ABSTRACT

Title of Dissertation: OPTIMIZING THE ACCURACY OF LIGHTWEIGHT METHODS FOR SHORT READ ALIGNMENT AND QUANTIFICATION

Mohsen Zakeri
Doctor of Philosophy, 2021

Dissertation Directed by: Professor Rob Patro
Department of Computer Science

The analysis of the high throughput sequencing (HTS) data includes a number of involved computational steps, ranging from the assembly of reference sequences, mapping or alignment of the reads to existing or assembled sequences, estimating the abundance of sequenced molecules, performing differential or comparative analysis between samples, and even inferring dynamics of interest from snapshot data. Many methods have been developed for these different tasks that provide various trade-offs in terms of accuracy and speed, because accuracy and robustness typically come at the expense of sacrificing speed and vice versa. In this work, I focus on the problems of alignment and quantification of RNA-seq data, and review different aspects of the available methods for these problems. I explore finding a reasonable balance between these competing goals, and introduce methods that provide accurate results without sacrificing speed.

Alignment or mapping of sequencing reads to known reference sequences is a challenging computational step in the RNA-seq pipeline mainly because of the large size of sample data and reference sequences, and highly-repetitive sequence. Recent

quantification methods introduced the concept of lightweight alignment in order to accelerate the mapping step, and therefore, the whole quantification pipeline. I collaborated with my colleagues to explore some of the shortcomings of the lightweight alignment methods, and to address those with a new approach called the selective-alignment. Moreover, we introduce an aligner, Puffaligner, which benefits from both the indexing approach introduced by the Pufferfish index and also selective-alignment to producing accurate alignments in a short amount of time compared to other popular aligners.

To improve the speed of RNA-seq quantification given a collection of alignments, some tools group fragments (reads) into equivalence classes which are sets of fragments that are compatible with the same subset reference sequences. Summarizing the fragments into equivalence classes factorizes the likelihood function being optimized and increases the speed of the typical optimization algorithms deployed. I explore how this factorization affects the accuracy of abundance estimates, and propose a new factorization approach which demonstrates higher fidelity to the non-approximate model.

Finally, estimating the posterior distribution of the transcript expressions is a crucial step in finding robust and reliable estimates of transcript abundance in the presence of high levels of multi-mapping. To assess the accuracy of their point estimates, quantification tools generate inferential replicates using techniques such as Bootstrap sampling and Gibbs sampling. The utility of inferential replicates has been portrayed in different downstream RNA-seq applications, i.e., performing differential expression analysis. I explore how sampling from both observed and unobserved data points (reads) improves the accuracy of Bootstrap sampling. I demonstrate the utility of this approach in estimating allelic expression with RNA-seq reads, where the absence of unique mapping reads to

reference transcripts is a major obstacle for calculating robust estimates.

Optimizing the accuracy of lightweight methods for short read alignment and quantification

by

Mohsen Zakeri

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2021

Advisory Committee:
Professor Rob Patro, Chair/Advisor
Professor Mihai Pop
Professor John Dickerson
Professor Erin Molloy
Professor Michael Cummings, Dean's representative

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ARD	**A**bsolute **R**elative **D**ifference
BWT	**B**urrows-**W**heeler **T**ransform
DP	**D**ynamic-**P**rogram
EM	**E**xpectation **M**aximization
HTS	**H**igh **T**hroughput **S**equencing
Indel	**In**sertion (and) **D**eletion
MARD	**M**ean **A**bsolute **R**elative **D**ifference
MEM	**M**aximal **E**xact **M**atch
NGS	**N**ext **G**eneration **S**equencing
SA	**S**uffix **A**rray
SGS	**S**econd **G**eneration **S**equencing
SMEM	**S**uper **M**aximal **E**xact **M**atch
uni-MEM	**U**nique **M**aximal **E**xact **M**atch
VBEM	**V**ariational **B**ayesian - **E**xpectation **M**aximization

Chapter 1: Introduction

Out of four major biological macromolecules (proteins, carbohydrates, lipids and nucleic acids), nucleic acids carry the most significant information about the identity of each organism. Even within each organism, the different content of nucleic acids in different organs and cells, defines their main functions and characteristics, also known as phenotypes. There are four types of nucleic acids (**A**denine, **C**ytosine, **T**hymine(**U**racil), **G**uanine) which are the main components of the **D**eoxyribo**N**ucleic **A**cids (DNA) and **R**ibo**N**ucleic **A**cid (RNA) in living organisms (Archaea, Bacteria, and Eukarya). Each DNA or RNA molecule is formed by a sequence of the nucleic acids. While the DNA content of different organisms are distinct, the DNA molecules across all cells of each individual are almost identical. Even during the cell division, all the DNA molecules are duplicated and preserved in each new cell's nucleus in the form of chromosomes. On the other hand, there exists different types of RNA molecules in different cells of each organism, leading to their different functions. RNA molecules are created from specific regions of chromosomes, called genes, in a process called transcription. A gene is called expressed in a specific cell if it is transcribed to RNA molecules. Different genes being expressed in different cell types leads to their vastly various functions. The set of all the genes, and the set of all the RNA molecules present in a cell are called the genome and

the transcriptome respectively.

The transcription of a gene is started by a specific protein called the RNA polymerase. This protein copies the sequence of nucleic acids from a gene into a new RNA-sequence, called the pre-mRNA. There are two main types of subsequences present in each pre-mRNA, introns (intragenic regions) and exons (expressed regions).The pre-mRNA molecules turn into the mRNA molecules after the intronic regions are spliced out. Alternative splicing of the set of introns and exons generates various mRNA molecules from a single gene. The set of all mRNA molecules generated from a single gene are called the isoforms or transcripts of the gene. Figure 1.1 shows how two different isoforms are generated from a single gene through alternative splicing.

Many technologies have been proposed for gathering information about the transcriptomic contents of an organism. RNA sequencing (RNA-seq) is a powerful sequencing technique, and has become very popular since its introduction [1]. In the RNA-seq protocols, RNA sequences are first fragmented into smaller pieces, then these fragments are amplified through the PCR process, and finally the set of amplified fragments are sequenced and read sequences are generated. If both ends (5' and 3') of the fragments are sequenced paired end reads will be generated, while single end reads are sequenced only from a single end of each fragment. Transcriptome assembly, detecting novel isoforms, and measuring the expression level of any isoform in a sample, are some of the main important applications of RNA-seq data. Mapping or alignment of RNA-seq reads to the set of known references is one of the early computational steps in many of these applications. Throughout this section, some famous computational approaches proposed for this critical step are introduced.

The number of reads generated from each isoform of a gene depends on the gene's level of expression in the cell. Each gene consists of encoding segments called exons. Each transcript or isoform of a gene consists of a set of particular exons. To illustrate the alternative splicing, consider the example in fig. 1.1 which shows two different isoforms of $gene_a$ which is a gene with three exons. Different splicing events can lead to new combinations of exons in each isoform, e.g., t_1 and t_2 are two isoforms generated from $gene_a$, each containing specific subset of $gene_a$'s exons. The number of reads mapping to specific exons and exon junctions indicate the expression level of the isoforms containing those exons. According to sequence similarities in different genes or exons shared between different isoforms of a gene, a read may map to multiple isoforms in the transcriptome, this ambiguity makes abundance estimation challenging. An example of reads generated from different isoforms of a gene in RNA-seq experiments is displayed in fig. 1.1. In this example, green and blue reads suggest expressions of both t_1 and t_2, while red reads are only compatible with t_1.

Reads spanning two different exons can be evidence for the splicing events, e.g., the green-blue read in fig. 1.1 spans the red and green exons which suggest the existence of an isoform containing these two exons next to each other, i.e., t_2. In this example, t_1 and t_2 are the known isoforms (also called transcripts) of the gene $gene_a$. In an RNA-seq experiment reads might be generated which do not map to any known isoforms of genes, e.g., the gray reads mapping to the intronic gray region of $gene_a$ in fig. 1.1 or green-gray read which maps to the exon-intron junction. Presence of such reads could be evidence for discovering new exons or isoforms of $gene_a$, or preserved intronic regions in the isoforms (intron retention). Finding evidence for novel isoforms or intron retention events is an

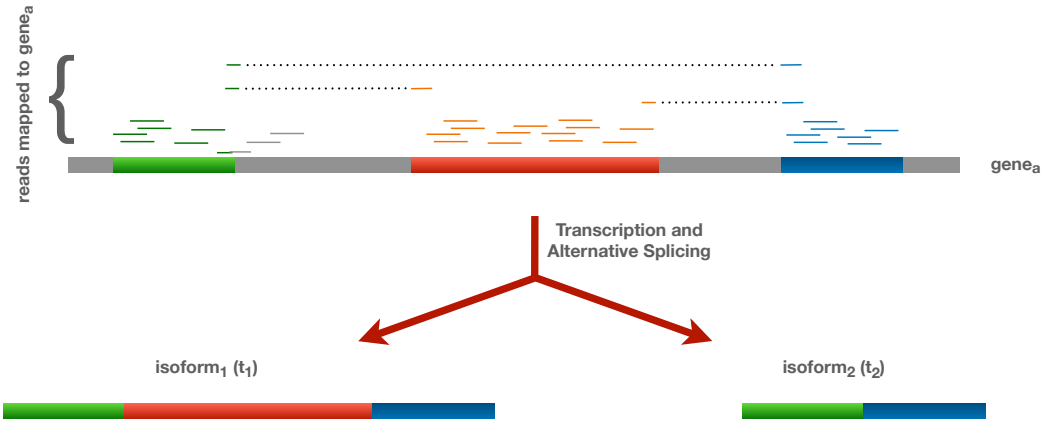


Figure 1.1: An example of splicing events in the gene $gene_a$, resulting in two transcripts t_1 and t_2. According to sequence similarities the RNA-seq reads might be mapped to single or multiple isoforms.

important advantage of RNA-seq assay over alternative sequencing techniques.

