the optimal number of hash functions given only the FPR. Simplification after substitution yields:

$$\frac{m}{n} = -\frac{ln\, f}{(ln 2)^2}$$

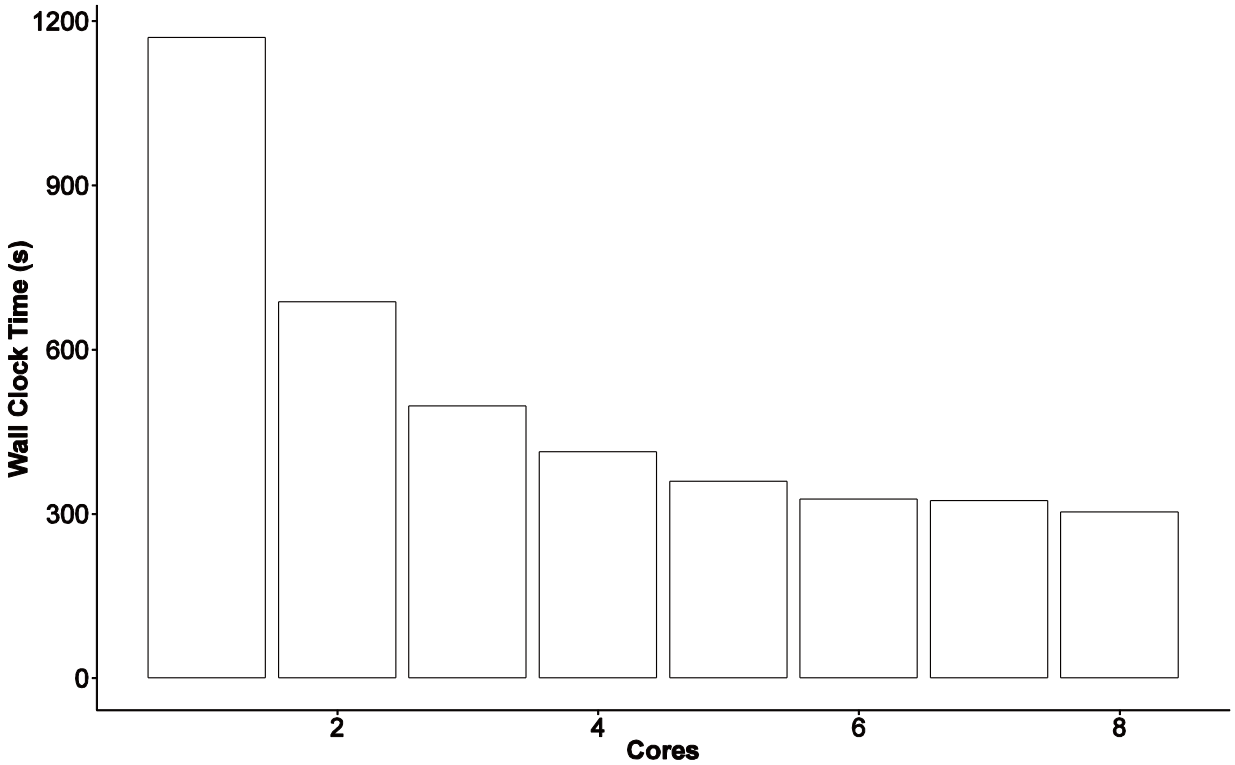
This means that increasing the FPR will make it necessary to increase the size of the filter with respect to the number of *k*-mers added to the filter. We should note that this calculation represents the minimum FPR because we use an approximate value of *n* to determine *m.* This is due to redundant sequences in the sequences. As elements are added to the filter, we expect

redundant sequences to be added, making the true value of *n* in our implementation lower to what is used in our calculations.



A.2 Effects of k-mer size using simulated mouse reads

Figure A.1 A ROC curve between BBT and FACS using simulated *H. sapiens* and *M. musculus* 100bp single-end reads filtered against an H. sapiens Bloom filter at multiple k-mer lengths.



A.3 Effects of k-mer size using simulated e. coli reads

Figure A.2 A ROC curve for BBT and FACS using simulated *H. sapiens* and *E. coli* 100bp single-end reads filtered against an *H. sapiens* Bloom filter at multiple k-mer lengths.

A.4 **Scaling on multiple threads**

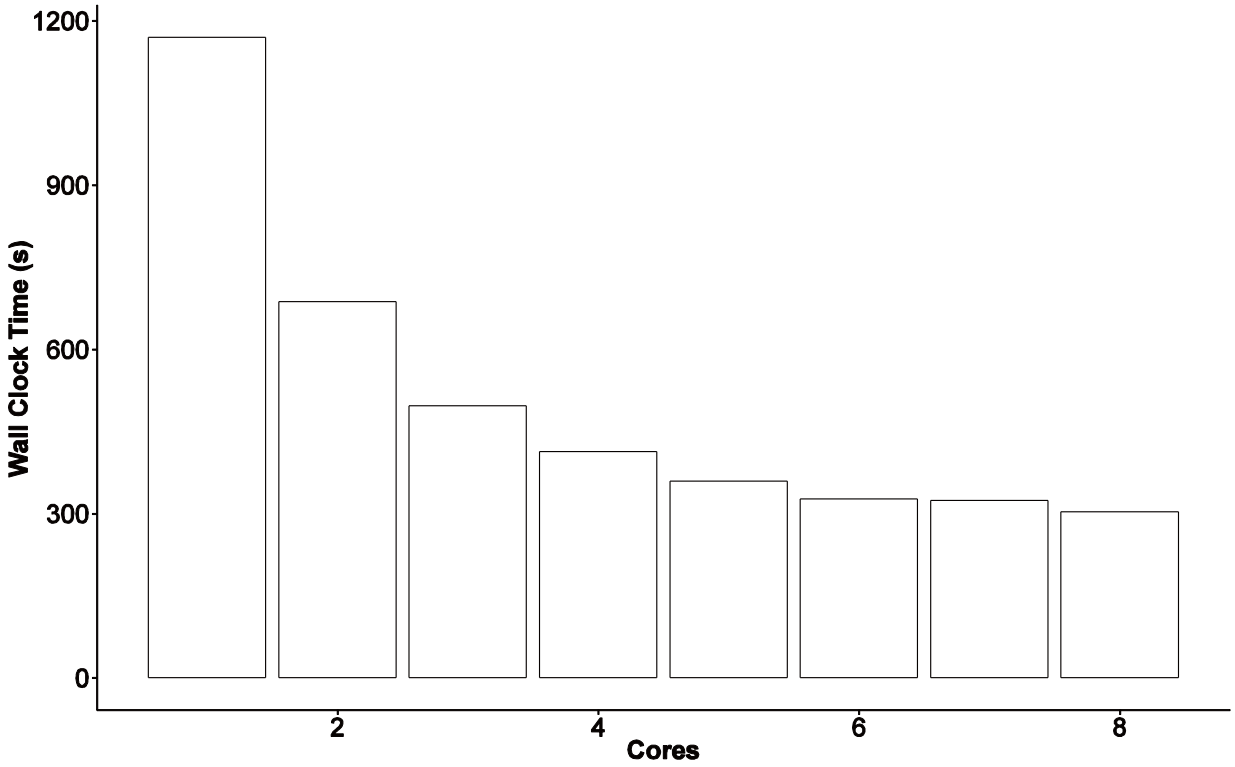


Figure A.3 The effect of the number of threads on BBT runtime using 50 Million reads from a set of real 2×150bp PE human DNA reads. These tests were run Red Hat Enterprise Linux WS release 4 server, Linux version 2.6.9-55.ELlargesmp, using a 16 x2.4GHz Intel Xeon processor with 62GB of memory. The effects of I/O dominate quickly, causing runtimes to plateau when at >6 cores. Thus, it is expected that if a faster disk were present better performance gains would be observed.

A.5 Scaling on datasets at different sizes

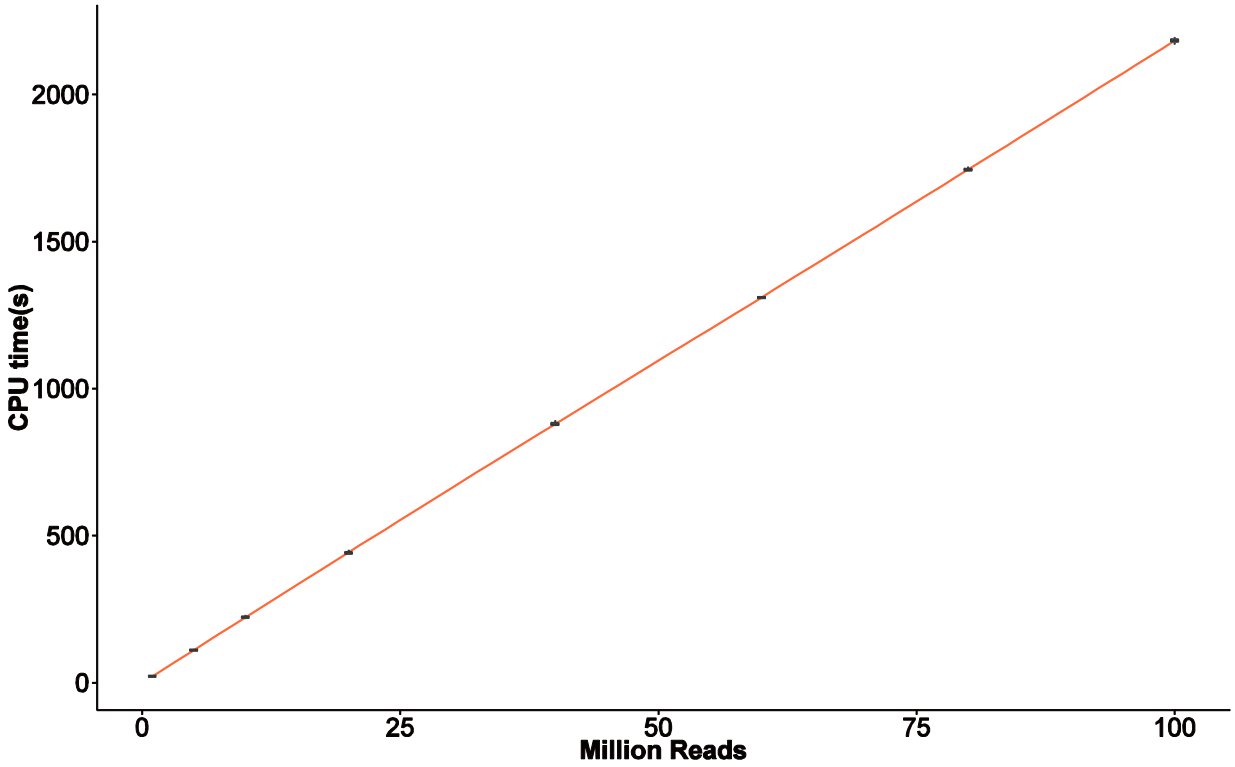


Figure A.4 Runtimes for BBT on different sized datasets. These tests were run Red Hat Enterprise Linux WS release 4 server, Linux version 2.6.9-55.ELlargesmp, using a 16 x2.4GHz Intel Xeon processor with 62GB of memory. The number of reads processed scales linearly with runtime.

Appendix B Chapter 3 Supplementary material

Algorithm B.1: Computing a progressive Bloom filter

Input: Parameters k and r, seed sequence q and reads pairs $P = \{(p_0,p'_0),...,(p_{|R|},p'_{|R|})\}$ with $|q| \geq k$

Output: Bloom filter of tagged k-mers from reads and seed sequence

Function TagKmer(q, P, k, r) **begin**

$F \leftarrow \emptyset$ //F is not a set, but a Bloom filter

for $i \leftarrow 0$ to $|q| - k + 1$ **do** //initial seed of the filter

$F \leftarrow F \cup$ kmer($q[i],...q[i+k]$) //add seeds to filter

for $i \leftarrow 0$ to $|R|$ **do** //iterate through all reads

$x \leftarrow 0,\ y \leftarrow 0$ //initialize k-mer overlap counts to 0

for $j \leftarrow 0$ to $|p_i| - k + 1$ **do** //check if first read k-mers present

if kmer($p_i[j],...p_i[j+k]$) $\in F$

$x \leftarrow$ x + 1 //increment if matches

for $j \leftarrow 0$ to $|p'_i| - k + 1$ **do** //check of second read k-mers present

if kmer($p'_i[j],...p'_i[j+k]$) $\in F$

$y \leftarrow y + 1$ //increment if matches

if $x > r$ **or** $y > r$ **do** //if k-mer counts reach threshold

for $j \leftarrow 0$ to $|p_i| - k + 1$ **do** //insert k-mers of first read

$F \leftarrow F \cup$ kmer($p_i[j],...p_i[j+k]$)

for $j \leftarrow 0$ to $|p'_i| - k + 1$ **do** //insert k-mers of second read

$F \leftarrow F \cup$ kmer($p'_i[j],...p'_i[j+k]$)

return F

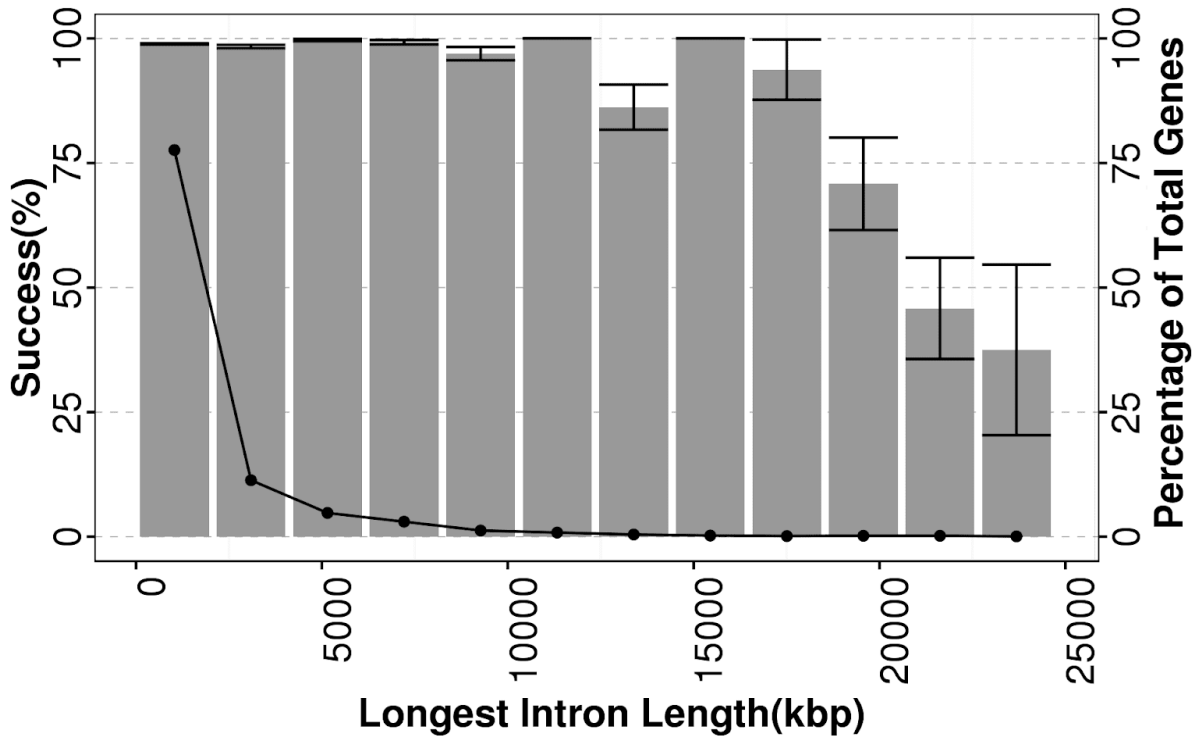


Figure B.1. The proportion of successful gene assemblies vs longest introns (bars), with percentage of total genes in each bin(lines). 86% of the genes are concentrated in the first two bins

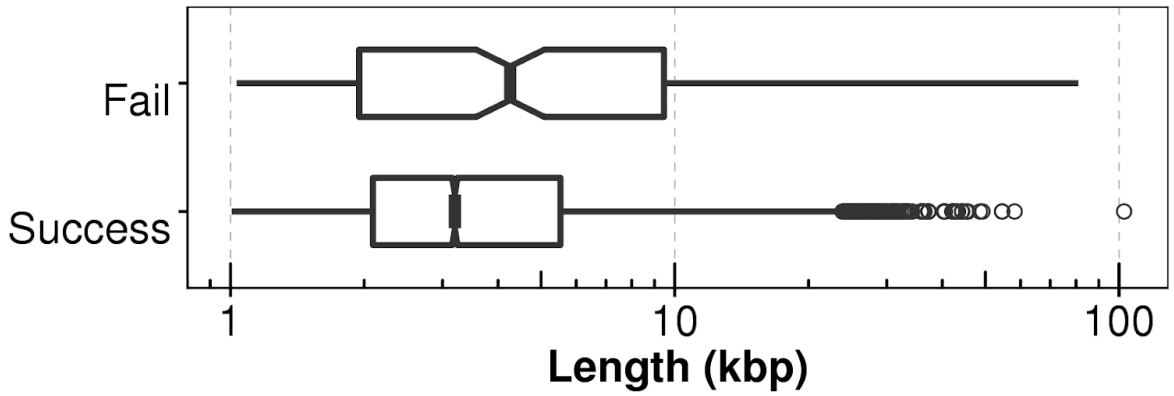


Figure B.2. Length comparison between *C.elegans* target genes that are successfully assembled by Kollector and those failed to assemble. Notches in the boxes represent a 95% confidence interval around the median. Length difference between two groups is found to be statistically significant by Student's t-test ($p=1.5x10^{-5}$)

Appendix C Chapter 4 Supplementary material

C.1 Read binning on COSMIC gene set by BBT versus BWA-MEM

Supplementary Note C.1. Tool Parameterization details of BWA MEM and BBT

BBT version 2.3.2 and BWA version 0.7.17-r118 were used

BWA index Construction:

```
BWA index refSeq.fa
BWA index ccg.fa
```

BWA MEM Mapping:

```
BWA mem -t 64 refSeq.fa file1.fq file2.fq | samtools view -bhS -
-o out.bam
BWA mem -t 64 ccg.fa file1.fq file2.fq | samtools view -bhS - -o
out.bam
```

BBT miBF construction:

```
biobloommimaker -F -b 0.8 -t 64 -p filter -S "
01111011100000011111101110000111100100000101001001011100011000010111101001100011
1010110100111101001110010111000111111111101001000011001100100100100011100001000
0001000011100010010001001100110000100101111111111000111010011100
```

```
1011110010110101
1100011001011110100001100011101001001010000010011110000111011111
1000000111011110" cancer_census_genes/*.fa
```

BBT Classification:

```
biobloommicategorizer -t 64 -f filter.bf --fq -p out -e file1.fq
file2.fq > out.fq
```

C.2 **Tool parameterization details of CLARK, CLARK-S & BBT**

BBT version 2.3.2 and CLARK version 1.2.5 were used

CLARK Database:

```
CLARK_PATH/set_targets.sh database_path bacteria viruses
```

CLARK:

```
CLARK_PATH/classify_metagenome.sh -n 64 -O out -R
CLARK_PATH/classify_metagenome.sh --spaced -n 64 -O out -R
```

BBT miBF Construction:

```
biobloommimaker -F -b 0.8 -t 64 -p filter -S
"1100011001110000011101001101101000100111010000011110010111011100000101101010011100011110
0111100010101101000001110111010011111000000
```

```
10111001000101101100101110000011100110001 1"
CLARK_EXTRACTED_REFERENCES/*.fa
```

BBT Default Classification:

```
biobloommicategorizer -t 64 -f filter -p out > out.tsv
biobloommicategorizer -b -c <*1 to 25> -r <*3 to 25> -t 64 -f filter -p
out > out.tsv
```

*Note: -c was set to -r unless -c < 3, in which case -r was set to 3 (default value). Total of 25 additional runs per dataset.