In the following sections of this chapter I review the existing tools which are widely used for alignment (computing the compatibility of the reads to the reference sequences) and quantification (estimating the expression ratio of the isoforms) of RNA-seq samples. Both of these steps greatly affect the accuracy of the downstream analysis in RNA-seq pipelines [2]. In chapter 2, I present a fast mapping technique called selective-alignment which is built alongside *quasi-mapping* to achieve a higher sensitivity which results in more accurate abundance estimates. Chapter 2 also introduces a new tool for computing alignment of DNA-seq and RNA-seq reads to known reference sequences. In chapter 3, an improved factorization of the quantification likelihood function is presented which maintains a high fidelity to the underlying data and will result in higher confidence for more fine grained analysis of transcripts.

1.1 Mapping RNA-seq reads to a known transcriptome

Alignment is a crucial and expensive computations step in an RNA-seq analysis pipeline. The main goal of this step is to find, for each read, the region on the reference genome (or transcriptome) from which it is originally fragmented. The length of the short RNA-seq reads is usually between 100 and 200 bases, while the latest human genome sequence is about three billion bases. The goal, for each read, is to find a substring in the reference sequences which best matches the read. Often it is not possible to find an exact match for a read because of the variations present in the samples with respect to the reference sequences. The variations are divided into two main types, variations introduced by technical errors or the biological variations existing in each new sample. Therefore, the alignment procedure most of the time results in an inexact match for each read rather than a region which exactly matches the read. In order to find the most compatible reginos in the reference sequences to each read, we should define the compatibility of two sequences. The compatibility is measured by the number of edits required to convert one sequence to the other one. Different types of edits are considered which are substitution, insertion and deletion. The penalties assigned to each type of edit might be different and can be usually configured in most of the aligners. The sum of all the penalties is considered to be the edit distance between two sequences. Therefore, we can define the alignment problem as finding the region on the reference with the minimum edit distance to each read. This can be achieved by classical algorithms such as Smith Waterman [3] in $O(n * m)$ time and $O(n * m)$ space, where n and m are the length of the reference sequence and the read sequence respectively. This solution becomes super expensive

when the length of reference sequences and the number read sequences are very large which is the common case in the RNA and DNA sequences. Therefore, a number of additional solutions proposed to accelerate this procedure while maintaining the accuracy of this approach. In the following sections, some of the main common approaches will be discussed briefly.

1.1.1 The main approaches for computing read alignments

One of the most common approaches for accelerating the alignment problem is "seed and extend". The seeds are supposed to reduce the search space for the alignment problem into smaller regions that are most likely to include a reasonable alignment for the queried read rather than comparing each read sequence to all the reference sequences. Seeds are often shorter than the reads and represent an exact match from a substring in the read to some region in reference. The seeds are later extended into full alignments for the read by computing the full alignment of the reads to the regions identified by each read.

Finding the seeds requires the pre-processing of the reference sequences which is called the indexing step. Different indexing strategies are used in different aligners which have various space and time requirements. There are two main types of indices, full-text indices and hash-based indices. The full-text indices are often smaller in size, and a sequence of any different size can be queried in them, while the hash based indices take more space and only accept queries with a fixed size. The main benefit of the hash based indices are their speed compared to the full text indices. Full text indices are used in

popular RNA-seq aligners such as Bowtie [4], Bowtie2 [5], BWA [6] and STAR [7]. It is worth noting that STAR's index employs a hybrid approach of both full text and hash based indices which results in being faster at the expense of larger memory requirements. Other aligners such as deBGA [8], Minimap2 [9], are based on hash based indices and often use exact k-mer searches as the first step of the alignment step to find the seeds. K-mers of each read are all the substrings of length k in the read sequence. Querying a k-mer into the index results in finding all the locations on the reference sequences where the k-mer exists.

FM-index and Suffix Array are two closely related full text indices. The suffix Array (SA) of a string S with the length n is defined by the sorted order of all suffixes of string S concatenated with a terminal character which is lexicographically smaller than all other characters in the alphabet. Adding the terminal character ($) ensures that no suffix is the prefix of any other suffix in S. In practice, the suffix array stores only the indices corresponding to each suffix in an array. If pattern p exist in the string S, then, it will be a prefix of some suffix in S. Lexicographically sorting the suffixes in SA, provides this property that all suffixes which include some pattern p as their prefix, will appear in consecutive rows in the SA. Therefore, to query a pattern in S, it suffices to find an interval [a,b) in the SA which are all the suffixes that include p as their prefix. Each pattern can be searched in the SA by a binary search, which takes O(log(—S—)x—p—). The search process can be enhanced by keeping some extra information like the longest common prefix lengths (LCP) for some pair of suffixes, as a result the query time will be O(log—S—+—p—) instead. STAR is one of the most popular aligners which use the Suffix Array to index the reference sequences. [cite] STAR uses some hash tables for

accelerating the search process as well which comes at the cost of increasing the index size.

FM-Index is another full text index which consists of some auxiliary data structures alongside the Burrows Wheeler Transform (BWT) of the reference string S. BWT(S) is closely related to the Suffix Array. To enable efficient search for every pattern using BWT, the LF mapping property in the BWT is utilized with the help of storing the occurrence information of every character in the BWT. Using the succinct data structures, this can be stored in O(—S—) space. One other useful characteristic of the BWT is that it tends to put repetitions of each character next two each other, this doesn't mean that all repetitions are put next to each other, but it is common to find longer substrings of A or any other character in the BWT of a sequence compared to the original sequence. This property of the BWT makes it more compressible compared to the original sequence which results in smaller index size. Bowtie, Bowtie2, and BWA are some of the popular aligners which index the reference sequences with a FM-Index.

The other main type of indices used for finding the alignment of a query in a large set of reference sequences are hash based indices. Hash based indices use substrings of a fixed size from the reference sequence to search each new query. The substrings of length k from the reference sequences are called k-mers. K-merss constitute the keys in the hash based indices. Hash based indices store the location where each k-mer occurs in the reference sequences. During the query time, we are able to extract all the k-mers from each read and query those e in the set of keys of the hash based index. The index will then retrieve all the positions each k-mer occurs in the reference which will later play the role of the seeds for the seed and extend procedure. It is important to note that queries of

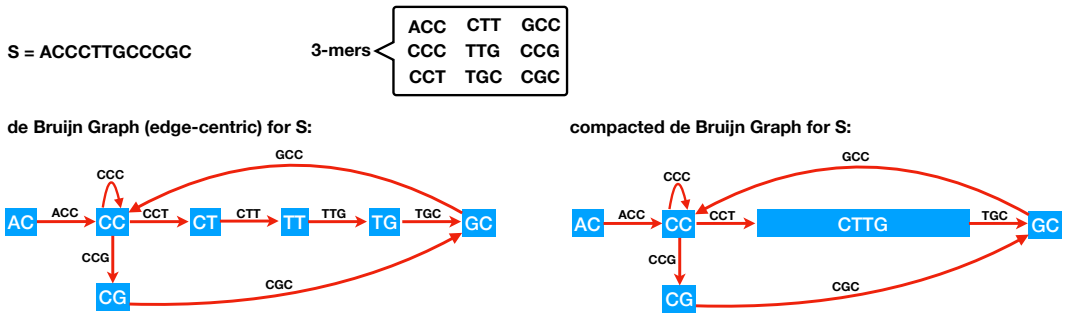


Figure 1.2: The edge-centric de Bruijn graph and the compacted De Bruijn graph for the sequence S. There are 9 3-mers in this sequence which correspond to the edges in the de Bruijn graph. 2-mers form the nodes of the de Bruijn graph. 3 nodes (CT, TT and TG) are combined together in the compacted De Bruijn graph since they are on a non-branching path of the de Bruijn graph.

length smaller than k cannot be made into these hash based indices, so, one drawback of these types of indices is in order t o find a seed for the query in the reference, there needs to be at least one substring of length k in the read which match the reference sequences without any edit distance. Therefore, the length of k should be carefully selected based on the error rate of the sequencing technology, so that with high probability at least one match from each read is found on the reference, if the read is actually originating from a position on the reference sequences.

Another approach for indexing all the k-mers of a set reference sequence is to build the de Bruijn graph. Each k-mer forms an edge in the de Bruijn graph between the prefix and suffix k-1 mers which it consists of. In the fig. 1.2, a de Bruijn graph is shown which is built from the sequence S = "ACCCTTGCCCGC" and the k equal to 3. One main property in the de Bruijn graph is that every k-mer appears exactly once in the graph. This property helps to reduce the redundancy of repeats which usually exist in the DNA and RNA sequences. The compacted de Bruijn graph is built after compacting the non branching paths from the original de Bruijn graph, as shown in the fig. 1.2. The length of

the nodes in the compacted de Bruijn graph might be larger than k-1, but still each k-mer appears at most once in the graph (either as an edge or as a substring in one node). If the de Bruijn graph is built from multiple sequences (e.g., the set of human transcripts, or a collection of microbial genomes), one is interested to know in which reference sequences each k-mer appears. A k-mer might appear in multiple reference sequences due to shared exons in transcripts, or sequence similarities in different strains of some species.