C.3 Impact of occupancy on filter saturation

As mentioned in the Methods section, the saturation of an element occurs when there is a loss of information due to collisions. That is, when a collision occurs at an intended location, either by random chance or due to legitimate repetitive sequence, we compensate by moving to another location to insert. If we cannot insert a value, we report all values in the frame as saturated. If information about a frame is lost, we can determine this through the saturated frames, and use the non-saturated frames to perform the classification in only a subset of the frames are saturated.

To investigate the expected saturation rates, we generated filters from a single randomly ~4Mbp sequences broken into 1Kbp sections on the same set of multiple spaced seeds (from one to seven seeds). We found that frame saturations that occur by random chance are impacted by the occupancy of the miBF, the number of seeds used and to a lesser extent the number of distinct references being indexed (Figure C.4). Given our tests to keep saturation low, it is advisable to use at least three seeds per frame (causing < 4% saturation at 50% occupancy in this example).

We note the number of indexes used does increase the rate of saturation, but the trends showed here hold when the number of indexes is sufficiently large (in these tests we use > 4000 indexes).

C.4 **Impact of the number of hash functions on index bias**

Our insertion strategy mitigates index bias via reservoir sampling. However, even with our strategies, our miBF construction may still be biased so it is important to evaluate the extent of this bias to ensure good real-world performance. In our tests, bias does not occur to a large extent, though in extreme cases some indexes may be under-represented, which should be minimized by increasing the number of spaced seeds used or elements indexed overall.

C.5 **Relationship between index frequency, false positive rate and bits per element**

Assuming equal proportions of each index, the optimal number of indexes to use is the maximum number relative to the bit representation. Accordingly, to minimize the memory usage of the data structures it makes sense to bit-pack elements in the data vector; however in threaded implementations, this can be quite difficult as atomic access for compare-and-swap operations is difficult to implement on unaligned memory resulting for irregular-sized (e.g. 7-bit integers) packed elements in the vector. We note that changes in regular-sized elements can be performed fairly easily and dynamically (e.g. using 8-bit IDs instead of 16bit IDs).

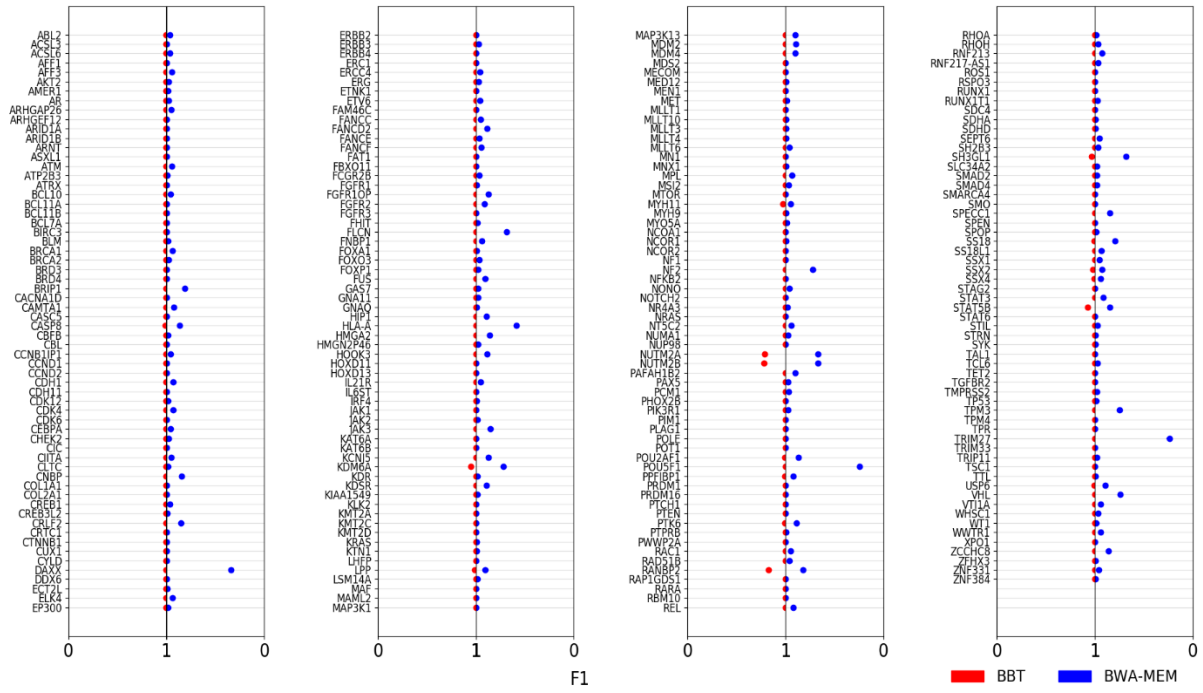


Figure C.1 Per-gene comparison of classification performance by BBT vs BWA-MEM indexing only sequences from the set of 580 cancer census genes. F1 scores for both methods using the same simulation dataset described in Figure 5 is calculated for each gene and plotted on the same horizontal line (red=BBT, blue=BWA-MEM). Only the subset of genes with F1 scores less than one in either method is shown here. The scale on the X-axis for BWA-MEM on the right is reversed for easy visual comparison such that higher scores for both methods localize to the middle while lower scores are off-centre.

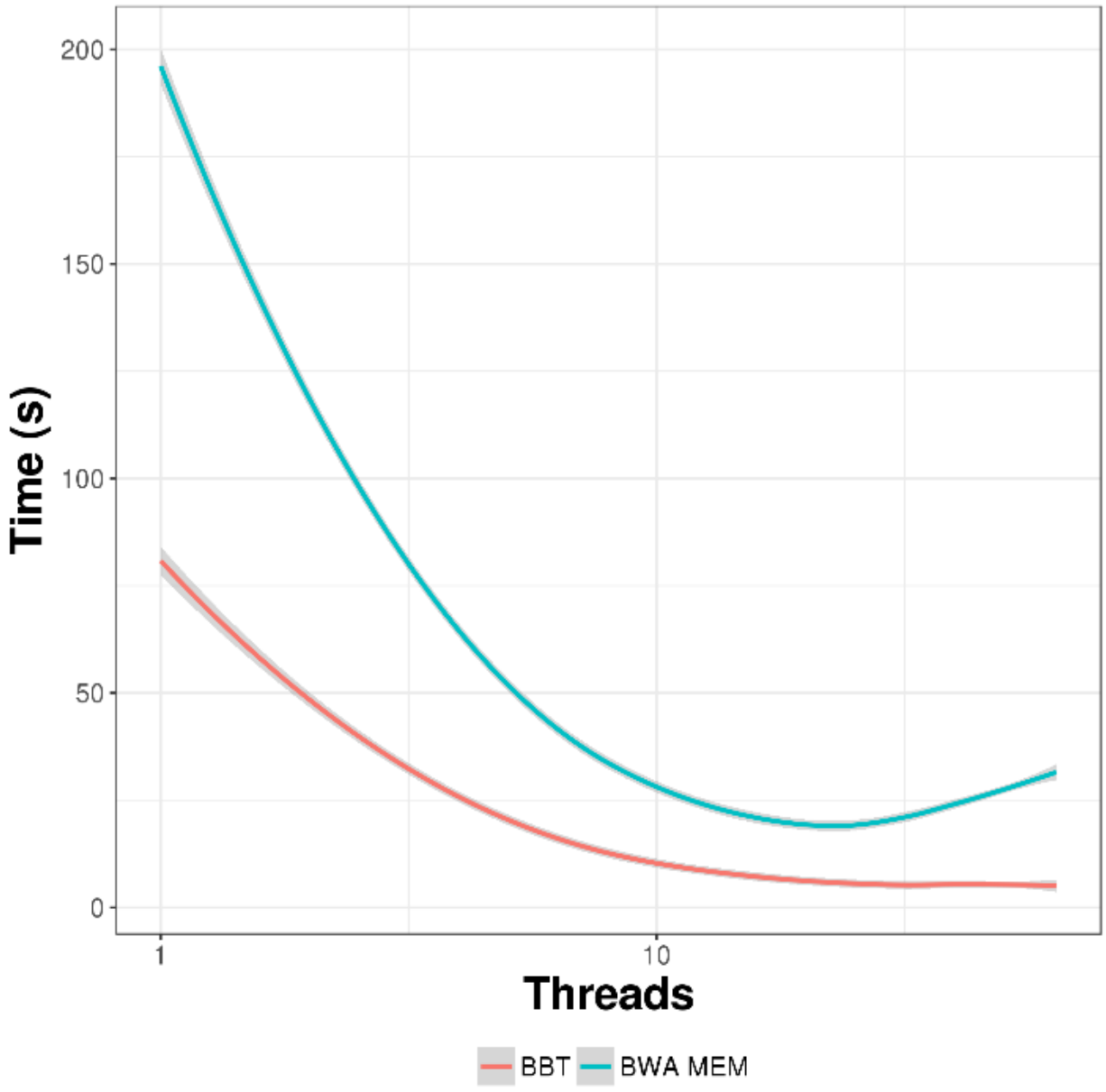


Figure C.2 Runtime performance of BWA MEM and BBT from threads 1 to 64 on a simulated dataset. Negative scaling occurs in BWA MEM at higher thread counts. Classification is measured only, index loading and file transformations (e.g. SAM to BAM) do not contribute to the runtime.

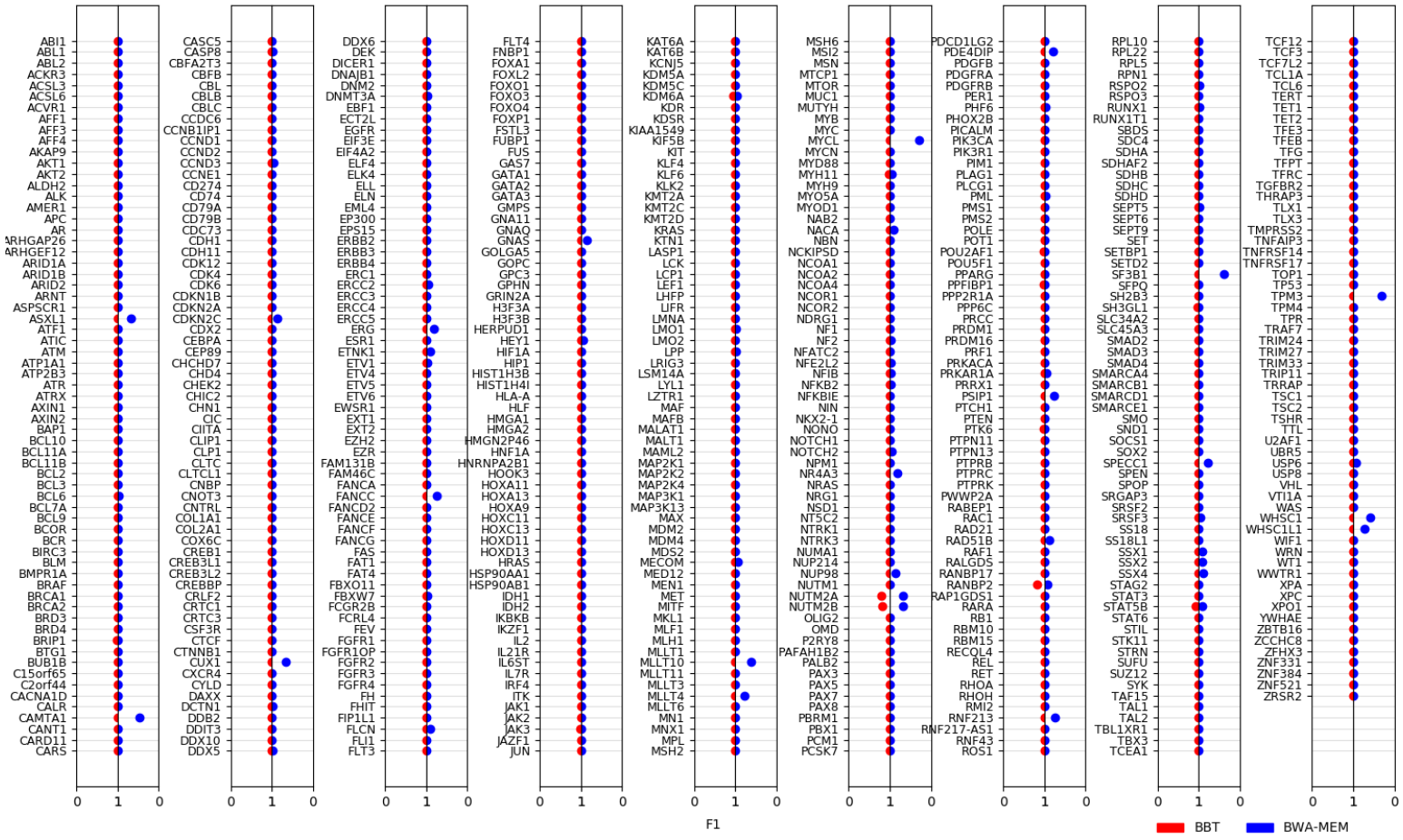


Figure C.3 Per-gene comparison of classification performance by BBT vs BWA-MEM. F1 scores for both methods using the same simulation dataset described in Figure 4.5 is calculated for each gene and plotted on the same horizontal line (red=BBT, blue=BWA-MEM). The scale on the X-axis for BWA-MEM on the right is reversed for easy visual comparison such that higher scores for both methods localize to the middle while lower scores are off-centre.

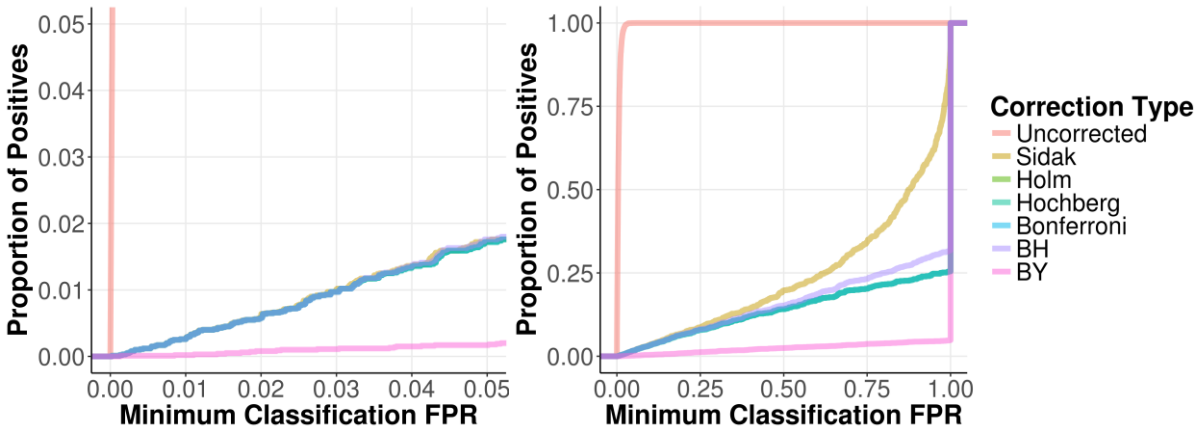


Figure C.4 Different methods of multiple testing correction on 10,000 150bp randomly simulated reads to a miBF generated on 580 genes from the COSMIC database. The proportion of positives refers to the number of reads that falsely classify to filter at various at a minimum classification FPR. Multiple test correction is similar in all methods when minimum classification FPR P-value is small (<5%). All testing corrections are overly conservative because testing correction assumes a uniform distribution of values, whereas the data follows a pseudo-uniform distribution because P-values are generated from a discrete binomial model (See the Methods section).

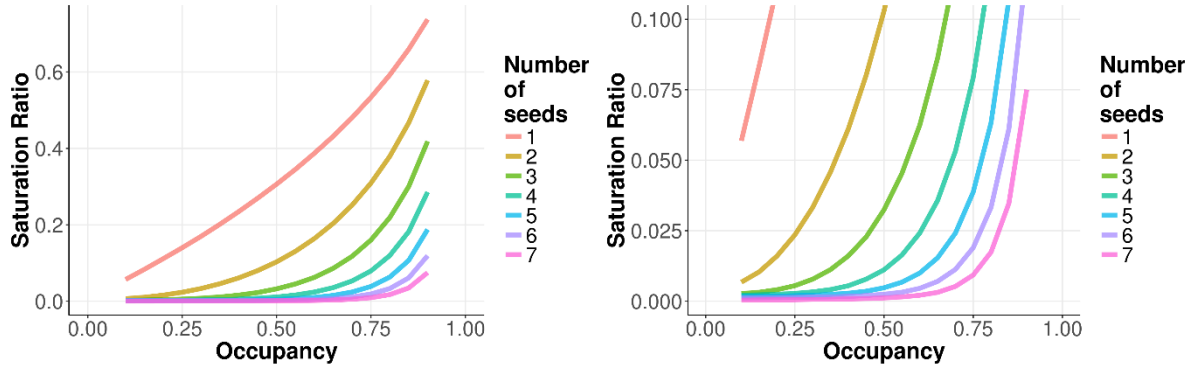


Figure C.5 Effect of occupancy and number of seeds per frame on index saturation rate. Because a random sequence was used, the contribution to saturation occurs primarily by random chance and not due to repetitive sequences.