A colored de Bruijn graph stores this data for each k-mer (edge) of the graph as the color information. For example, in the case of the transcriptome, each color represents the set of transcripts in which the k-mer corresponding to that edge exists. K-mers appearing in the same set of transcripts (e.g., due to shared exons) will have the same colors. A contig in the compacted De Bruijn graph is a non branching path in the graph, and if the edges are colored, not all the k-mers in a contig might have the same color. If all the k-mers that are part of a non branching path also have the same color set assigned, those k-mers can be combined together and form a unitig in the colored compacted de Bruijn graph.

Pufferfish [10] is a space and time efficient index built on top of the compacted colored de Bruijn graph. So, it can be used to perform efficient k-mer queries from each read sequence to large transcriptomic or genomic references. In the second chapter of this manuscript, we introduce Puffaligner which uses the Pufferfish index to query k-mers from the short RNA-seq reads to compute read alignments through the seed and extend strategy.

1.1.2 Alignment Free Approaches for Mappings reads

Exploiting the methods of abundance estimation for RNA seq reads revealed the fact that although alignment is very useful for finding the candidate transcripts to which reads map, the full alignment information (The exact details of all the gaps and mismatches) are unnecessary for performing quantification. In fact the position, orientation and length of the matching fragment on each transcript is adequate for achieving accurate estimates. Therefore, lightweight methods were developed to avoid performing full alignments upstream of the quantification. These methods typically build an index over the reference sequence (similar to the FM index in alignment tools). Then, in the quantification step, the reads are mapped to the reference on the fly, rapidly, using the built-in index. Therefore, the memory peak footprint of such methods is bounded by the reference size and complexity and scales well as the number of reads increases. Note that for performing quantification over multiple samples to the same reference, the index needs to be built only once.

The first lightweight algorithm for mapping reads to reference transcripts was introduced in *Sailfish* [11]. *Sailfish* is an alignment-free quantification tool which builds an index over all subsequences of size k from the reference, called k-mers . The *Sailfish* index consists of a perfect hash function mapping each k-mer in the reference to a unique integer, an array indicating the counts of each k-mer , an index mapping each k-mer to the set of transcripts in which it appears, and another index mapping each transcript to the multiset of k-mers it contains. In the quantification step, *Sailfish* explores all the k-mers the read contains and keeps the count for the ones appearing in the reference. So instead of mapping the whole reads, *Sailfish* only maps the k-mers , and uses their count for each

transcript as evidence for the relative expression of the transcripts. Although *Sailfish*'s approach is 30 times faster than fastest quantification tools which perform alignment upstream of quantification, it suffers from increased ambiguity of a large rate of multi mapping k-mers , which sometimes reduces the accuracy of abundance estimation. The smaller k size causes a higher multi mapping rate, while the larger size of the k results in less robustness to sequencing errors because each k-mer is mapped with no error using the perfect hash function.

The idea of mapping k-mers instead of the whole reads to reference transcriptome introduces a huge improvement in the speed of quantification tools. However, it is sub-optimal to only consider occurrence of subsequences of size k as evidence of expression while the observed data is of larger length. Hence, the developers of *Sailfish* introduced a new idea for mapping the whole reads using the perfect hash function of k-mers by benefiting from the suffix array data structure. The new mapping approach is called *Quasi-mapping* [12] and is utilized in the newer version of *Sailfish* software and also in the new quantification tool called Salmon [13].

The suffix array of a sequence is a sorted array of all of the sequence's suffixes. Therefore, all suffixes starting with the same prefix are located in adjacent positions of the suffix array. Note that only the position of the occurrence of suffixes in T are stored in the suffix array. The number of elements of the array is equal to the length of the sequence and there is a one-to-one mapping from each row of the suffix array to a character in the BWT of the sequence. We can also introduce suffix array intervals similar to intervals of the Burrows Wheeler transform. In the *quasi-mapping* index the suffix array $SA(T)$ is built from reference transcriptome T. Therefore, each row in the SA starts at a unique

transcript of the transcriptome. There is a hash function $I(k_i) = [b, c)$ from each k-mer k_i in T to a suffix array interval from row b until row c; all rows that contain k_i as a prefix. For mapping each read, the k-mers , k_i of the read (existing in the hash table) are hashed to find SA intervals. It is often possible to extend a match between the query and a subset of rows of the SA interval. The *quasi-mapping* algorithm attempts to find the longest subsequence of the query starting with the k_i as a prefix in the interval (also called maximum mappable prefix of k_i (MMP_i) [7]) with a binary search, as the SA is sorted lexicographically. *Quasi-mapping* retrieves a set of transcripts for each k-mer , the transcripts appearing in all sets are reported as mapping candidates for the read. The reverse complement of the read is also mapped and the sequence (either forward or reverse complement) with the higher number of matching k-mers determines the mapping orientation. For paired end reads, the other end of the read is also quasi-mapped to the reference. Then, transcripts appearing as candidates for both ends of the reads are reported as the mapping possibilities for the paired end reads.

It has been demonstrated that *quasi-mapping* finds very high quality mappings which result in highly-accurate abundance estimations. However, there are different aspects of the algorithm which can be modified in order to retrieve mappings with higher specificity and sensitivity. In fact, this idea is presented in chapter 2 as selective-alignment. *Quasi-mapping* extends the k-mer matches by the MMP length in order to find legitimate matches for queries. However, if an extension is not possible and no other k-mer match exists in the read, *quasi-mapping* may report all transcripts of the interval as the mapping candidates for the read. These low quality matches introduce a number of spurious mappings. A filtering process shall be introduced in order to filter the spurious hits in this

case. Other than suffering from spurious mappings *quasi-mapping* could also miss true mappings of the read in rare cases where errors are positioned adversarially on the read. An obvious case of losing the true mapping is if a read contains no subsequence of size k from the true transcript. In another case, the true mapping of the read might be lost from the SA interval by performing MMP extension, if a longer exact match of the read to the interval masks the match to the row with true location. The hits in the reverse complement of the read are only considered if there are less number of hits in forward strand compared to reverse complement. Therefore, spurious mappings in the forward strand might mask the true hit in the reverse complement. For some reads, multiple positions might be found on the same transcript where the read maps. *Quasi-mapping* greedily considers the left most one as the true mapping while that might not be the best possible matching of the read to that transcript. To address these challenges in *quasi-mapping*, selective-alignment is introduced as a new lightweight approach to achieve both higher sensitivity and specificity than *quasi-mapping*.

A similar approach to *quasi-mapping* is employed in *kallisto* [14] called pseudoalignment. *kallisto* index consists of a colored deBruijn graph from reference where nodes are k-mers and each node receives the colors of transcripts in which it appears. The contigs in the graph are formed from the linear stretches of the nodes (k-mers) with identical sets of colors. Kallisto also maintains a hash table mapping each k-mer to the contig it is contained in and the k-mer 's position in the contig. Using this index, the reads' k-mers are mapped to contigs. Since all the k-mers appearing in the same contig receive the same set of colors, and therefore the same transcripts, the rest of the k-mers in the contig can be skipped for mapping, similar to the idea of NIP skipping in *quasi-mapping*.

1.2 Abundance estimation of the transcriptome

In this section, we formalize the problem of abundance estimation with RNA-seq reads according to the model laid out by Li et al. [15]. There are M transcript types in transcriptome T, $t_1, t_2, ..., t_M$. In a given sample there are c_i copies of transcript type t_i, which are not observed directly.

1.2.1 The generative model of a sequencing experiment

The generative model of RNA-seq experiments states that the number of fragments sequenced from i^{th} transcript type is proportional to the total number of sequenceable nucleotides belonging to transcripts of type t_i. If the length of the i^{th} transcript is given by l_i, assuming all the reads have the size l_r, we can define effective length, $\tilde{l}_i = l_i - l_r + 1$ which is all possible start positions on transcript t_i for sequencing a read of size l_r. The portion of sequenceable nucleotides of transcript type t_i is $\eta_i = \frac{c_i l_i}{\sum_j c_j l_j}$ and $\alpha_i \propto \eta_i$, where α_i is the number of fragments drawn from transcripts of type t_i.

If $\mathcal{F}$ with $|\mathcal{F}| = N$, is the set of sequenced fragments, assuming independence for drawing each fragment, the likelihood of the underlying transcript abundances, θ, can be written as:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{F}) = \prod_{f_i \in \mathcal{F}} \sum_{j=1}^{M} \Pr(t_i \mid \boldsymbol{\theta}) \Pr(f_i \mid t_j). \tag{1.1}$$

The conditional probability of drawing a particular fragment f_i, given transcript t_j, $\Pr(f_i|t_j)$, is particularly critical for reaching accurate estimates and is derived from

mapping information. This term encodes, given parameters of the model and experiment, how likely it is to observe a specific fragment f_i arise from transcript t_j. Many terms can be included in such a conditional probability, some common terms include:

$$\Pr\left(d_i \mid f_i, t_j\right) = \frac{\Pr_D\left(d_i\right)}{\sum_{k=1}^{\tilde{l}_j} \Pr_D\left(k\right)}, \tag{1.2}$$

the probability of observing a mapping of implied length d_i for f_i given that it derives from t_j, where $\Pr_D\left(k\right)$ is the probability of observing a fragment of length k under the empirical fragment length distribution D;

$$\Pr\left(p_i \mid d_i, f_i, t_j\right) = \frac{1}{l_j - d_i + 1}, \tag{1.3}$$

the probability of a observing a mapping starting at position p_i for fragment f_i given that it has implied length d_i and is derived from t_j;

$$\Pr\left(o_i \mid f_i, t_j\right) = \begin{cases} \begin{cases} 0.5 & \text{if unstranded} \end{cases} \\ \begin{cases} 1.0 & \text{if compatible orientation} \\ \epsilon & \text{if incompatible orientation} \end{cases} & \text{if strand-specific} \end{cases}, \tag{1.4}$$

the probability of observing a mapping with a specific orientation o_j (i.e., forward or antisense) with respect to the underlying transcript for f_j, given t_i, ϵ (a user-defined constant), and knowledge of the underlying protocol, and

$$\Pr\left(a_i \mid f_i, o_i, d_i, p_i, t_j\right), \tag{1.5}$$

the probability of observing the particular alignment (e.g., `CIGAR` string) a_i for f_i given it is sampled from transcript t_j, has orientation o_i, implied length d_i and starts at position p_i—such a probability is calculated from a model of alignments, like those presented in [13, 15, 16].

In fact, one can conceive of many such general models of "fragment-transcript agreement" [13]. However, here we consider that $\Pr\left(f_j \mid t_i\right)$ is simply the product of the conditional probabilities defined in Equations (1.2) to (1.5), appropriately normalized.

1.2.2 Expectation-Maximization for optimizing the model parameters

Exact inference from the likelihood function is intractable for the large scale of RNA seq data. Local optimization methods, like expectation maximization (EM), are often applied to fit the best parameters in the model. The parameters of the model indicate the rate of expression for each transcript in the underlying samples. The EM approach is employed by both alignment based tools such as *RSEM* [15], *mmseq* [17], *IsoEM* [18] and also non-alignment based tools like *Sailfish* [11], Salmon [13] and *kallisto* [14].

Algorithm 1: Overview of the EM algorithm for optimizing the generative model

Data: transcriptome T, fragment set $\mathcal{F}$,conditional properties $\Pr(f_i|t_j)$ for fragment transcript pairs

Result: θ, relative abundance of transcripts

uniform initialization;

while *not converged* **do**

  for $t_i \in T$ **do**

    $\alpha_i = 0, \theta_i = \frac{1}{|T|}$

  end

  E-step:

  for $f_i \in \mathcal{F}$ **do**

    $sum = \sum_{t_k \in T} \theta_k \times \Pr(f_i|t_k)$

    for $t_j \in T$ **do**

      $\alpha_j += \frac{\theta_j \times \Pr(f_i|t_j)}{sum}$

    end

  end

  M-step:

  $sum = \sum_{t_k \in T} \frac{\alpha_k}{\tilde{l_k}}$

  for $t_i \in T$ **do**

    $\theta_i = \frac{\alpha_i / \tilde{l_i}}{sum}$

  end

end

The overview of the EM algorithm for optimizing eq. (1.1) is displayed in algorithm 1. In the E-step, the expected number of fragments sequenced from each transcript type in the sample is calculated. Using these expectations, alongside effective lengths the prior probability of observing each transcript type is obtained in the M-step. This iterative process is repeated until the convergence on θ values is reached.

If a transcript t_j is not present in the set of transcripts to which fragment f_i is mapped, the value of $\Pr(f_i|t_j)$ is equal to zero. The EM updates can benefit from the sparsity of $\Pr(f_i|t_j)$ matrix by only performing updates in the E-step for the set of transcripts that f_i maps to instead of the whole set of transcripts. Hence, if $\Omega(f_i)$ is the set of compatible transcripts with read f_i, in algorithm 1, the line 8 shall be modified to : $sum = \sum_{t_k \in \Omega(f_i)} \theta_k \times \Pr(f_i|t_k)$ and the loop iteration in line 9 to : $t_j \in \Omega(f_i)$.

1.2.3 Factorizations of the likelihood function

Each iteration of the EM algorithm updates the α values for each fragment independently. Although each update cost has collapsed considerably by benefiting from the sparsity of mapping matrix, the number of EM updates still scales with the number of fragments (and alignments). Sequence similarities in reads shall be utilized for factorizing the likelihood function, which results in bounding the number of updates as the number of fragments grows.

If a set of fragments, F', exactly map to the same set of transcripts, T', with the same conditional probabilities (meaning that for each pair $f_i, f_j \in F'$, $\Omega(f_i) = \Omega(f_j) = T'$ and the equation $\Pr(f_i|t_k) = \Pr(f_j|t_k)$ holds for all $t_k \in T'$), then all fragments in

F' are exactly equivalent, and they result in the same update rule in the EM iterations. Hence, they can be grouped together to apply the update once for all such fragments. This factorization, which is introduced by *IsoEM* [19], maintains full fidelity to information regarding fragment mappings to transcripts because all fragments in a group are identical.

A more popular approach for factorizing the likelihood is employed by *mmseq* [17] and also later in alignment-free tools like *Sailfish* [11], Salmon [13] and *kallisto* [14]. *mmseq* introduced a notion of fragment equivalence classes, which treats as equivalent any fragments that map to the same set of transcripts. Unlike *IsoEM*, the equivalence notion does not depend on the values of conditional probabilities. According to this definition, every set of fragments like $\mathcal{F}^q$ such that for all $f_i, f_j \in \mathcal{F}^q$, $\Omega(f_i) = \Omega(f_j) = \Omega(\mathcal{F}^q)$, form an equivalence class with the label $\Omega(\mathcal{F}^q)$. Define $N^q = |\mathcal{F}^q|$ to be the number of fragments in class $\mathcal{F}^q$. Then, the likelihood function based on these equivalence classes, can be approximated as:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{F}) \approx \prod_{[q] \in \boldsymbol{\mathcal{C}}} \left(\sum_{\langle i, t_i \rangle \in \Omega([q])} \Pr(t_i \mid \boldsymbol{\theta}) \cdot \Pr(f \mid [q], t_i) \right)^{N^q}, \tag{1.6}$$

where $\boldsymbol{\mathcal{C}}$ is the set of all equivalence classes, and $\Pr(f \mid [q], t_i)$ is the probability of generating a fragment f given that it comes from equivalence class $[q]$ and transcript t_i. The key to the efficiency of likelihood evaluation (or optimization) under this factorization, is that the probability $\Pr(f \mid [q], t_i)$ is assumed to be identical for each of the N^q fragments in each equivalence class $[q]$—hence, we do not subscript f in eq. (1.6). This allows one to replace the product over all fragments f_j in full model (eq. (1.1)) with a product over all equivalence classes in eq. (1.6). The approximation, of course, stems from the

fact that, under the full model, a fragment f_j may have a probability $\Pr(f_j \mid t_i)$ that is arbitrarily different from $\Pr(f \mid [q], t_i)$. Moreover, the most common approximations, like those adopted in *mmseq*, *Sailfish*, and *kallisto* consider this probability to be fixed and essentially independent of any fragment-level information (e.g., it is set to one divided by the effective length of t_i).

After applying any factorization which groups a set of fragments together in equivalence classes $\mathcal{F}^q$, the fragments in the EM iteration can be substituted with equivalence classes (groups) and each update would increase the α values based on the number of fragments in each equivalence class. The modified version of the E step in algorithm 1 is displayed in algorithm 2.

Algorithm 2: Modified E step after employing factorization

E-step:

for $\mathcal{F}^q \in \mathcal{C}$ **do**

  $sum = \sum_{t_k \in \Omega(\mathcal{F}^q)} \theta_k \times \Pr(f|\mathcal{F}^q, t_k)$

  for $t_j \in \Omega(\mathcal{F}^q)$ **do**

    $\alpha_j += \frac{\theta_k \times \Pr(f|\mathcal{F}^q, t_k)}{sum}$

  end

end

1.2.4 Online EM for optimizing the likelihood function

The conditional probability values are stored in memory during the EM iterations in order to avoid expensive I/O operations and re-computation in each EM round. If no factorization is used, for each existing mapping pair f_i and t_j, a value is stored. This

makes the memory requirement scale with the number of mappings. *eXpress* [16] attempts to bound the memory requirement by benefiting from an online-EM algorithm rather than a batch-EM. An online-EM consists of a single iteration over all fragments in the sample, updating α values once for each fragment. Fragments are not stored in memory after being observed, which makes *eXpress*'s memory requirement independent of the number of fragments in the sample. The large number of fragments in RNA-seq samples lets *eXpress* often achieve high quality abundance estimates with a single run over the data. *eXpress* requires the output of an alignment tool for mapping reads to transcripts to run the online phase. Again, here, the mapping information shall limit the number of updates performed for each fragment.

eXpress applies a modified version of online updates which prevents the algorithm from performing updates for each transcript in each iteration. The online update rules for each fragments are:

$$\alpha^{i+1} = \alpha^i + m_i \tilde{\tau}^i, \tag{1.7}$$

where:

$$\tilde{\tau}_t^i = \Pr\left(T = t | F = f_i\right), \tag{1.8}$$

and

$$m_{i+1} = m_i \times \frac{\gamma_{i+1}}{1 - \gamma_i} \times \frac{1}{\gamma_i}, \tag{1.9}$$

α^i is the optimized value after observing the first i fragments. Bayes' rule can be applied to obtain the probability in eq. (1.8) from the conditional probabilities. The value m_i is called forgetting mass and depends on the forgetting factor γ_i. The γ values are set

as $\gamma_i = \frac{1}{i^c}$ where $0.5 < c < 1.0$. After observing all N, fragments the relative counts of fragments from each transcript type can be obtained from the vector α^N.