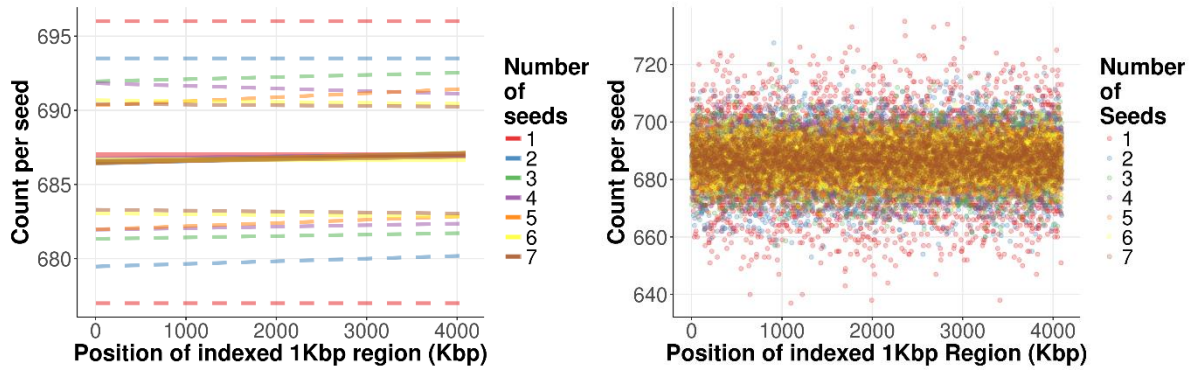


Figure C.6 Index bias of a miBF for different numbers of seeds per frame on a random sequence. Ideally, the count per seed for each index should be equal across all positions to be completely unbiased. On the left, the solid line is the median count and the dotted regions show the standard error interval of the index counts observed. Note that an occupancy rate of 0.5 was used for each miBF tested.

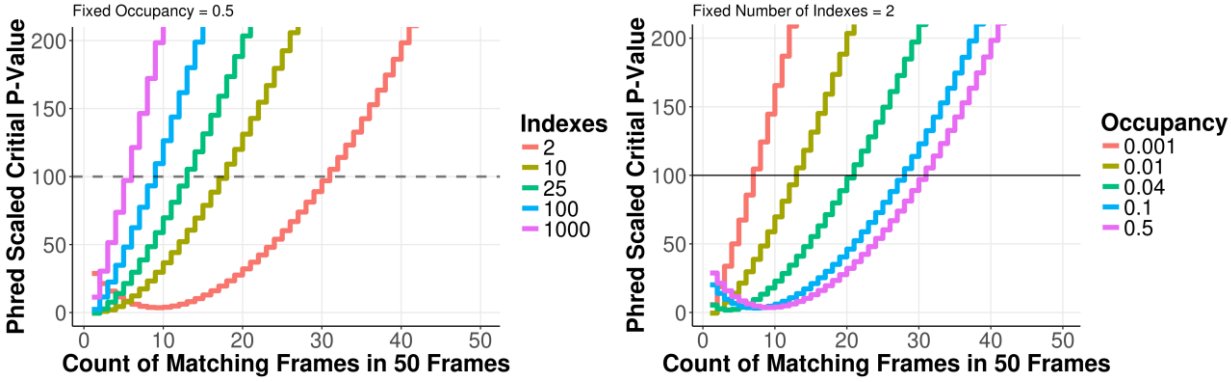


Figure C.7 Effects of index multiplicity (assuming equal index size) and Bloom filter occupancy on the number of matching frames required for a significant match. The right assumes a fixed occupancy and varies the number of indexes. The left assumes a fixed number of indexes and varies the occupancy. Both plots assume an FPR threshold of 10^{-10} (middle line, the default value in our implementation).

Dataset	Method	Best Hit			Multi-Hits Included		
		F1	Precision	Sensitivity	F1	Precision	Sensitivity
Buc12	BBT	**93.39%**	95.02%	**91.78%**	**93.02%**	**93.50%**	**92.55%**
	CLARK	91.77%	96.64%	87.14%	-	-	-
	CLARK-S	93.28%	**97.13%**	89.58%	76.54%	64.86%	90.33%
CParMed48	BBT	**95.64%**	97.18%	**94.11%**	**94.95%**	**94.79%**	**95.12%**
	CLARK	95.01%	**98.47%**	91.68%	-	-	-
	CLARK-S	95.11%	98.23%	92.08%	79.84%	69.02%	92.37%
Gut20	BBT	**97.26%**	98.63%	**95.91%**	**97.01%**	**97.50%**	**96.51%**
	CLARK	95.07%	**99.39%**	90.95%	-	-	-
	CLARK-S	96.23%	99.25%	93.30%	78.21%	65.45%	93.45%
Hou21	BBT	**93.61%**	97.91%	**89.49%**	**93.24%**	**96.75%**	**89.85%**
	CLARK	90.82%	98.61%	83.65%	-	-	-
	CLARK-S	92.88%	**98.68%**	87.43%	76.08%	66.06%	87.62%
Hou31	BBT	**93.64%**	96.14%	**91.21%**	**92.86%**	**93.39%**	**92.34%**
	CLARK	92.29%	**98.06%**	86.86%	-	-	-

	CLARK-S	92.89%	97.28%	88.70%	76.25%	65.27%	89.07%
simBA-525	BBT	93.85%	97.03%	**90.77%**	**92.96%**	**94.42%**	**91.52%**
	CLARK	**94.08%**	**98.75%**	89.63%	-	-	-
	CLARK-S	93.86%	98.39%	89.54%	80.18%	71.68%	89.69%
Soi50	BBT	**95.47%**	97.72%	**93.27%**	**94.81%**	**95.42%**	**94.20%**
	CLARK	94.99%	**98.92%**	91.21%	-	-	-
	CLARK-S	94.93%	98.49%	91.51%	79.64%	69.19%	91.68%

Table C.1 Classification F1, precision and sensitivity on unambiguous datasets from the CLARK-S paper.

Highest values for F1, Precision and Sensitivity of each dataset are in bold.

Dataset	Method	Best Hit			Multi-Hits Included		
		F1	Precision	Sensitivity	F1	Precision	Sensitivity
Buc12	BBT	**90.48%**	91.84%	**89.14%**	**89.40%**	**88.04%**	**90.77%**
	CLARK	87.57%	**95.26%**	80.51%	-	-	-
	CLARK-S	88.53%	94.43%	83.00%	72.50%	62.52%	84.08%
CParMed48	BBT	**93.68%**	95.08%	**92.29%**	**92.32%**	**90.02%**	**94.68%**
	CLARK	92.33%	**97.97%**	87.02%	-	-	-
	CLARK-S	92.93%	97.73%	88.36%	77.33%	67.48%	88.60%
Gut20	BBT	**94.26%**	94.54%	**93.99%**	**93.18%**	**88.32%**	**98.30%**
	CLARK	90.94%	**99.84%**	82.84%	-	-	-
	CLARK-S	93.57%	98.52%	88.86%	74.99%	63.04%	89.21%
Hou21	BBT	**91.88%**	95.51%	**88.38%**	**90.59%**	**91.74%**	**89.46%**
	CLARK	87.57%	**97.67%**	78.51%	-	-	-
	CLARK-S	89.69%	97.25%	82.71%	73.29%	64.69%	83.04%
Hou31	BBT	**92.05%**	94.38%	**89.78%**	**90.41%**	**88.51%**	**92.35%**
	CLARK	89.78%	**98.01%**	82.24%	-	-	-

	CLARK-S	90.16%	96.26%	84.45%	73.54%	63.62%	85.01%
simBA-525	BBT	**73.09%**	80.25%	**66.57%**	**70.71%**	**71.34%**	**70.09%**
	CLARK	72.61%	**89.52%**	58.90%	-	-	-
	CLARK-S	71.74%	85.72%	60.04%	60.24%	59.82%	60.66%
Soi50	BBT	**94.22%**	95.88%	**92.58%**	**93.00%**	**91.21%**	**94.81%**
	CLARK	93.15%	**98.70%**	87.91%	-	-	-
	CLARK-S	93.55%	98.18%	89.15%	77.62%	67.42%	89.36%

Table C.2 Classification F1, precision and sensitivity on default datasets from the CLARK-S paper. Highest values for F1, Precision and Sensitivity of each dataset are in bold.

Appendix D Chapter 5 Supplementary material

D.1 Parameters used for ROC-like curves [Highest F1 settings in brackets]:

The **bolded values** are the default parameters used for that particular parameter. We have included the results of our parameters sweeps in supplementary TSV files. Our comments on parameter effects are based on our observations on the software versions tests and the datasets used in this study. Our parameters sweep values were inferred based on our understanding of the tools, their documentation, as well as communication with the authors.

We did not mention parameters that change thread numbers here, but we did test each tool with multiple threads. DALIGNER was notable in that its thread number is set by altering header files and must be a power of 2.

BLASR (git SHA fa06086ad6c2aaabda6bbd005e3fb37df8bacc98)

Options for anchoring alignment regions:

-minMatch 11, **12**, 13 [PB: 11, ONT: 11]

- Minimum seed length. Higher minMatch will speed up alignment but decrease sensitivity.

-maxMatch 15, 16, 17, **Inf** [PB: 17, ONT: 17]

- Stop mapping a read to the genome when the longest common prefix reaches this threshold. This is useful when the query is part of the reference, for example when constructing pairwise alignments for de novo assembly

-nCandidates coverage × 1, coverage × 5, coverage × 10 [PB: coverage × 5,ONT: coverage × 5]

- Keep up to 'n' candidates for the best alignment

-fastMaxInterval on, **off** [PB: off, ONT: off]

- Fast search maximum increasing intervals as alignment candidates

-fastSDP on, **off** [PB: on, ONT: on]

- Use a fast heuristic algorithm to speed up sparse dynamic programming

Options for alignment output:

-bestn coverage × 1, coverage × 5, coverage × 10 [PB: coverage × 5, ONT: coverage × 5]

- Report the top 'n' alignments

-m 4

- Alignment output format

Notable BLASR options unchanged:

-minPctIdentity

- Threshold by percent identity

-maxAnchorsPerPosition

- Do not add anchors from a position if it matches to more than 'm' locations in the target.

-aggressiveIntervalCut

- Aggressively filter out non-promising alignment candidates, if there exists at least one promising candidate.

BLASR is a read alignment tool (usually to a reference genome) and had many options that were not very relevant for overlap detection. We avoid parameters that would not affect initial overlap candidate finding such as *-minPctIdentity* and *-maxAnchorsPerPosition* which affect the algorithm during or post local alignment. We also refrained from using options that would interfere with the need for all pairwise overlaps such as *-aggressiveIntervalCut*.

Of the parameters tested we found that -nCandidates/-bestn to be the most important to maximizing specificity and sensitivity. Large values of *-nCandidates* slow down mapping. This value must be carefully parameterized, too low and sensitivity is reduced, to high and specificity and is affected. -maxMatch also affected specificity and sensitivity and was suggested to be used when performing read-to-read overlaps. Higher values of *-maxMatch* increase specificity and low values increase sensitivity. Finally, we also investigated if efficiency-related parameters affect overlap specificity and sensitivity such as *-fastMaxInterval* and *-fastSDP*. We found that *-fastSDP* did not reduce overlap specificity and sensitivity so it may be used when overlapping reads to help increase computational performance.

DALIGNER (git SHA 29234506d87e2d3e207034340777550e81cf55a6)

-m dust

- Use DUST soft masking tracks to filter out *k*-mers

-l 200,600,**1000** [PB: 200, ONT: 200]

- Minimum base pairs when searching for local alignments

-e **0.7**, 0.8 [PB: 0.8, ONT: 0.7]

- Average correlation rate in local alignment search

-w 5, **6**, 7 [PB: 5, ONT: 5]

- The bandwidth of 2^w that contains a collection of exact matching *k*-mers

-k 12, **14**, 16 [PB: 14, ONT: 16]

- *k*-mer length

-h 30, **35**, 40 [PB: 40, ONT: 40]

- Total number of bases covered by *k*-mers in a potential hit

-t 5, 10, 15, **no filtering** [PB: 10, ONT: 10]

- Suppresses the use of any *k*-mer that occurs more than *t* times in a batch

Notable DALIGNER options unchanged:

-M

- Using a maximum memory (in GB) to suppresses the use of *k*-mers. -*t* will be selected based on a memory limit rather than an explicit multiplicity threshold.

To reduce misalignments in repetitive regions, and to increase the alignment speed, one or more interval tracks can be provided with the -m option (in this case a DUST track). Any *k*-mers that comprise bases from the masked intervals are ignored for the purposes of seeding a match. The options -*l* and -*e* influence the speed and accuracy of local alignment, while the options -*k*, -*h*, and -*w* control the initial filtration search for possible matches between reads. Option -*t* is tuned for correcting the over-represented *k*-mers (especially homopolymers).

Of the parameters tested, we found that -*l* and -*t* to be the most important options. For -*t* (*k*-mer filtering per batch) it seems, for the most part, any filtering is better than none, however, it seems that it is possible to over and under suppress *k*-mers depending on the dataset. For -l, we had expected lower values to produce a large number of false positives, however in practice, at low values it only marginally increased false positives and while increasing sensitivity. *k* also seems to be important but seems to depend on the overall error rate (lower values of *k* are suggested for more error-prone sequences).

MHAP (v1.6)

-k 12, 14, **16** [PB: 14, ONT: 14]

- *k*-mer size used for MinHashing

--num-hashes **512**, 2048, 4844 [PB: 4844, ONT: 4844]

- Number of min-mers to be used in MinHashing

--num-min-matches 2, **3**, 4 [PB: 3, ONT: 3]

- Minimum number of min-mer that must be shared before computing second stage filter

--max-shift **0.2**, 0.3, 0.5 [PB: 0.2, ONT: 0.2]

- region size to the left and right of the estimated overlap, where *k*-mer matches are still considered valid.

--threshold **0.04**, 0.02, 0.005 [PB: 0.02, ONT: 0.04]

- The threshold similarity score cutoff for the second stage sort-merge filter

-f file

- *k*-mer counts file for repetitive *k*-mer filtering.

Of the parameters tested, we found that the --*num-hashes* and --*threshold* to have the largest effect on specificity and sensitivity. Increasing --*num-hashes*, for the most part, will increase both specificity and sensitivity, but unfortunately it also has a large detrimental effect on time and memory, so one must be careful when tuning this parameter. Decreasing the --*threshold* parameter understandably increases sensitivity and decreases specificity so it must be carefully tuned. -*k* may have some effect on sensitivity and specificity but this likely error-rate specific (higher error rates need smaller *k*). The other options did not seem to have much of an effect, at least in term of overlap detection (may have an effect on estimated region size).

GraphMap (v0.22)

-w owler [PB: owler, ONT: owler]

- Option to select how to report overlap results. Mode owler is fast and reports overlaps in MHAP format

Graphmap had options that would affect alignment stages but none of these options, other than the number of threads would do anything when -w owler was used.

Minimap (0.2-r124-dirty)

-k 13,14,**15**,16 [PB: 14, ONT: 16]

- *k*-mer size for indexing

-w 5,..,**10**,..,50 [PB: 5, ONT: 5] (default is k⅔)

- Minimizer window size (indexing stage)

Notable Minimap options unchanged:

-I

- The batch size of the index

We did not change any values related to the mapping itself as it was recommended by the authors to simply modulate -*w* to have the largest impact on sensitivity and specificity. We found this to be the case as well, generally the smaller the -*w* the better the sensitivity, with a minimal effect on the specificity. Increasing *k* seemed to help increase specificity but has a consequence of decreasing sensitivity, similar to other methods the degree of this is dependent on the error rate of the reads (large *k* can be tolerated if reads have lower error). The batch size -I of index

limits the maximum memory usage it will use. Though we did not change this value, we would expect it to allow for faster performance on large values albeit at the cost of increased memory.

D.2 Parameters for performance tests using default parameters

DALIGNER

-mdust

GraphMap

-w owler

BLASR

-bestn coverage × 1

-nCandidates coverage × 1

-m 4

D.3 Dataset preprocessing details and parameters used for read simulation

PB Datasets

We ran the smrtanalysis tool (v2.2.0.133377) pls2fasta (with option -trimByRegion) on the raw bax.h5 files outputting fastq files, which were then converted to fasta format. Reads shorter than 1000bp were filtered from each dataset.

ONT Datasets

We did not preprocess the datasets used in our ONT experiments as our source already provided 2D reads in fasta format.

PBsim (v1.0.3)

We used --sample-fastq on the PB P6-C4 *E. coli* dataset, which PBSim uses to generate the read length distribution of the simulated reads. We set the --depth option to match the raw coverage of our read datasets.

NanoSim (git SHA 5c02a2e3d71924adfc8055cd82b8f6b0ae8502a2)

The error profile and read length distributions were derived from the ONT SQK-MAP-006 *E. coli* dataset was used to generate both the simulated ONT *E. coli* dataset and the simulated ONT *C. elegans* dataset. We set -n (number of reads) to match our real *E. coli* dataset and 1000000 for our simulated ONT *C. elegans* dataset.

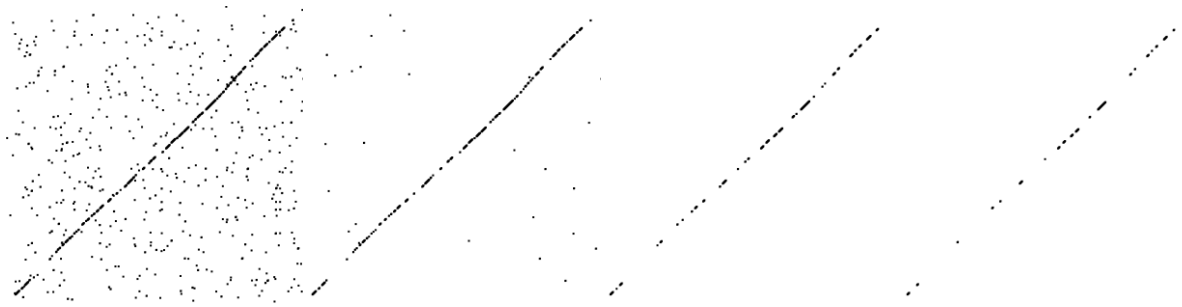


Figure D.1 Dot plots depicting, from top left to bottom right, a 1400bp overlap region between two 2D ONT reads at k=6, 8, 12, 16 from the ONT SQK-MAP-006 *E. coli* dataset.