1.2.5 Dual phase optimization

The inference algorithm in Salmon [13] consists of two phases. First, Salmon runs an online EM optimization to obtain high quality primary estimates of abundances. In this phase, Salmon is able to achieve a good estimation of fragment length distribution by examining many fragments as they are streamed in the online EM. Therefore Salmon can derive good estimates of conditional probabilities using the fragment length distribution and other information provided with mappings. The equivalence classes over sets of fragments are also created in the online phase. Salmon introduces the notion of rich equivalence classes by assigning a single scalar to each transcript in an equivalence class, by averaging the conditional probabilities of all fragments in the class to the transcript. This value is equal to $\frac{1}{|\Omega(F^q)|}$ in non-rich equivalence classes.

Salmon uses the estimates obtained in the online phase as a starting point for performing a batch EM algorithm in the second phase. This two-phase optimization allows Salmon to rich very high quality estimates compared to other existing quantification tools. The online phase of the Salmon enables deriving a new factorization of the likelihood function to be optimized in the batch EM phase, which does not discard any necessary information for accurate abundance estimation. The details of this factorization is discussed in chapter 3.

1.2.6 Metrics for evaluating quantification accuracy

The formula for calculating the metrics used for evaluating the abundance estimation results in the manuscript are as follows. The metrics are Mean Absolute Relative Difference (MARD), Mean Absolute Error (MAE), and Mean Squared Log Error (MSLE).

$$\begin{aligned} \text{MARD}(y, \hat{y}) &= \frac{1}{n_{\text{refs}}} \sum_{i=0}^{n_{\text{refs}}-1} \frac{|y_i - \hat{y}_i|}{y_i + \hat{y}_i}. \\ \text{MAE}(y, \hat{y}) &= \frac{1}{n_{\text{refs}}} \sum_{i=0}^{n_{\text{refs}}-1} |y_i - \hat{y}_i| \,. \\ \text{MSE}(y, \hat{y}) &= \frac{1}{n_{\text{refs}}} \sum_{i=0}^{n_{\text{refs}}-1} (y_i - \hat{y}_i)^2. \end{aligned} \tag{1.10}$$

All of these metrics compute the difference of the estimated abundances with the truth. In addition to these metrics, we also evaluate the correlation between the estimations and truth by computing the Spearman correlation. Spearman correlation is computed using the pandas library [20] in Python.

1.2.7 Inference of the Posterior Distribution of RNA-seq quantification

Estimating the inference uncertainty of the RNA-seq quantification is one of the crucial steps for many downstream analysis, e. g., finding the differentially expressed genes or transcripts, i. e., DE analysis. In fact, methods like Swish [21] directly use the inferential replicates created by RNA-seq quantification tools for finding the DE genes with a higher precision compared to other approaches.

There are two main approaches for sampling the posterior distribution for estimating

the uncertainty of quantification estimates; Gibbs sampling and Bootstrap sampling. The Gibbs sampling is a MCMC procedure which walks through the space that the EM explores for finding the maximum likelihood estimations of the $\mathcal{T}$ expressions. At the end of each iteration of the Gibbs sampling, the t_e xpression vector could be identified as a new inferential replicate for estimating the posterior distribution. To decrease between replicate correlations, the sampling could take place after every fixed number of iterations which is called the thinning factor for the sampling. Running the Gibbs sampling procedure for long enough could reach to the convergance of the posterior estimate, this will be only reached only after the Gibbs sampler has explored all the posterior samples. The number of iterations which is required for Gibbs sampling to converge usually depends on the properties of the sample, and as the number of $\mathcal{T}$ in the sample increases the convergance usually takes longer.

Bootstrap sampling is the other main approach for estimating the posterior distribution of the abundance estimations in RNA-seq. The bootstrap procedure [22] is a widely-used and computationally straightforward procedure for calculating measures of accuracy of an estimator. It works by resampling (with replacement) from the observed data, and treating these as population samples. The procedure has been used in many contexts for non-parametric estimation. In RNA-seq, Computing the abundance estimation of all Bootstrap sample leads to a estimating a posterior distribution for the abundances.

RNA-seq quantification tools ([23, 24] have implemented the regular bootstrap sampling by resampling the equivalent class counts. equivalent classes are a summerized representations of the reads, therefore, sampling the equivalent class counts instead of each individual read improves the efficiency of the Bootstrap sampling. positional Bootstrap

sampling is also another way of generating Bootstrap samples by sampling the positions where the reads map to on each transcript [25]. Furthermore, the RNA-seq quantification tool, Salmon [24] also includes a Gibbs sampling procedure for estimating the posterior distribution. Bitseq [26] applies a MCMC Gibbs sampler to generate samples from the posterior probabilty distribution.

Chapter 2: Improving accuracy and speed of mapping and alignment methods

In this chapter, I explore new ideas for improving the alignment and mapping methods. In the first part of the chapter, I address some of the existing obstacles with the lightweight alignment approaches. These methods are employed in the recent non-alignment based quantfication tools such as salmon [13], and kallito [14] to map the RNA-seq reads to the set of known reference sequences instead of aligning the reads. We show that the accuracy of the lightweight quantification methods decline in the difficult cases which often exist in the real samples. We introduce a new algorithm for selectively aligning RNA-seq reads to a transcriptome, with the goal of improving transcript-level quantification in adversarial scenarios. [1]

Furthermore, in the second part of this chapter, we build upon the idea of selective alignment to introduce Puffaligner [28], an all purpose aligner for short read sequencing reads which can be used in many cases ranging from alignment of RNA-seq, DNA-seq and metagenomic samples to known set of references. Puffaligner indexes the reference sequences using the Pufferfish index [10] which efficiently indexes the compacted De Bruijn graph built from the reference sequences.[2]

[1] This is a joint project with Hirak Sarkar and is presented in BCB 2017 [27]

[2] This is a joint project with Fatemeh Almodaresi and is published in Bioinformatics [28]

2.1 Towards Selective Alignment

While alignment is a staple of many genomic analyses, it sometimes represents more information than is actually necessary to address the analysis at hand. For example, recent tools like *Sailfish* [11], *RNAskim* [29], *kallisto* [14], Salmon [13], and *Fleximer* [30], demonstrate that accurate quantification estimates can be obtained without all of the information encoded in traditional alignments. By avoiding traditional alignment procedures, these tools are much faster than their alignment-based counterparts. Furthermore, by building the mapping phase of the analysis directly into the quantification task, they dispense with the need to write, store, and read, large intermediate alignment files. However, these non-alignment-based tools, while highly-efficient, have the disadvantage of potentially losing sensitivity or specificity in certain cases where alignment-based methods would perform well. For example, in the presence of paralogous genes, with high sequence similarity, there is an increased probability that the mapping strategies employed by such tools, and the efficient heuristics upon which they rely, will mis-map reads between the paralogs (or return a more ambiguous set of mapping locations than an aligner, which expends computational resources to verify the returned alignments) [31]. Similarly, in the case of *de novo* assemblies, poorly assembled contigs may have a larger number of mis-mapped reads due to lower sensitivity (here, the issue would be primarily due to aberrant exact matches masking the true origin of a read).

Other than suffering from spurious mappings, these fast non-alignment-based approaches can also miss true mappings of a read in rare cases where errors are positioned adversarially on the read. An obvious case of losing the true mapping is if a read contains no subsequence

of sufficient length from the true transcript. In another case, the true mapping of the read might be lost from the set of potential mapping loci due to the greedy nature of the mapping procedures. For some reads, multiple positions might be found on the same transcript where the read maps. In such cases, improved heuristics are required to address these challenges. In this paper, we present a novel algorithm, selective-alignment, that extends quasi-mapping to compute and store edit distance information where necessary. The reads for alignment are chosen based on certain criteria calculated during mapping. This strikes a balance between speed and accuracy; not compromising the superior speed of non-alignment-based algorithms, while also addressing some of the challenges mentioned above. Specifically, the motivation for selective-alignment is to enhance both the sensitivity and specificity of fast mapping algorithms by reducing or eliminating cases where spurious exact matches mask true mapping locations as well as cases where small exact matches support otherwise poor alignments. Selective-alignment algorithm is built atop the framework of *RapMap* [12], which uses an index that combines a fixed-length prefix hash table and an uncompressed suffix array [32]. We introduce a coverage-based consensus scheme to identify critical read candidates for which alignment is necessary.

Furthermore, we explore the challenging cases where the heuristics employed by fast mapping algorithms may fail to locate the correct locations for a read, while the traditional aligners do not. We do this by making a number of modifications to the underlying mapping algorithm to increase its sensitivity. We also introduce multiple filters and scoring schemes designed to eliminate spurious mappings (i.e., situations where the best mapping is unlikely to represent the true origin of the read). In this work, we focus on the effect of selective-alignment on transcript quantification estimates, and we leave a

thorough evaluation of the alignment qualities themselves as future work. In particular, the evaluation of alignment qualities is considerably complicated due to prevalent multi-mapping in the transcriptome.

2.1.1 Main characteristics of selective-alignment

The process of selective-alignment builds upon the basic data structures of Srivastava et al. [12], but there are a number of important algorithmic distinctions. Specifically, compared to the algorithm of RapMap, selective-alignment introduces the k-safe longest common prefix (k-safe-LCP), replaces maximum mappable prefixes (MMP) with maximum mappable safe prefixes (MMSP), increases mapping sensitivity by adopting a different consensus rule over hits, makes use of co-mapping to filter and prioritize potential mapping loci, introduces a new mechanism for selecting a mapping position for a read when multiple candidates exist on the same transcript, and, finally, introduces a fast edit distance filter (with alignment sub-problem caching) to remove spurious mappings and provide quality scores for mappings that pass the filter. A block diagram of different steps used in the selective alignment pipeline is shown in fig. 2.2.

Below, we recapitulate the basic data structures and concepts that will be required to explain the selective alignment algorithm. To start with, the index built on the transcriptome in selective-alignment is a combination of a suffix array and a hash table constructed from unique k-mers (substrings of length k) and suffix array intervals. The suffix array of a sequence, T— denoted $SA(T)$—is an array of starting positions of all suffixes from T in the original sequence. The values in the array are sorted lexicographically by the suffixes

they represent. Therefore, all suffixes starting with the same prefix are located in adjacent positions of the suffix array. Formally, given a suffix array, $SA(T) = \Lambda$, constructed from the transcriptome sequence, T, we construct a hash table, h, that maps each k-mer , κ, to a suffix array interval, $\mathtt{I}(\kappa) = [b, e)$, if and only if all the suffixes within interval $[b, e)$ contain the k-mer κ as a prefix. We define $\Lambda[i]$, for every $0 \leq i \leq |\Lambda|$, to be the suffix $T[SA[i]]$ (i.e., the suffix of T starting from position $SA[i]$). In the selective-alignment index, in addition to suffix array intervals, we store two extra pieces of information for each interval; the longest common prefix (LCP) and the k-safe-LCP corresponding to the interval. The longest common prefix (LCP) of any pair of suffixes in the suffix array is simply the length of the prefix that these suffixes share. Though the LCPs for the suffixes in the suffix array can be pre-computed, we instead compute them on demand using a linear scan. These methods are detailed below. As an alternate to the suffix array and the LCP array, one could make use of other data structures which also encode this information. For example, the recently-introduced method, Fleximer [30] makes use of the suffix tree for selecting informative sig-mers [29] from the transcriptome, and matching reads against them.

2.1.2 Defining and computing k-safe-LCPs

Here, we formally define the concept of k-safe-LCPs (see figure fig. 2.1). The determination of k-safe-LCPs starts by labeling each suffix array interval with the length of its corresponding longest common prefix and the associated transcript set it represents. Formally, $\mathtt{LCP}(\Lambda[b], \Lambda[e-1])$ for an interval $[b, e)$ is the length of the common prefix

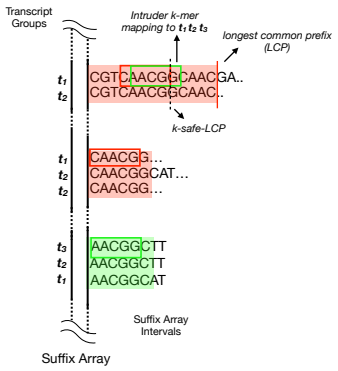


Figure 2.1: Calculation of k-safe-LCP from the suffix array data structure. The transcripts present in each suffix array interval determine the relevant transcript sets, and which k-mers will be considered as intruders. To determine the k-safe-LCP of the suffix array interval starting with the k-mer $CGTCA$, we check all the k-mers sequentially. Some k-mers do not yield an interval with transcripts other than t_1 and t_2, e.g., $CAACG$. Detection of a k-mer ($AACGG$) (as intruder) that maps to suffix array interval labeled (t_1, t_2, t_3) determines the k-safe-LCP here.

of the suffixes $\Lambda[b]$ and $\Lambda[e-1]$. Given k-mer κ, where $\kappa \in \boldsymbol{\mathcal{K}}$ and $\boldsymbol{\mathcal{K}}$ is the set of all k-mers from the reference sequence T, and the related interval $\mathtt{I}(\kappa) = [b, e)$, for all $p \in [b, e)$, we consider each transcript t such that the suffix $\Lambda[p]$ starts in transcript t in the concatenated text. Then, for this interval, we can construct a set $\mathcal{C}^{\kappa} = \{t_i, t_j, \dots\}$, which denotes the set of distinct transcripts that appear in the suffix array interval, indicated by κ. We note that this notion discards duplicate appearances of the same transcript in this interval.

We compute the k-safe-LCP for an interval indicated by k-mer κ_i iteratively. The initial length for the k-safe-LCP of the interval is k, length of a k-mer . We check, sequentially, each of the k-mers in the longest common prefix of the interval. For each new k-mer , the k-safe-LCP is increased by one character. We terminate the k-safe-LCP extension if any of the following conditions is encountered: (1) we reach the last k-mer contained in the LCP of this interval, (2) we encounter a k-mer κ_j such that

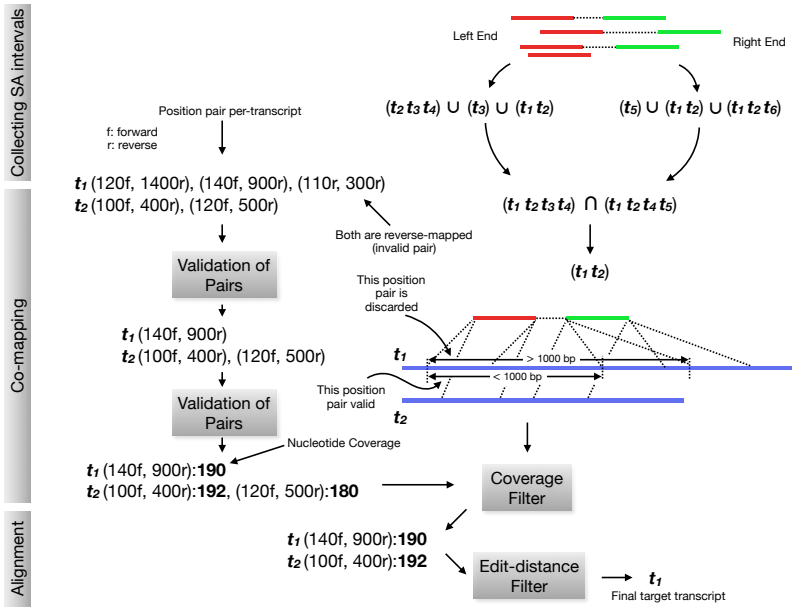


Figure 2.2: The three main steps of the selective-alignment process are demonstrated here. First, suffix array "hits" are collected. Then, in co-mapping, spurious mappings are removed by the orientation filter and then distance filter. At most a single locus per-transcript is selected based on the coverage filter. Finally, an edit-distance-based filter is used to select the valid target transcripts.

$\mathcal{C}^{\kappa_j} \not\subseteq \mathcal{C}^{\kappa_i}$ or (3) we encounter a k-mer κ_j such that the reverse complement of κ_j appears elsewhere in the transcriptome. When we encounter case (2) or (3), we call the k-mer κ_j an *intruder*. That is, the k-mer will potentially alter our belief about the set of potential transcripts to which a sequence containing this k-mer maps (by strictly expanding this set), or the orientation with which it maps to the transcriptome. We denote the k-safe-LCP of a particular interval $\texttt{I}\left(\kappa_i\right)$ as $\texttt{k-safe-LCP}(\texttt{I}\left(\kappa_i\right))$.

As shown in figure fig. 2.1, the k-safe-LCP determination for the top suffix array interval starts with matching k-mers within the longest common prefix. The k-mer "CAACG" maps to a suffix array interval labeled with (t_1, t_2). The next k-mer "AACGG", on the other hand, maps to a suffix array interval (shaded in green) labeled with (t_1, t_2, t_3), thereby implying the k-safe-LCP, shown as a dotted line. For each k-mer in the hash

table, we store the length of the LCP and k-safe-LCP, along with the corresponding suffix array interval.