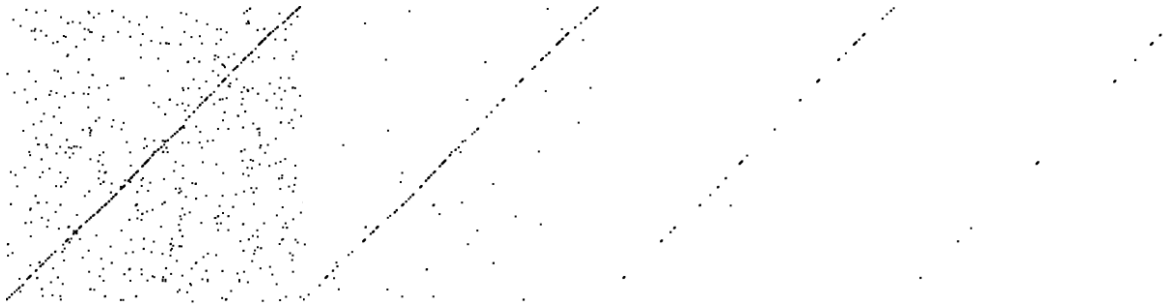


Figure D.2 Dot plots depicting, from top left to bottom right a 1400bp overlap region between two PB reads at k=6, 8, 12, 16 from the PB P6-C4 *E. coli* dataset

Trends in the assembly of error-prone long read data

While *de novo* assembly of short reads often utilizes de Bruijn graphs (DBG) (Pevzner, et al., 2001), due to its favourable time complexity (no overlap stage needed). DBGs work by breaking up sequence *k*-mers (substrings of length *k*) and traversing a *k*-mer graph to construct contigs (contiguous sequences). Unfortunately, when investigating the number of common short *k*-mers (<16bp) between two long reads, we observe a sharp decline with increasing values *k* (Supp. Figure D.11,12). Thus, though it may be theoretically possible to use the DBG paradigm to assemble long reads, given random error, it is apparent that in practice, it would be unfeasible due to the depth required to acquire enough *k*-mers. Because of this, OLC (requiring overlap detection) dominates the assembly of long reads.

Dataset	Source
PB P6-C4 E. coli	https://github.com/PacificBiosciences/DevNet
Simulated PB E. coli	Generated based on PB P6-C4 E. coli dataset and E. coli genome using PBSim
ONT SQK-MAP-006 E. coli	http://lab.loman.net/2015/09/24/first-sqk-map-006-experiment/
Simulated ONT E. coli	Generated based on ONT SQK-MAP-006 E. coli dataset and E. coli genome using NanoSim
PB P6-C4 C. elegans	https://github.com/PacificBiosciences/DevNet
Simulated ONT C. elegans	Generated based on ONT SQK-MAP-006 dataset and C. elegans genome using NanoSim

Table D.1 Benchmarking Datasets used in overlapper comparisons

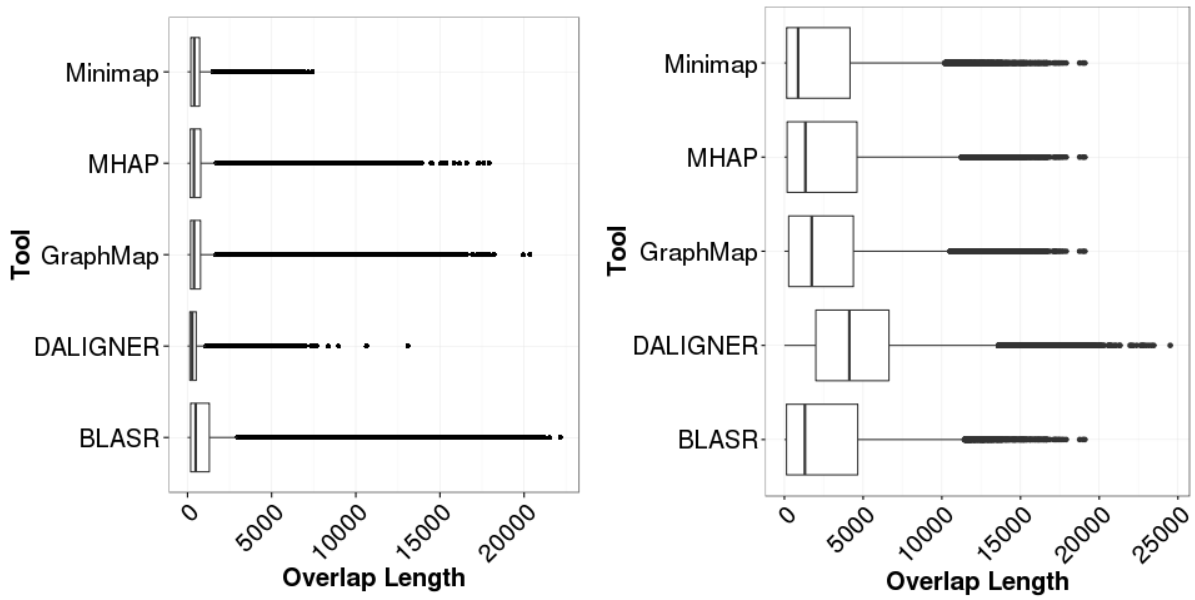


Figure D.3 Overlap length boxplots on the F1 score optimized overlap runs on the simulated PB *E. coli* (right) and simulated ONT *E. coli* (left) datasets. The length was based only on the correctly assigned overlaps according to the ground truth that the simulated reads were derived from. Overlap length used was based on reference ground truth and not the overlap length reported by the tool to prevent skewing due to systematic overestimation or underestimation of the overlap regions size.

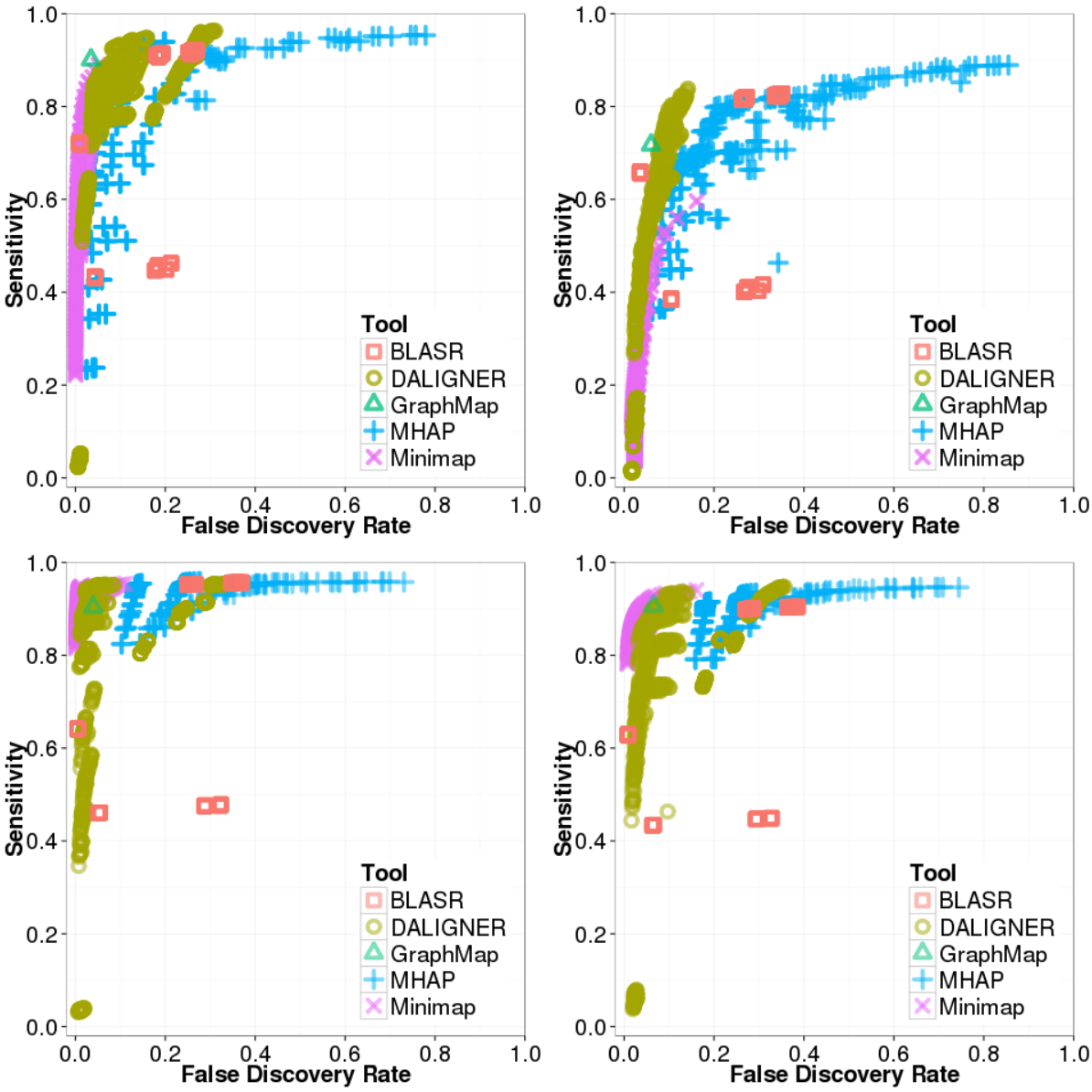
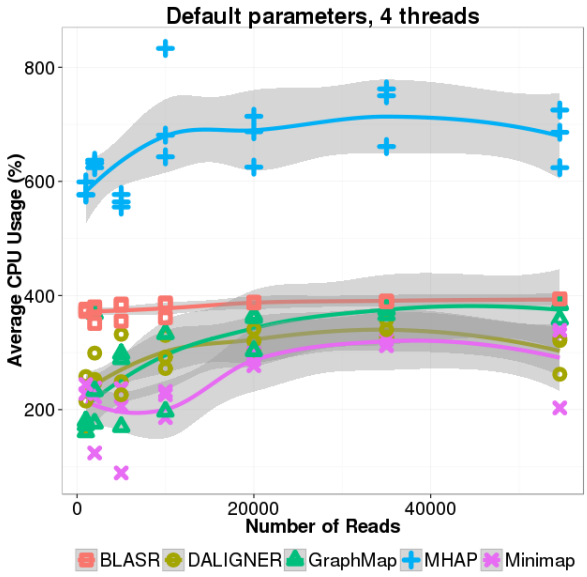
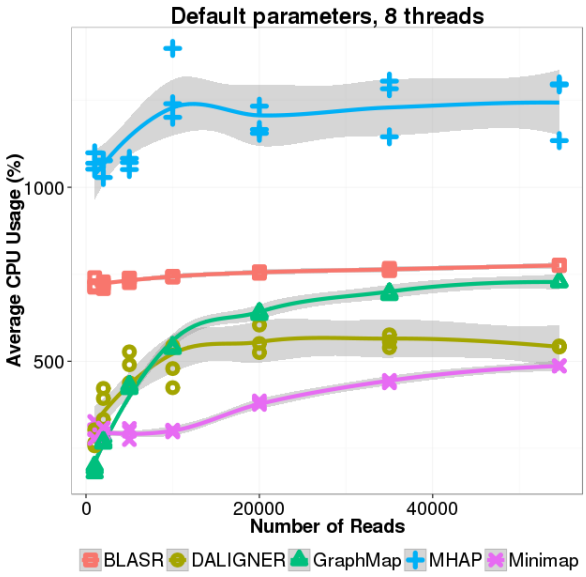
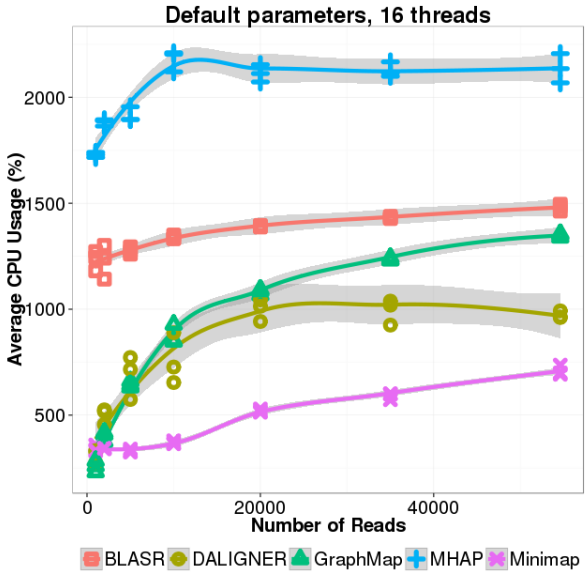
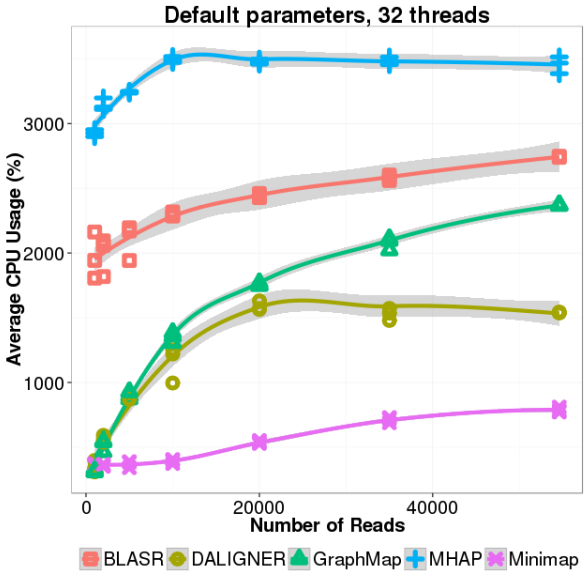


Figure D.4 ROC-like plot on BLASR, DALIGNER, GraphMap, MHAP, GraphMap, and MHAP. Top left: PB P6-C4 E. coli simulated with PBsim. Top right: PB P6-C4 E. coli dataset. Bottom left: ONT SQK-MAP-006 E. coli simulated with Nanosim. Bottom right: ONT SQK-MAP-006 E. coli dataset. We have only included successful runs in these plots, as some runs have failed (F1 score < 1%) due to parameters incompatible with a particular dataset.


Default parameters, 4 threads
Average CPU Usage (%)
800
600
400
200
0
20000
40000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap


Default parameters, 8 threads
Average CPU Usage (%)
1000
500
0
20000
40000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap


Default parameters, 16 threads
Average CPU Usage (%)
2000
1500
1000
500
0
20000
40000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap


Default parameters, 32 threads
Average CPU Usage (%)
3000
2000
1000
0
20000
40000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap

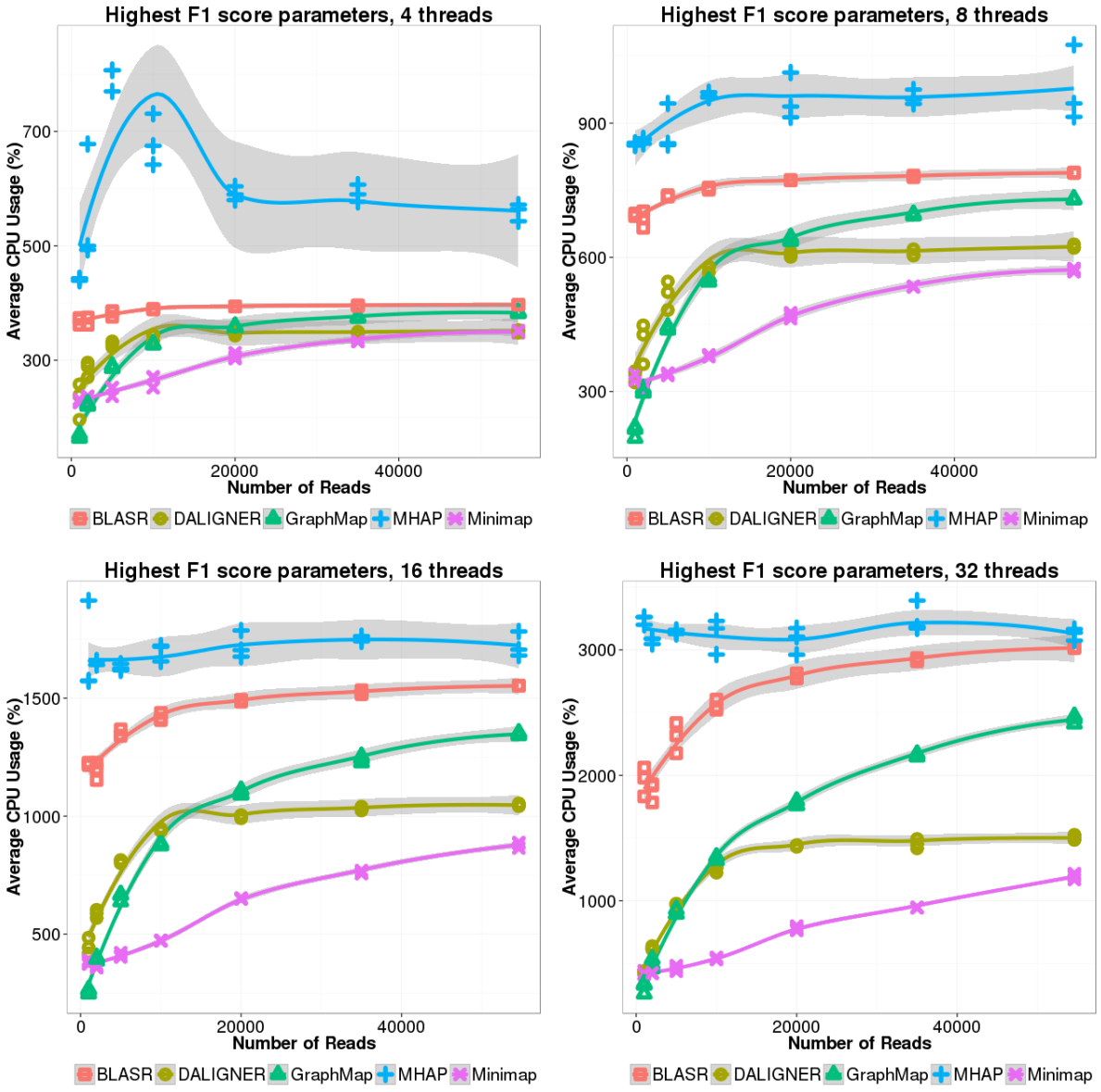
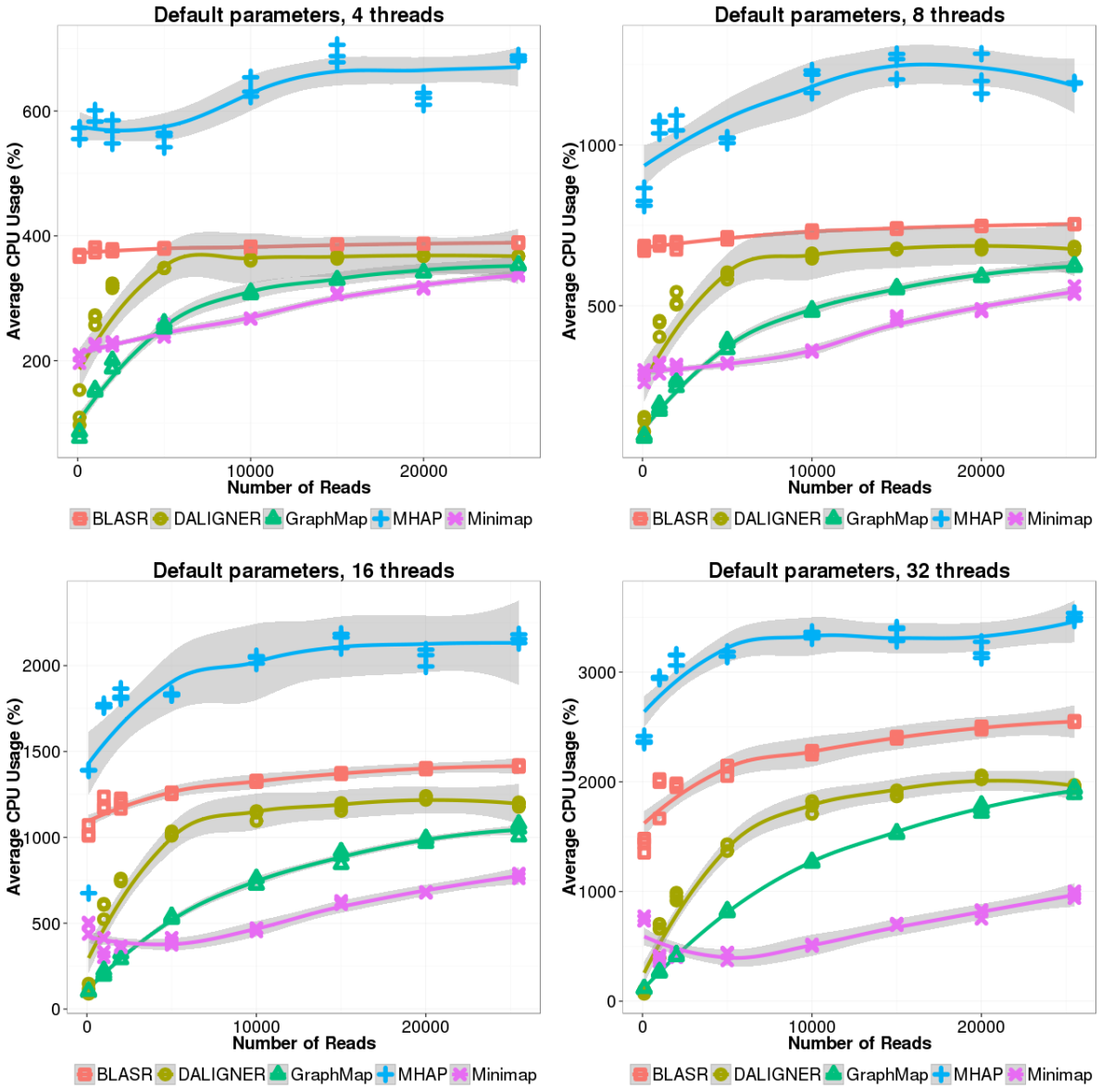


Figure D.5 Average CPU usage on the PB P6-C4 *E. coli* dataset at a differing number of randomly subsampled reads. Each tool was parameterized using default settings or settings yielding the highest F1 score.


Default parameters, 4 threads
Average CPU Usage (%)
600
400
200
0
10000
20000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Default parameters, 8 threads
Average CPU Usage (%)
1000
500
0
10000
20000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Default parameters, 16 threads
Average CPU Usage (%)
2000
1500
1000
500
0
10000
20000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Default parameters, 32 threads
Average CPU Usage (%)
3000
2000
1000
0
10000
20000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap

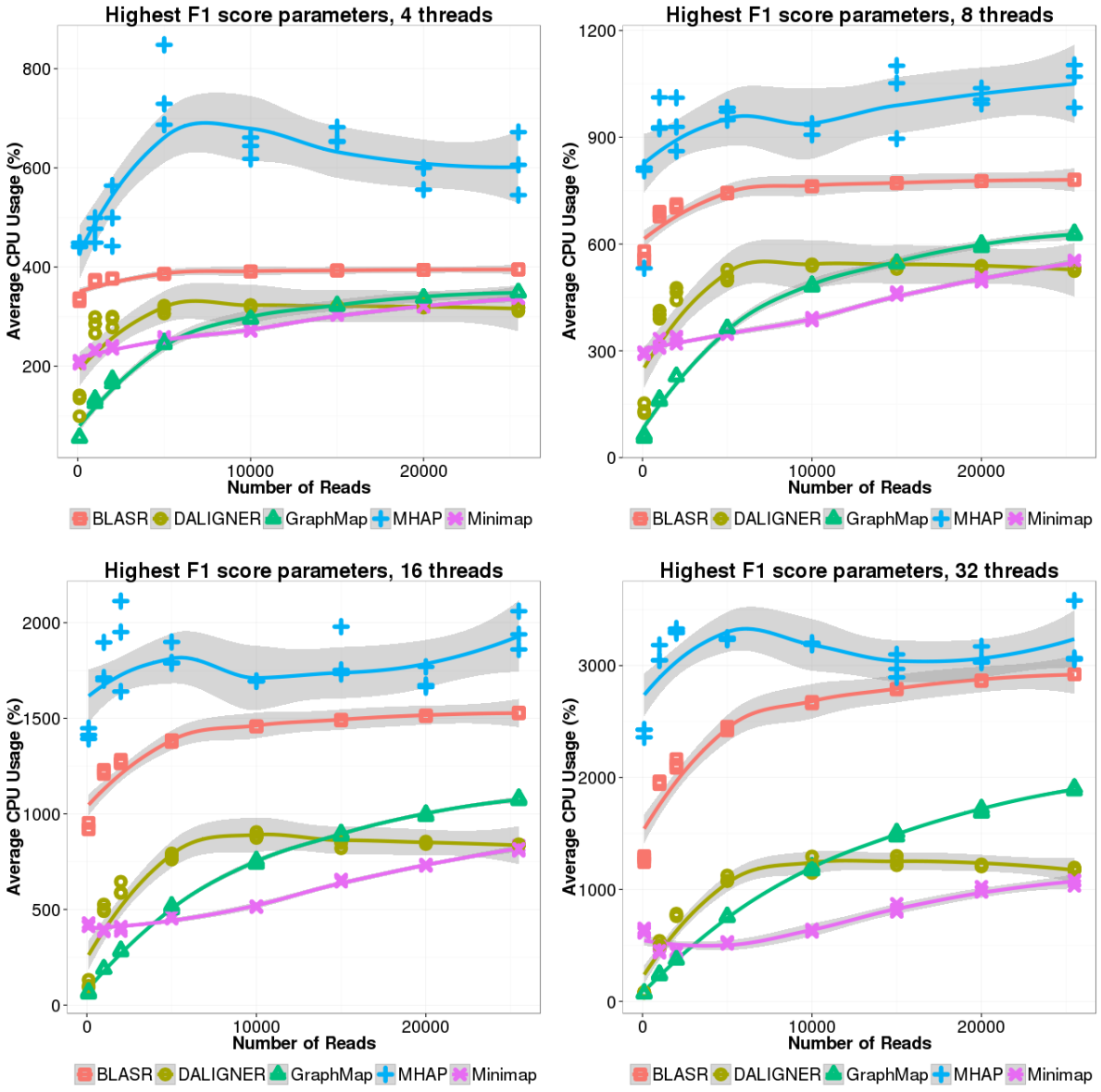
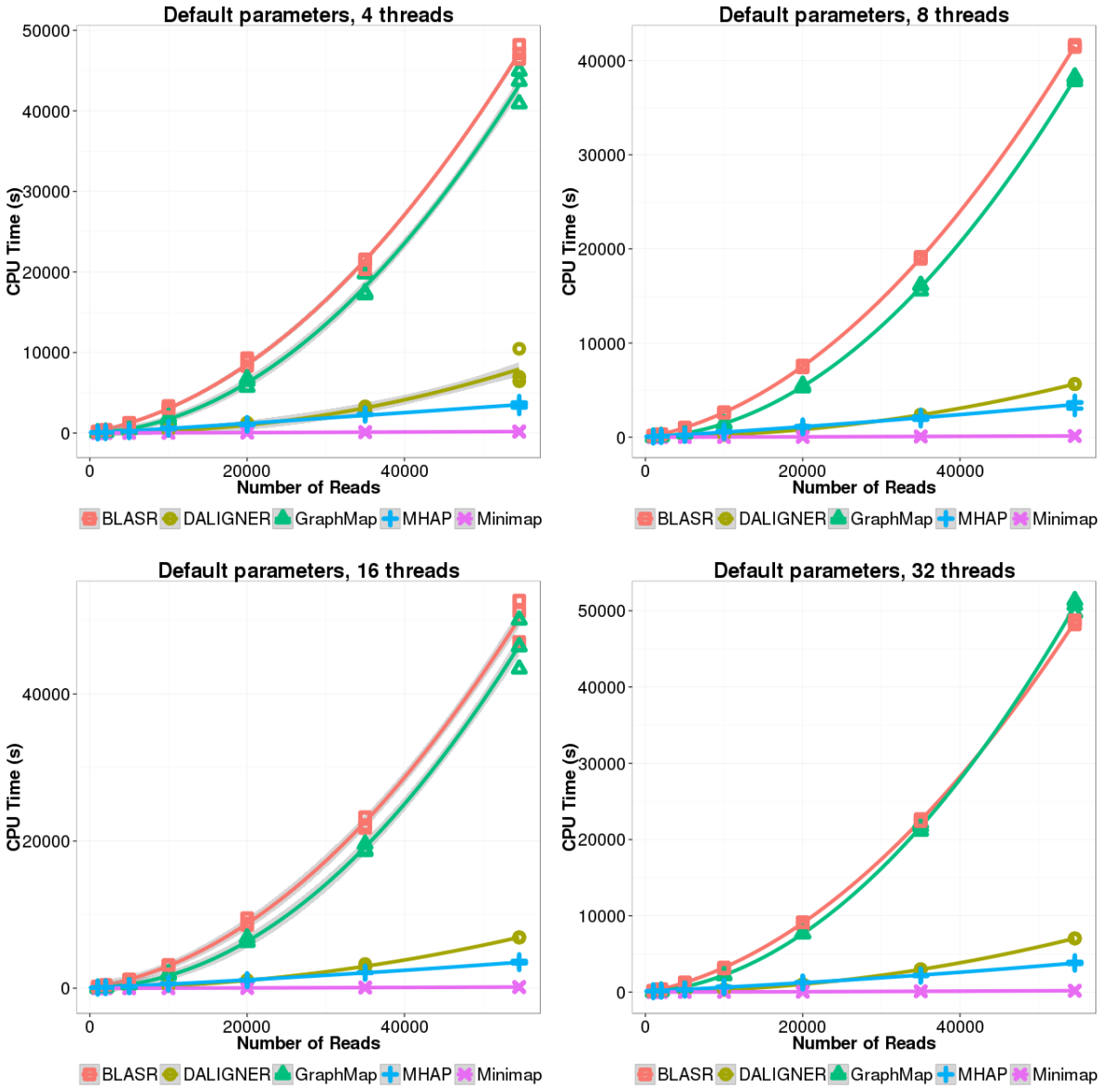


Figure D.6 Average CPU usage on the ONT SQK-MAP-006 *E. coli* dataset at a differing number of randomly subsampled reads. Each tool was parameterized using default settings or settings yielding the highest F1 score.


Default parameters, 4 threads
Default parameters, 8 threads
Default parameters, 16 threads
Default parameters, 32 threads
CPU Time (s)
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap

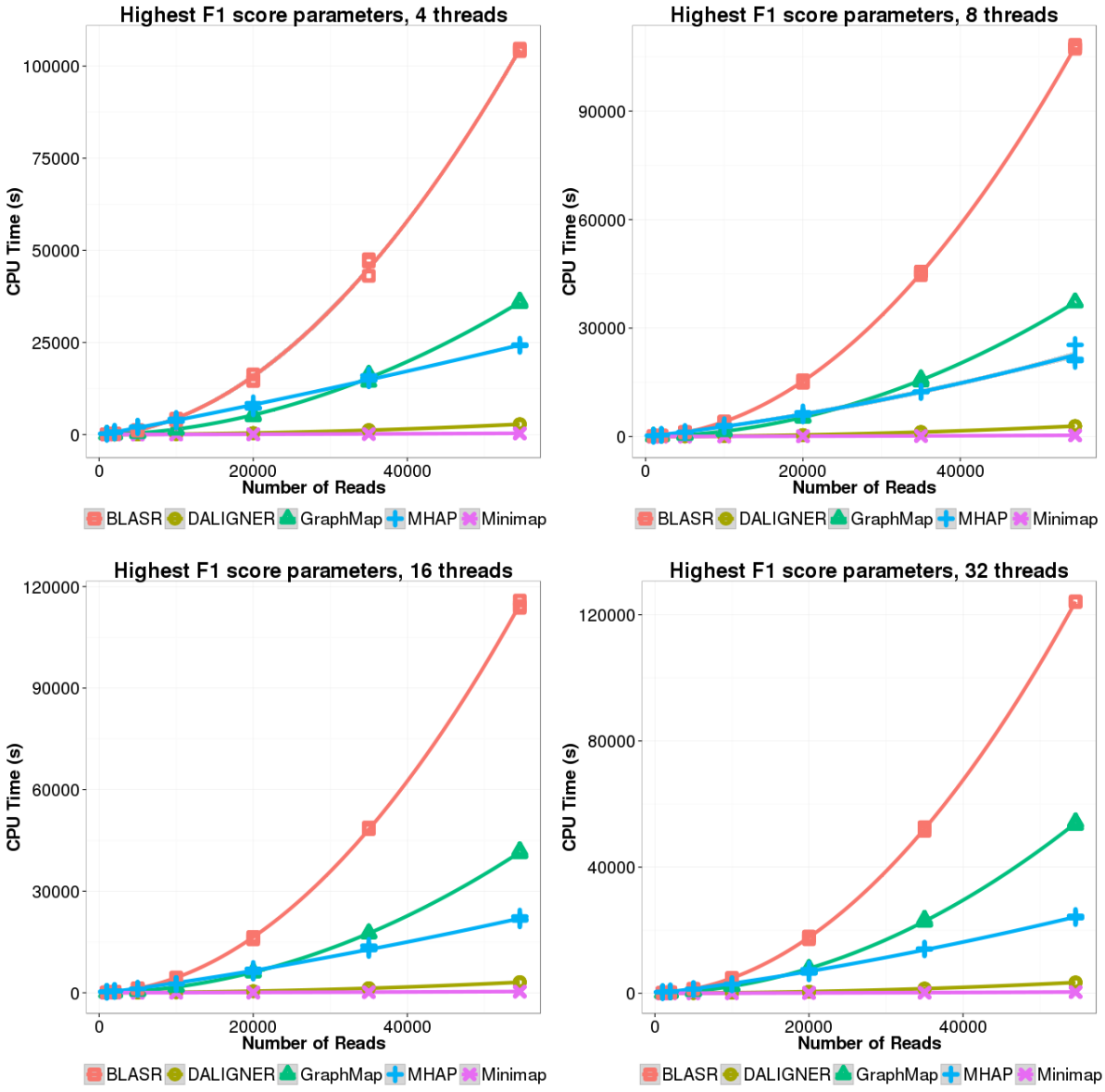
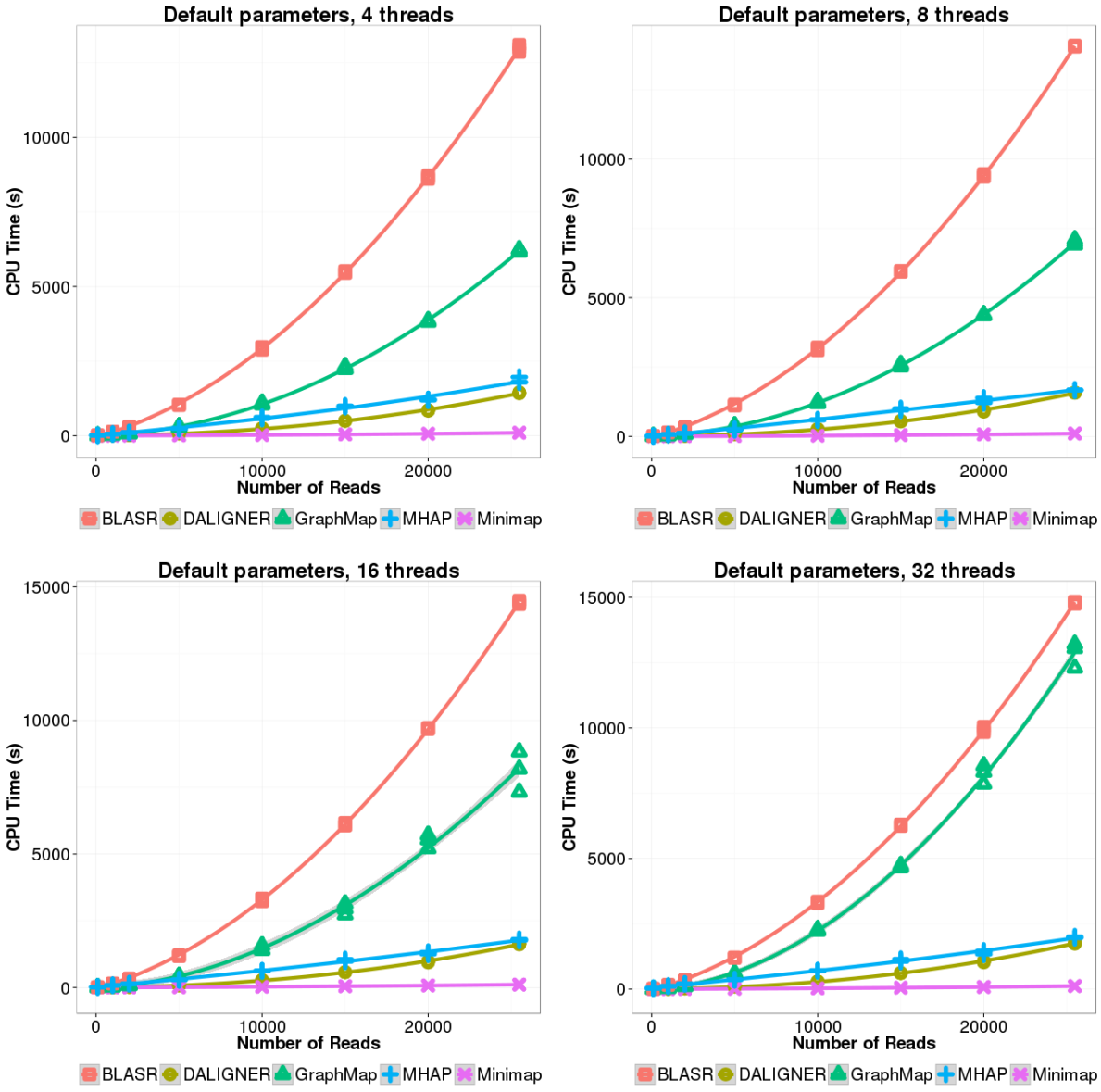


Figure D.7 CPU time on the PB P6-C4 *E. coli* dataset at a differing number of randomly subsampled reads. Each tool was parameterized using default settings or settings yielding the highest F1 score.