2.1.3 Discovering relevant suffix array intervals

As shown in figure fig. 2.2, the selective-alignment approach can be broken into three major steps: collecting suffix array intervals, co-mapping, and selecting the high quality mappings. Gathering the suffix array intervals for a query read closely follows the *quasi-mapping* approach. It involves iterating over the read from left to right and repeating two steps. First, hashing a k-mer from the read sequence and then discovering the corresponding suffix array intervals. The process of k-mer lookup is aided by the k-safe-LCP stored in the index (discussed in section 2.1.2). The inbuilt lexicographic ordering of the suffixes in the suffix array, and the computed k-safe-LCP values of intervals enable safely extending k-mers to longer matches without the possibility of masking potentially-informative substring matches. Given a matching k-mer , κ_r, from the read sequence r, we extend the match to find the longest substring of the read that matches within $\texttt{k-safe-LCP}(\texttt{I}(\kappa_r))$. The matched substring can be regarded as maximum mappable prefix (MMP) [7], that resides within the established k-safe-LCP. We call this a maximal mappable safe prefix (MMSP — eliding k where implied). For a k-mer , κ_r, and interval, $[b, e)$, we note that $\texttt{k-safe-LCP}(\texttt{I}(\kappa_r)) \geq \ell_{\text{MMSP}_{\kappa_r}}$, where $\ell_{\text{MMSP}_{\kappa_r}}$ is the length of MMSP_{κ_r}, the MMSP between the read's suffix starting with κ_r and the interval $\texttt{I}(\kappa_r)$. The next k-mer lookup starts from the $(\text{MMSP}_{\kappa_r} - k + 1)$-th position. By restricting our match extensions to reside within the MMSP, we ensure that we will

not neglect to query any k-mer that might *expand* the set of potential transcripts where our read may map. We note here both the theoretical and practical relation between the MMSP matching procedure, and the concept of a uni-MEM, as introduced by Liu et al. [8]. The k-safe-LCP for suffix array intervals are closely related to the lengths of unipaths in the reference de Bruijn graph of order k. Thus, our procedure for finding MMSPs, that limits match extension by the k-safe-LCP, is similar to the uni-MEM seed generation procedure described in debga [8], with the distinction that in our method, we only consider extending seeds in one direction, and that we also choose not to terminate the k-safe-LCP when the set of implied reference transcripts corresponding to the interval decreases in cardinality.

Given all the suffix array intervals collected for a read end (i.e. one end of a paired-end read), we take the *union* of all the transcripts they encode. Formally, if a read r maps to suffix array intervals labeled with $\mathcal{C}^{r_1}, \ldots, \mathcal{C}^{r_n}$, then we consider all transcripts in the set $\mathcal{C}^{r_1} \cup \mathcal{C}^{r_2} \cup \ldots \cup \mathcal{C}^{r_n}$, and the associated positions implied by the suffix array intervals. As shown in fig. 2.2; this step is done before co-mapping.

We adopt a heuristic to avoid excessive k-mer lookups when we encounter a mismatch. When extension of an MMP is no longer possible, it is most probable that the mismatch results from an error in the read. If the mismatch is due to the presence of an error, then checking each k-mer overlapping this error can be a costly process. Instead, we move forward by a distance of $k/2$ in the read, and check the k-mer from the read such that the mismatch occurs in the middle position. If this k-mer lookup leads to another suffix array interval, we continue with the MMP extension process there; otherwise, we move again to the first k-mer that does not overlap this mismatch position. We observe that, in

practice, the k-safe-LCP, and hence the MMSP lengths can be quite large (Figure 2.3).

2.1.4 Co-Mapping

After collecting the suffix array intervals corresponding to left and right ends of the read, we wish to exploit the paired-end information in determining which potential mapping locations might be valid. Hence, from this step onward, we use the joint information for determining the position and target transcripts. Given the suffix array intervals for individual ends of a paired-end read, the problem of aligning both ends poses a few challenges. First, a single read can map to multiple transcripts, and we wish to report all equally-best loci. Second, there can be multiple hits from a read on a single transcript (e.g., if a transcript contains repetitive sequence), and extra care must be taken to determine the correct mapping location. Finally, there may be hits that do not yield high-quality alignments (i.e. long exact matches that are nonetheless spurious). To address the first and third points, we employ an edit distance filter to discard spurious and sub-optimal alignments. To address the second challenge, we devise a consensus strategy to choose at most one unique position from each transcript.

Before applying the above mentioned strategy, we remove transcripts that do not contain hits from both the left and right ends of the read. Formally, given two ends of a read r, r^{e_1} and r^{e_2}, and the corresponding suffix array intervals labeled with $\mathcal{C}^{r_1^{e_1}}, \ldots, \mathcal{C}^{r_n^{e_1}}$ and $\mathcal{C}^{r_1^{e_2}}, \ldots, \mathcal{C}^{r_m^{e_2}}$ respectively, we only consider transcripts present in the set $(\mathcal{C}^{r_1^{e_1}} \cup \ldots \cup \mathcal{C}^{r_n^{e_1}}) \cap (\mathcal{C}^{r_1^{e_2}} \cup \ldots \cup \mathcal{C}^{r_m^{e_2}})$. We further refine this set by checking the validity of the alignments these hits might support. Currently, we use two validity checks illustrated

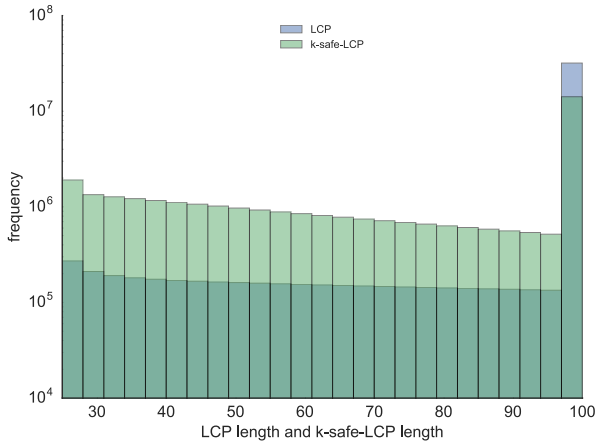


Figure 2.3: The distribution of k-safe-LCP lengths and LCP lengths are similar and tend to be large in practice (human transcriptome). Here, we truncate all lengths to a maximum value of 100 (so that any LCP or k-safe-LCP longer than 100 nucleotides is placed in the length 100 bin).

in fig. 2.2. First, we apply an orientation-based check, and second, we employ a distance-based check. The orientation check removes potential mappings which have an orientation inconsistent with the underlying sequencing library type (e.g., both ends of a read mapping in the same orientation). The distance check removes potential alignments where the implied distance between the read ends is larger than a given, user-defined threshold ($1,000$ nucleotides by default).

Coverage based consensus: In selective-alignment, the potential positions on a transcript are scored by their individual coverage on the target transcript. Figure 2.4 depicts the mechanism of choosing the best postion on a transcript from multiple probable mappings to the same transcript. The coverage mechanism employed in selective-alignment makes use of the MMSP lengths collected during a prior step of the algorithm rather than simply counting k-mers . In fig. 2.4, the transcript t_2 has two potential mapping positions given the reads: position 10 and 20. The coverage consensus mechanism selects position 20

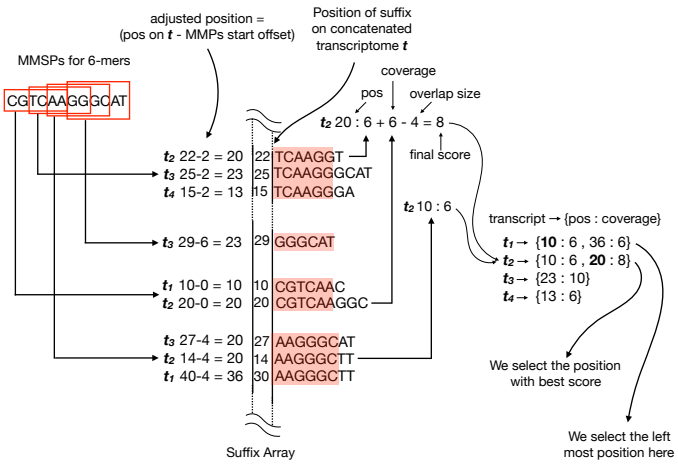


Figure 2.4: The MMSPs corresponding to a read are derived from multiple suffix array intervals. Here, all MMSPs happen to be of length k as LCPs are of size k. The coverage scheme finds out the exact positions on each transcript by adjusting the starting position of the MMSPs. The total score takes into account the positions where matches overlap. The final position is chosen by selecting the locus with maximum coverage.

over position 10 due to the higher coverage by tiling MMSPs on the read.

2.1.5 Selecting the best candidate transcripts

Once the positional ambiguity within a transcript is resolved, the next step is selecting the best candidate transcripts from a set of mappings. Since mapping relies on finding exact matches, the length of the matched subsequence between the read and reference can sometimes be misguiding when comparing different candidate transcripts. That is, the transcripts with the longest exact matches do not always su A block diagram of the steps described below are depicted in Figure pport optimal alignments for a read. At this point in our procedure, we follow the approach taken by many conventional aligners, and use an existing optimal alignment algorithm to compute the edit distance, by which we select the best candidate transcripts.

When performing alignment, we assume that a given read aligns starting at the

position computed in the previous steps. This helps us to reduce the search space within the transcript where we must consider aligning the read, and thereby considerably reduces the cost of alignment. To align the read at a specific position on the transcript and calculate the edit distance between them, we use $Myer's$ bounded edit distance bit-vector algorithm [33], as implemented in `edlib` [34]. For a fixed maximum allowable edit distance, this algorithm is linear in the length of the read. We note that the bounded edit distance algorithm we employ will automatically terminate an alignment when the required edit distance bound is not achievable.

We remove all alignments with edit distance greater than a user-provided threshold. This is similar to the approach used by many existing aligners, and allows us to specify that even the best mapping for a given read may have too many edits to believe that it reasonably originated from a known transcript in the index. An appropriate threshold should be based on the expected error rate of the instrument generating the sequenced reads, and a very low threshold can lead to a decreased mapping rate.