Default parameters, 4 threads
Default parameters, 8 threads
Default parameters, 16 threads
Default parameters, 32 threads
CPU Time (s)
Number of Reads
0
5000
10000
15000
20000
BLASR
DALIGNER
GraphMap
MHAP
Minimap

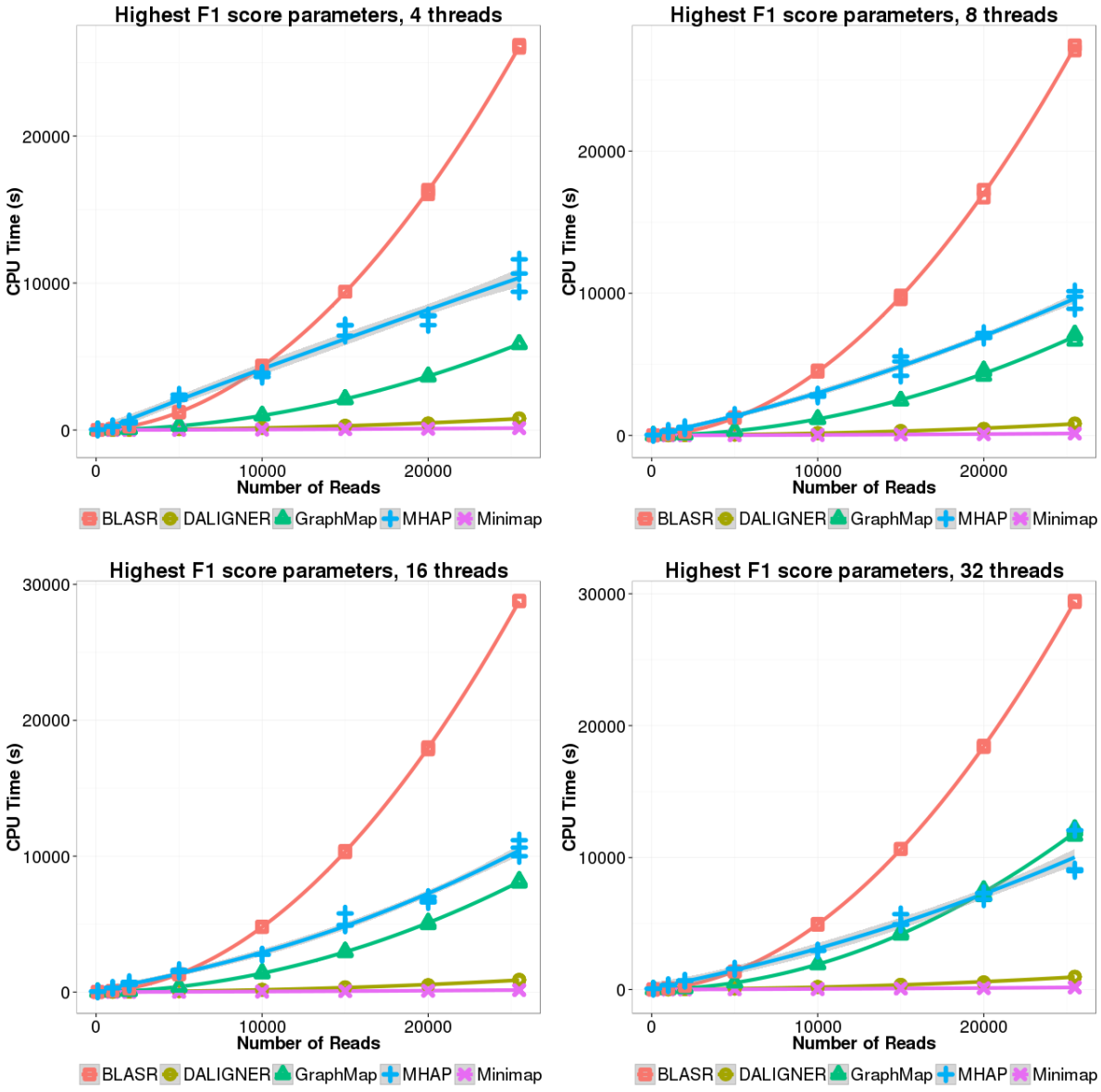
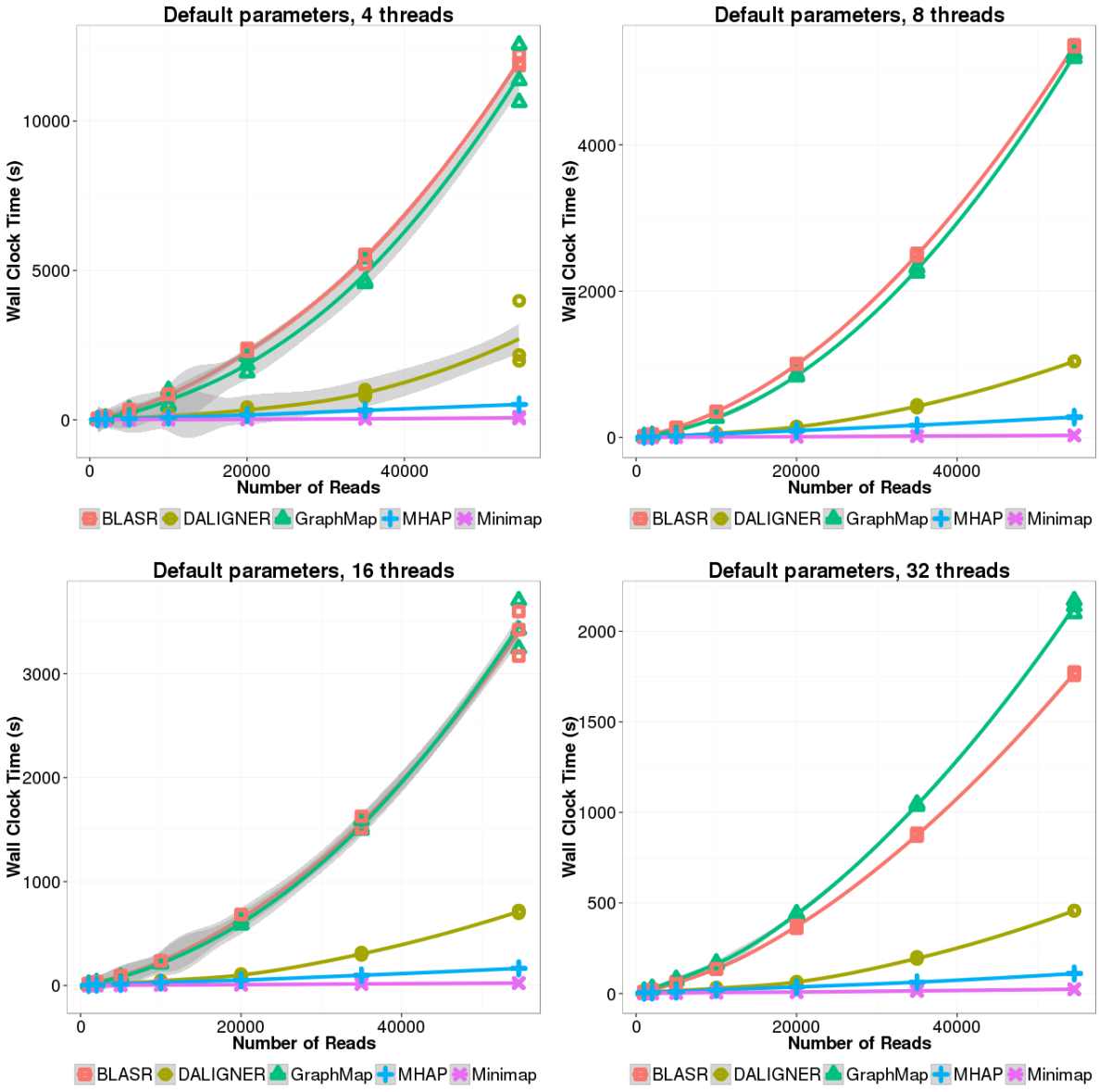


Figure D.8 CPU time on the ONT SQK-MAP-006 *E. coli* dataset at a differing number of randomly subsampled reads. Each tool was parameterized using default settings or settings yielding the highest F1 score.


Default parameters, 4 threads
Wall Clock Time (s)
10000
5000
0
0
20000
40000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Default parameters, 8 threads
Wall Clock Time (s)
4000
2000
0
0
20000
40000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Default parameters, 16 threads
Wall Clock Time (s)
3000
2000
1000
0
0
20000
40000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Default parameters, 32 threads
Wall Clock Time (s)
2000
1500
1000
500
0
0
20000
40000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap

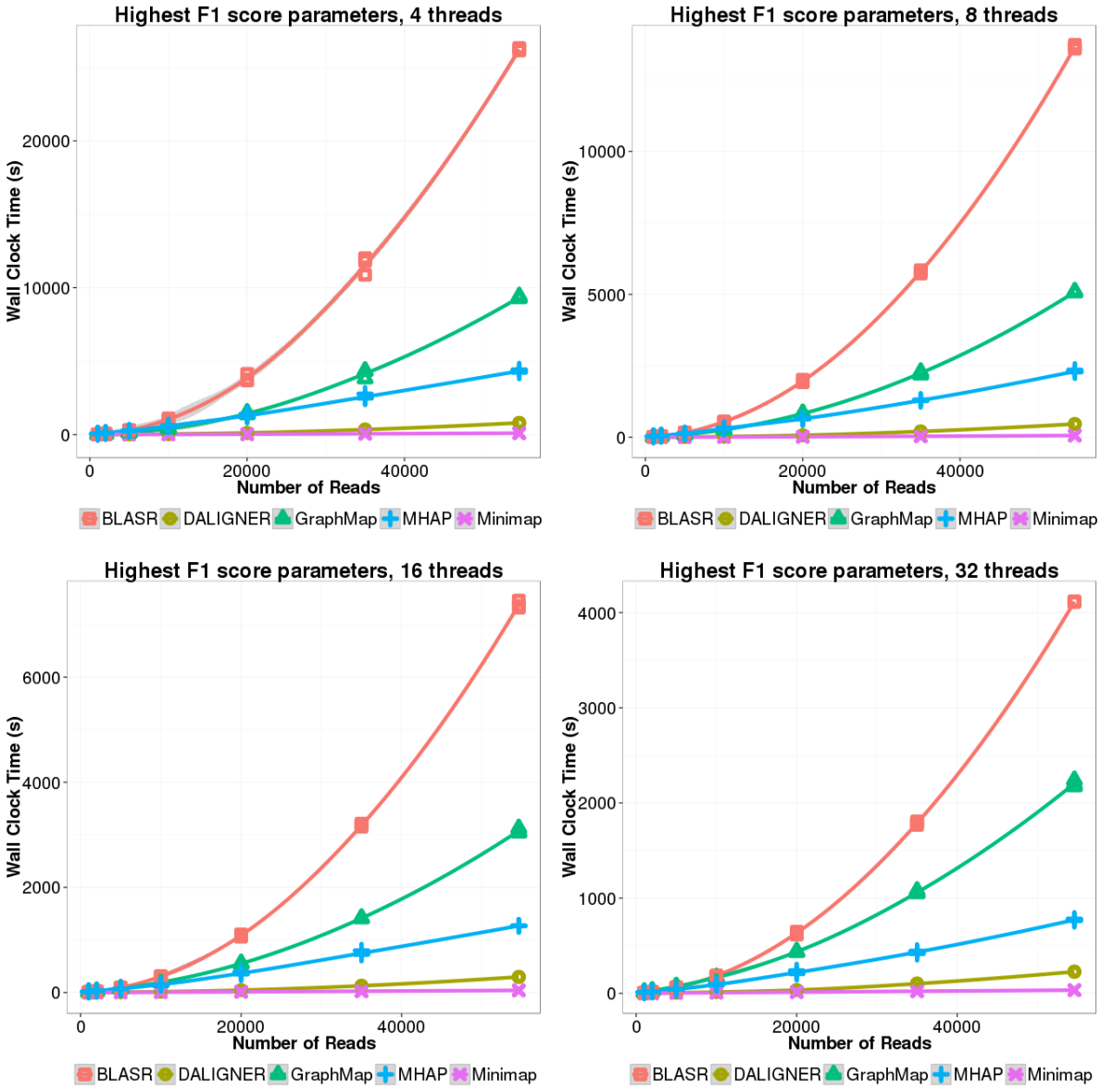
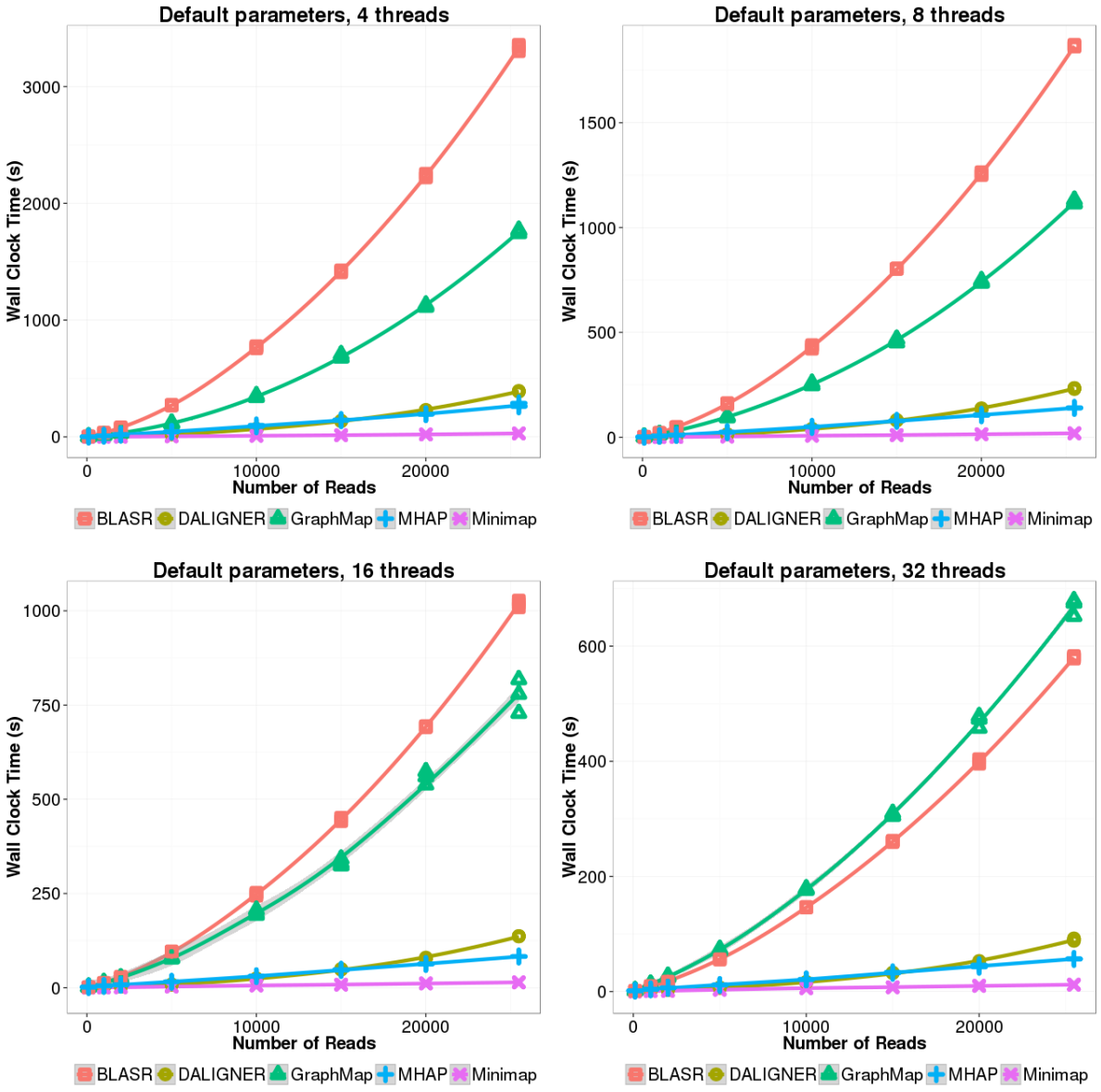


Figure D.9 Wall clock time on the PB P6-C4 *E. coli* dataset at a differing number of randomly subsampled reads. Each tool was parameterized using default settings or settings yielding the highest F1 score.


Default parameters, 4 threads
Wall Clock Time (s)
3000
2000
1000
0
0
10000
20000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Default parameters, 8 threads
Wall Clock Time (s)
1500
1000
500
0
0
10000
20000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Default parameters, 16 threads
Wall Clock Time (s)
1000
750
500
250
0
0
10000
20000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Default parameters, 32 threads
Wall Clock Time (s)
600
400
200
0
0
10000
20000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap

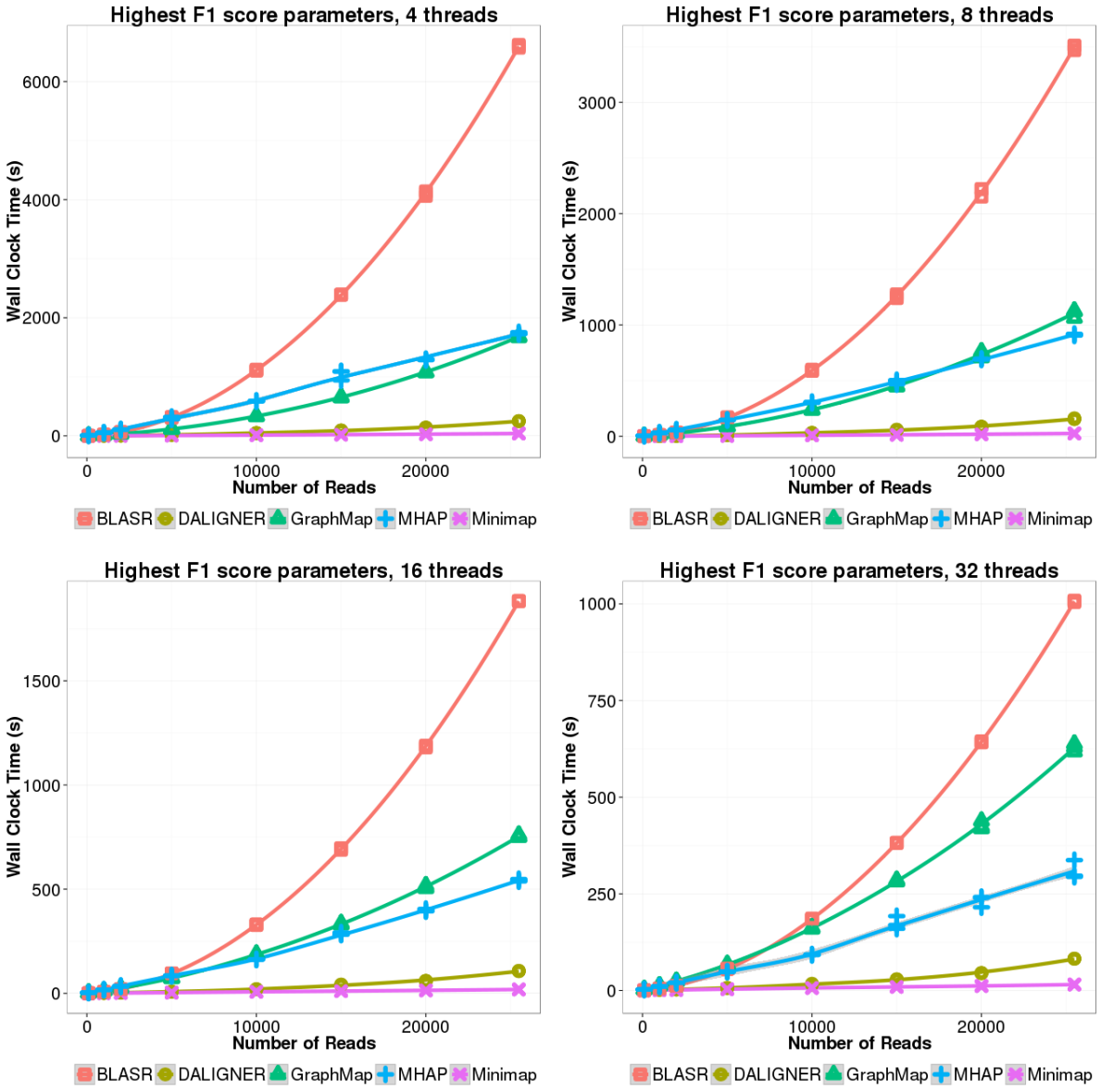
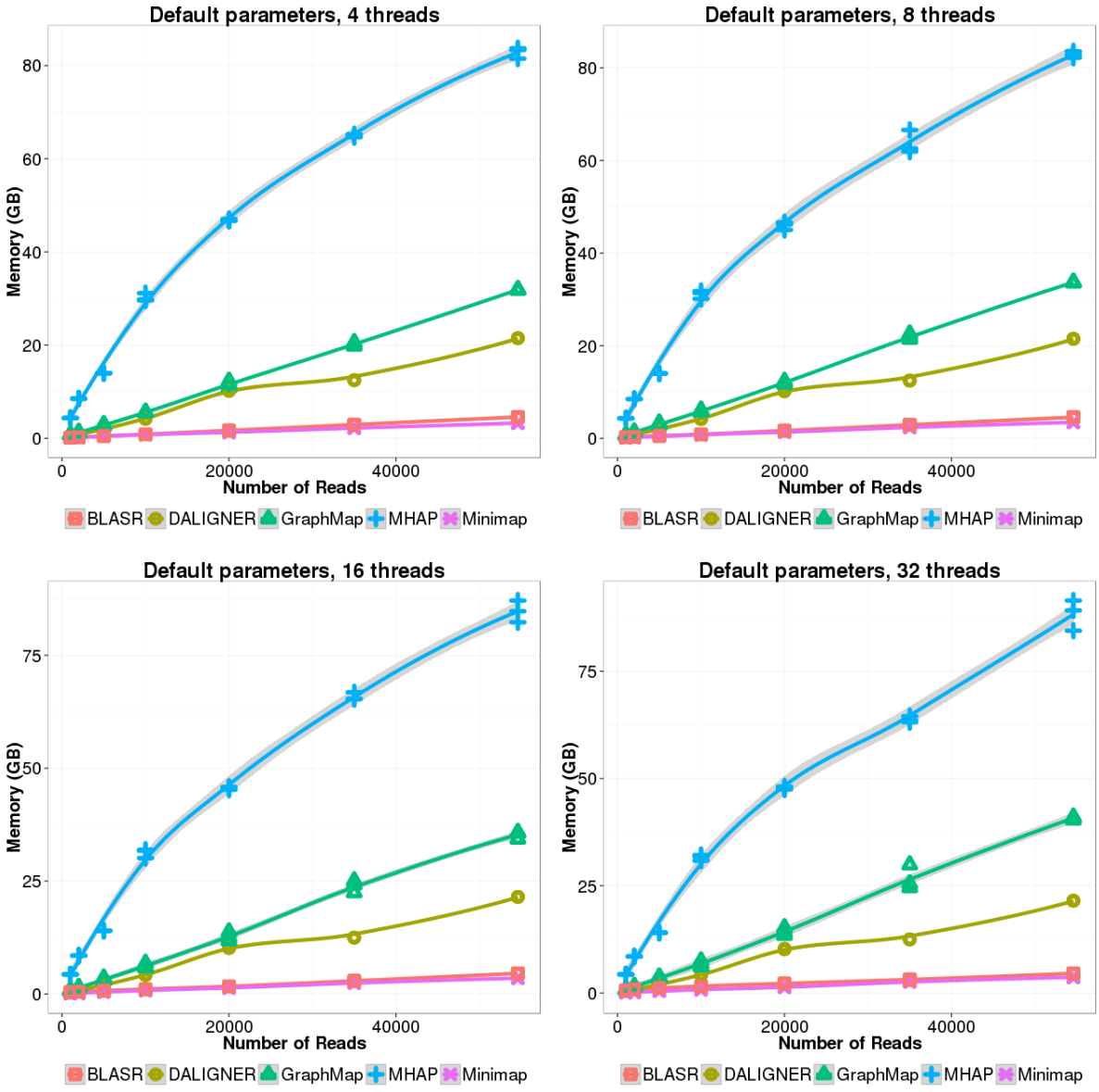


Figure D.10 Wall clock time on the ONT SQK-MAP-006 *E. coli* dataset at a differing number of randomly subsampled reads. Each tool was parameterized using default settings or settings yielding the highest F1 score.


Default parameters, 4 threads
Default parameters, 8 threads
Default parameters, 16 threads
Default parameters, 32 threads
Memory (GB)
Number of Reads
0
20000
40000
0
20
40
60
80
0
25
50
75
BLASR
DALIGNER
GraphMap
MHAP
Minimap

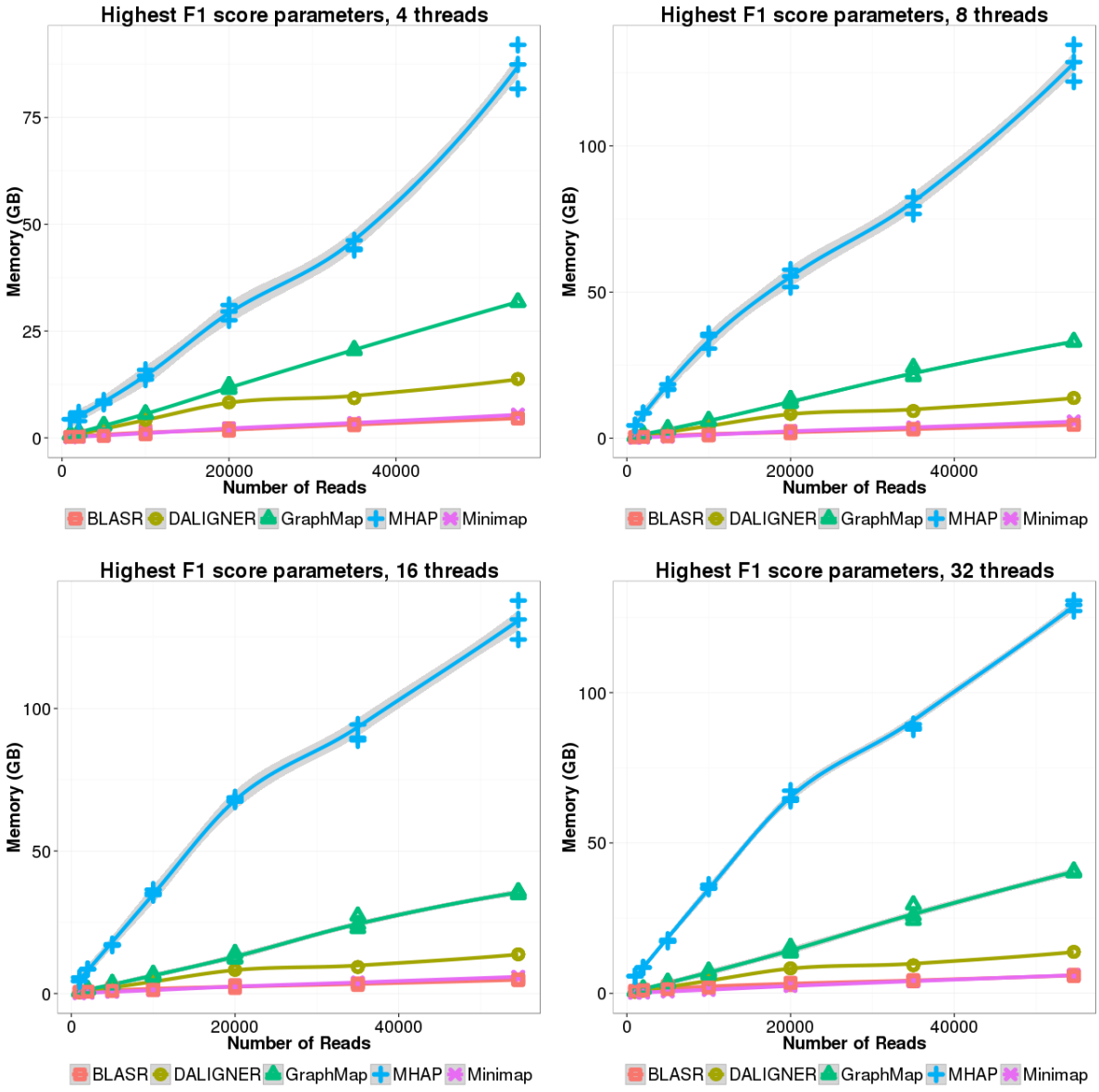
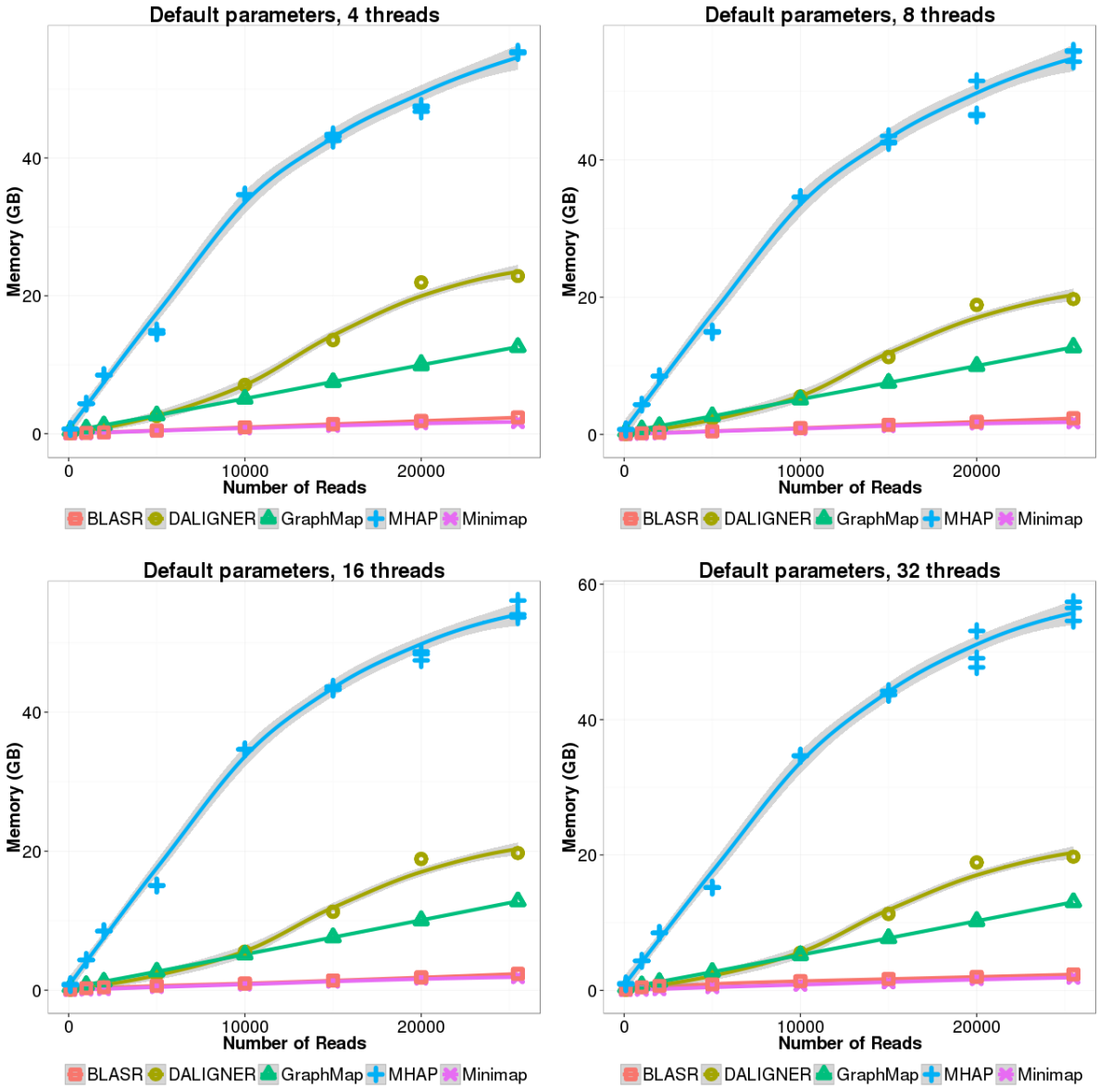


Figure D.11 Peak memory usage on the PB P6-C4 *E. coli* dataset at a differing number of randomly subsampled reads. Each tool was parameterized using default settings or settings yielding the highest F1 score.


Default parameters, 4 threads
Default parameters, 8 threads
Default parameters, 16 threads
Default parameters, 32 threads
Memory (GB)
Number of Reads
0
10000
20000
0
20
40
60
BLASR
DALIGNER
GraphMap
MHAP
Minimap

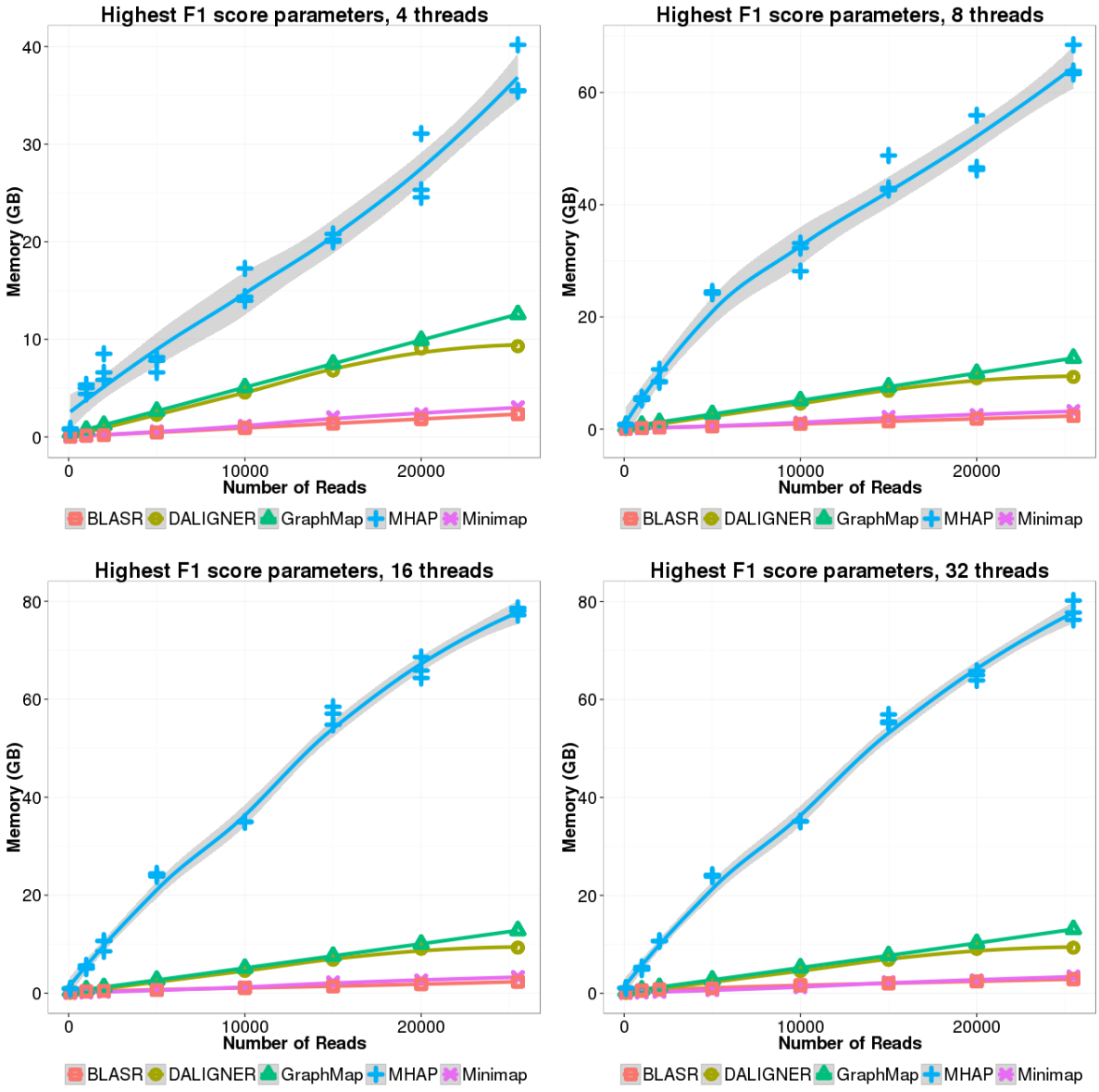
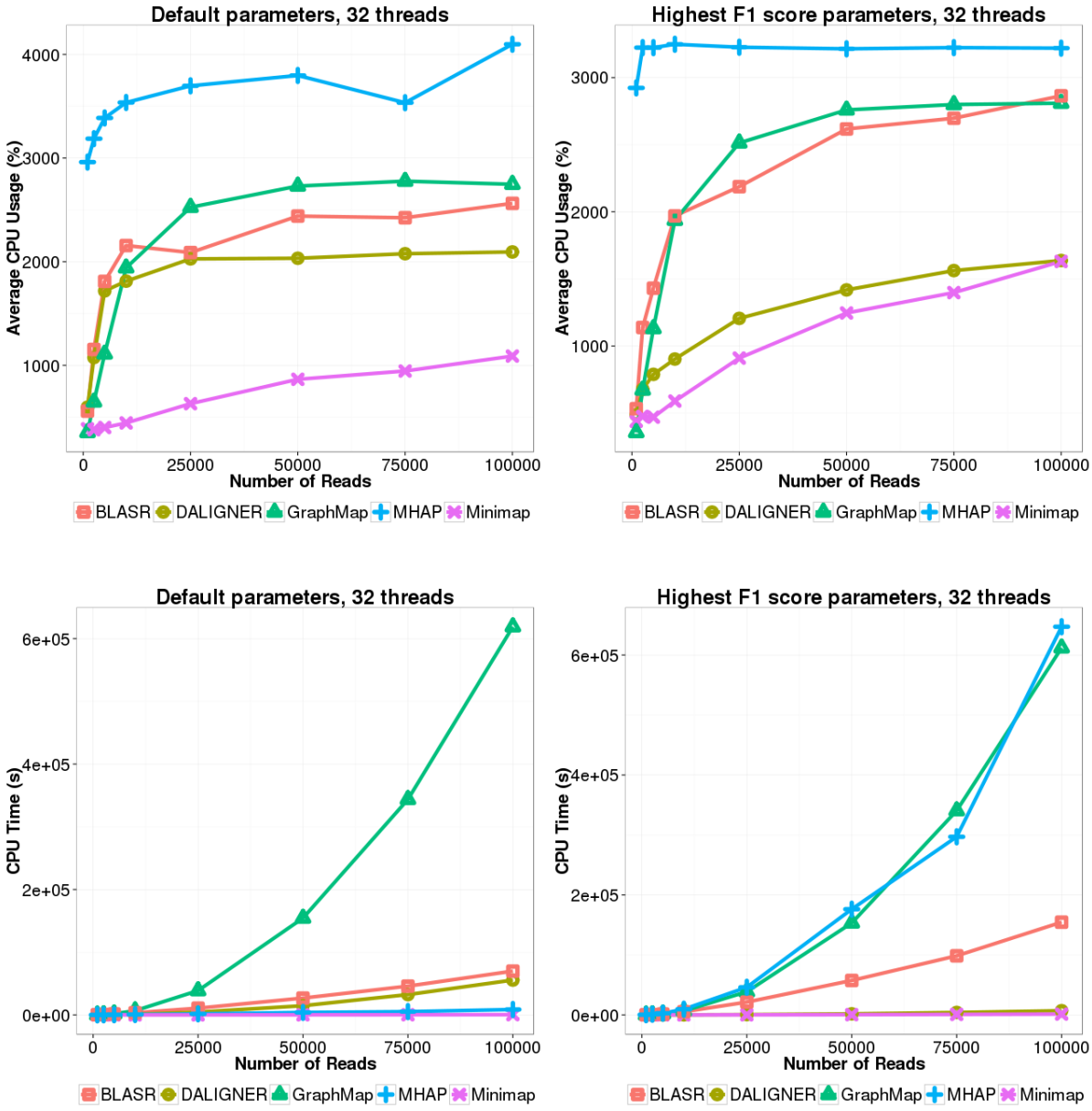


Figure D.12 Peak memory usage on the ONT SQK-MAP-006 *E. coli* dataset at a differing number of randomly subsampled reads. Each tool was parameterized using default settings or settings yielding the highest F1 score.


Default parameters, 32 threads
Average CPU Usage (%)
4000
3000
2000
1000
0
25000
50000
75000
100000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Highest F1 score parameters, 32 threads
Average CPU Usage (%)
3000
2000
1000
0
25000
50000
75000
100000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Default parameters, 32 threads
CPU Time (s)
6e+05
4e+05
2e+05
0e+00
0
25000
50000
75000
100000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Highest F1 score parameters, 32 threads
CPU Time (s)
6e+05
4e+05
2e+05
0e+00
0
25000
50000
75000
100000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap

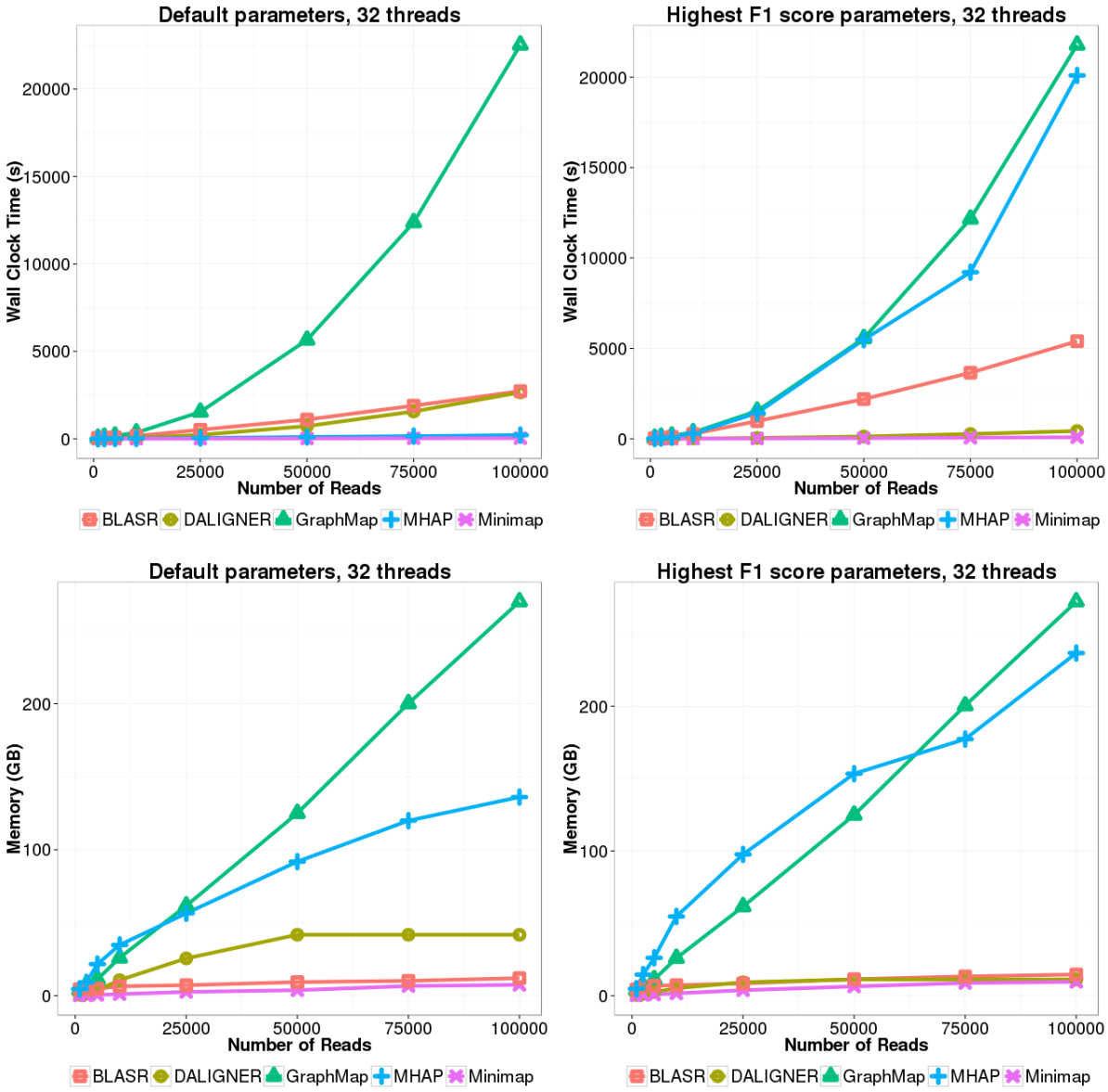
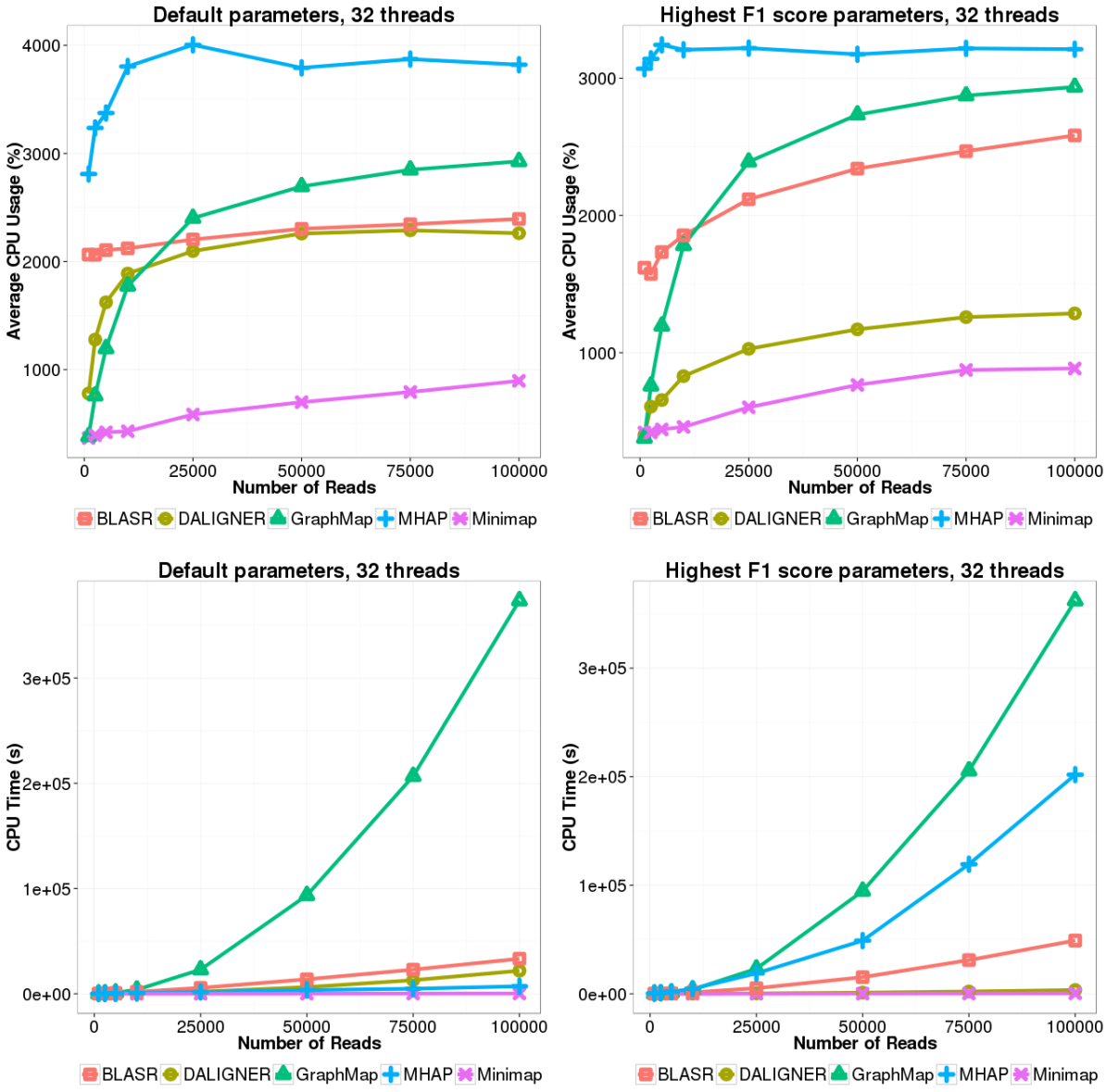


Figure D.13 Average CPU usage, CPU time, Wall clock time and peak memory usage on the PB P6-C4 *C. elegans* dataset at a differing number of randomly subsampled reads. Each tool was parameterized based on the runs yielding the highest F1 score.


Default parameters, 32 threads
Average CPU Usage (%)
4000
3000
2000
1000
0
25000
50000
75000
100000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Highest F1 score parameters, 32 threads
Average CPU Usage (%)
3000
2000
1000
0
25000
50000
75000
100000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Default parameters, 32 threads
CPU Time (s)
3e+05
2e+05
1e+05
0e+00
0
25000
50000
75000
100000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap
Highest F1 score parameters, 32 threads
CPU Time (s)
3e+05
2e+05
1e+05
0e+00
0
25000
50000
75000
100000
Number of Reads
BLASR
DALIGNER
GraphMap
MHAP
Minimap

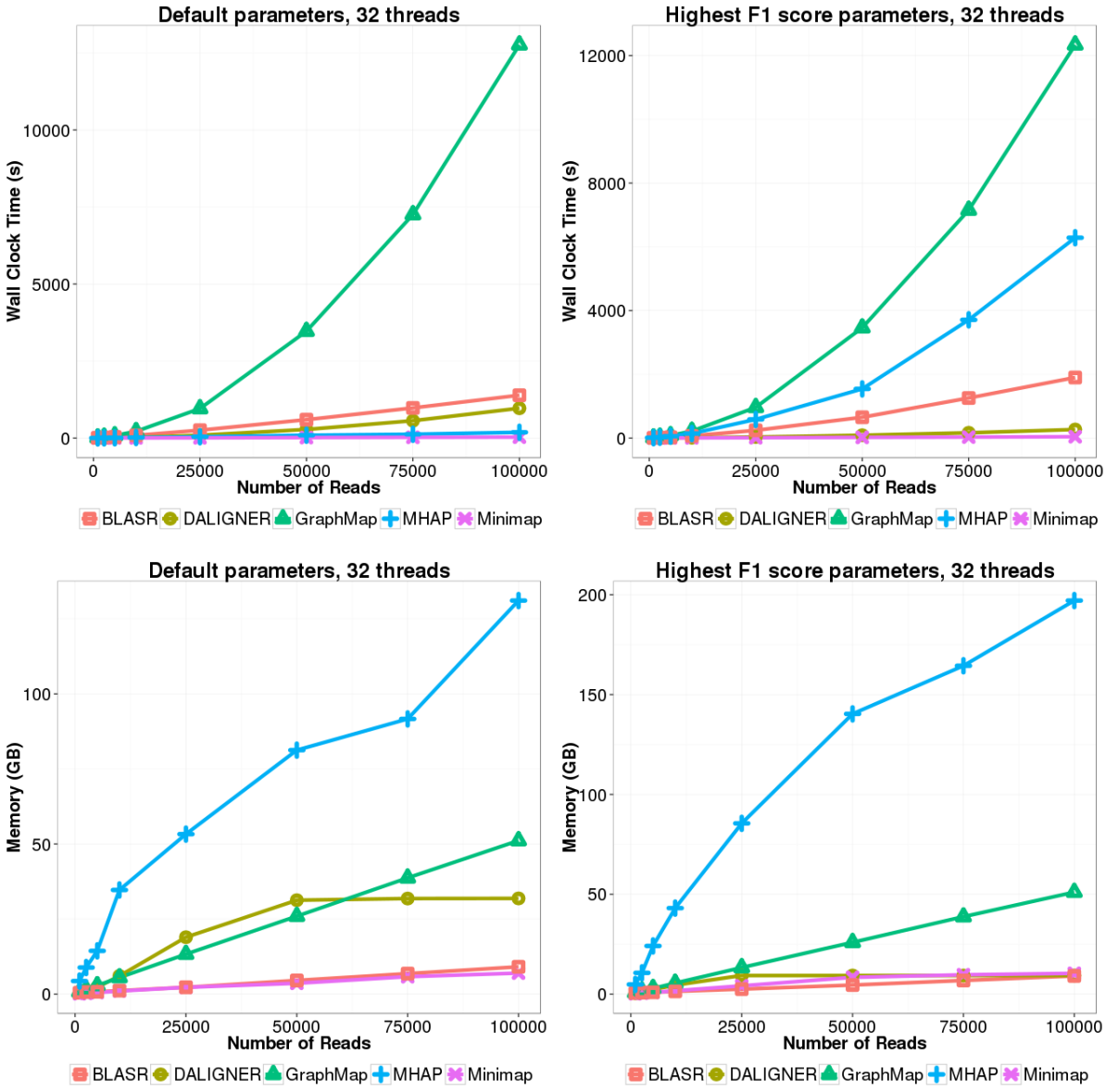


Figure D.14 Average CPU usage, CPU time, Wall clock time and peak memory usage on the simulated ONT SQK-MAP-006 *C. elegans* dataset at a differing number of randomly subsampled reads. Each tool was parameterized based on the runs yielding the highest F1 score.