2.1.6 Enhancement of quantification accuracy based on edit distance

We investigated the effect of incorporating edit distance in downstream quantification. Since we integrated the selective-alignment scheme into the quantification tool Salmon [13], the edit distance scores from selective-alignment can be used as a new parameter to Salmon's inference algorithm.

In the framework of abundance estimation, we define the conditional probability of a generating a particular fragment, f_j, given that it comes from a specific transcript,

t_i, as $P(f_j \mid t_i)$. Given the edit distance between the fragment and the transcript, we can incorporate this parameter into this conditional probability. *Soft* filtering introduces a new term in the conditional probability based on $d_{i,j}$, which is the sum of the edit distances between the read ends of fragment f_j and transcript t_i. We set this probability according to an exponential function, $P(a_j|f_j, t_i) = e^{-4d_{i,j}}$. The aggregate of threshold filtering and *soft* filtering can be described as follows:

$$\Pr\left(a_j \mid d_{i,j}, t_i\right) = \begin{cases} 0 & d_{i,j} > threshold \\ e^{-4d_{i,j}} & d_{i,j} \leq threshold \end{cases}. \tag{2.1}$$

Preventing redundant alignments by exploiting shared LCPs: Exploiting the common subsequences in the transcriptome is instrumental to the superior speed of fast mapping, non-alignment-based tools. Reads generated from exonic sequences common to multiple transcripts from the same gene or paralogous genes are the main source of ambiguous mappings. As we rely on the suffix array data structure to obtain the initial set of transcripts to which a read maps, there are cases where exactly identical reference sequences all act as mapping targets for the read. For a suffix array interval $[b, e)$, we identify such common subsequences by examining the *longest common prefix* (LCP) of the interval. If the length of the LCP is equal or greater than the length of the read, then the actual alignment against the underlying reference at these positions will be identical. We observed that for almost half of the read-transcript pairs, the alignment process can be avoided. Note that if the read sequence shares a complete match with the common prefix, meaning that maximum

mappable safe prefix length is equal to read length (i.e., the read matches the reference exactly at some set of positions), we can also bypass the Meyer's edit distance algorithm call completely.

Avoids redundant work by caching alignment sub-problems further: We also extend a similar idea to the scenario where only part of the reference sequence is shared between references. Specifically, when performing an alignment between anchoring exact matches, we store the result in a hash table where the key is a tuple $(i, j, h\,(i', j'))$ and the associated value is the computed edit distance. Here, i and j denote the start and end of the read interval being aligned and i' and j' denote the start and end of the reference sequence; $h(i', j')$ is a hash of the corresponding reference sequence (we use `xxhash` [35]). This allows us to detect when a redundant alignment sub-problem for a read is shared between references, and to reuse the cached result in such cases.

2.1.7 Evaluating the performance of selective-alignment

To evaluate the effectiveness of selective-alignment, we coupled it with the quantification tool Salmon (branching from the v0.9.1 release). This enables us to measure the effect of different alignment based and non-alignment based algorithms on transcript-level quantification results directly, holding the statistical estimation procedure fixed. We also include *kallisto* (v0.43) in our benchmarks, which provides a perspective on pseudoalignment-based quantification. Furthermore, we compare the performance of selective-alignment with the recent, fast, hashing and alignemnt-based, abundance estimation tool (currently un-published) *Hera* [3].

[3] `https://github.com/bioturing/hera`

We note, this is an early version of the *Hera* (v1.2) software, which is already performing very well in our testing, but is subject to changes and improvements. Given it's impressive performance (in both time and accuracy), we decided to include *Hera* in our comparisons with the consent of its authors (personal communications). We measure the Spearman correlation and Mean Absolute Relative Differences (MARD) of read counts as performance metrics when comparing the different methods (further metrics are also provided for some of the experiments in the supplementary material). All experiments were performed on an Intel(R) Xeon(R) CPU (E5-2699 v4 @2.20GHz with 44 cores and 56MB L3 cache) with 512GB RAM and a 4TB TOSHIBA MG03ACA4 ATA HDD running ubuntu 16.10 and each method was run using 16 threads.

In all our experiments, reads are mapped to the transcriptome using using *Bowtie2*, *kallisto*, *Hera*, selective-alignment and *STAR*. Subsequently, transcripts are quantified by Salmon (v0.9.1) using the relevant mappings (from alignment or the non-alignment-based methods) as input (except in the cases of *kallisto* (0.43) and *Hera* (1.2), which include implementations of their quantification algorithms). The alignment mode of Salmon enables us to use *STAR* (v2.5) and *Bowtie2* (v2.3) output as a direct input to the quantification module — thereby reducing variability due to differences in the underlying methodology used for quantification. To achieve the most sensitive alignment, *Bowtie2* is run with the alignment options suggested for use with *RSEM* [36]. For aligning reads to the transcriptome using *STAR*, we used the same options described in [12]. When processing alignments, Salmon was run with `--rangeFactorizationBins 4` [37] and `--useErrorModel`. With selective-alignment, Salmon was run using the `--softFilter` flag (discussed in section 2.1.6), a range factorization value of 4 and an edit distance

threshold of 7. *kallisto* was run with default parameters. Both the selective-alignment and *kallisto* indices were built with $k = 25$; *Hera* does not include k-mer size as a user-defined parameter.

2.1.8 Quantification of simulated reads against mutated transcriptomes

We explored the performance of different alignment-based and alignment-free methods by quantifying simulated short RNA-seq reads against mutated reference sequences. The simulation process consists of two steps. In the first step, we mapped an experimental RNA-seq sample (accession number SRR5638585) to the human transcriptome (Ensembl release 80 [38]) using Salmon. The resulting abundance vector, in conjunction with the full transcriptome sequence generated from the full human genome and the corresponding annotations (version GRCh37.p13), is used to simulate five batches of 100bp paired-end RNA-seq samples, where each batch contains $\sim$ 47M reads. We used the sequence simulator Polyester [39] for generating the read datasets.

While the simulated dataset enables comparison with the ground truth, the quality of the reads is high and does not show the subtle nuances that arise when mapping reads from experimental sequencing datasets. In reality, the sequenced reads could differ from the annotated reference sequence due the presence of mutations (variants) in the sequenced organism. In other cases, a reference sequence from one species could be used to analyze data from a phylogenetically closely related species, for which an annotated reference in unavailable. Therefore, to recapitulate these adversarial situations, instead of mapping the simulated reads to the exact underlying transcriptome used for read generation, we

map them against references mutated at a controllable rate.

The mutated version of the transcriptome is derived from the underlying reference genome that was subject to random mutations. The nucleotides of the reference genome were randomly altered based on a Poisson process with a tunable *rate* parameter. The rate parameter enables controling the rate of mutation that we want to introduce in the reference genome. For the current manuscript we have used 5 equally spaced rate parameters from 0.01 to 0.05. The mutated genome sequences and the original annotation are used to generate the mutated reference transcriptomes. As the resulting transcriptomes contain devations from the indexed reference, we believe that mapping to these references will capture some aspects of the difficulties encountered when applying such tools to certain experimental datasets.

To evaluate the performance we have measured the quantification accuracy of different tools with respect to the ground truth provided to Polyester. As explained earlier, tools such as *kallisto*, *Hera* and selective-alignment have a quantification pipeline attached to the mapping module and are, therefore, capable of generating abundance vectors directly. On the other hand, *Bowtie2* and *STAR* generate alignment files that we have coupled with Salmon (run in alignment-based mode) to obtain abundance estimates.

Performance of the various methods on a simulated sample is shown in table 2.2. In this case, the simulated sample is mapped against 5 different mutated transcriptomes with increasing error rates and the corresponding spearman correlation and MARD values calculated using the ground truth. As shown in table 2.2, the correlation between quantification estimates using selective-alignment and the ground truth is higher than the other self-contained quantification methods, *kallisto* and *Hera*. This gap between correlation values

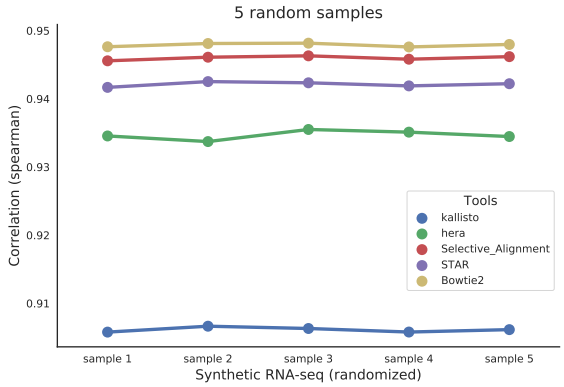


Figure 2.5: Performance variation of different tools on paired end reads produced with five random seeds.

increases as the rate of mutation in the reference transcriptome is increased, showing the ability of selective-alignment to accurately map reads against diverging transcriptomes. The MARD values for selective-alignment are lower in comparison with other non-alignment-based methods as well.

To measure the variation in quantification about a single random instance of simulated data (i.e., data generated with a particular random seed), we have also generated five different simulated RNA-seq datasets by passing different seeds to Polyester. To minimize external variation, we used the least mutated transcriptome (rate 0.01) as reference. By plotting the spearman correlations, as shown in fig. 2.5, we observe that, given that all the tools perform well on the random samples, the performance of selective-alignment is grouped with the alignment-based methods, such as *Bowtie2* and *STAR*. Further, the variation in quantification performance of all methods (i.e. the standared error) across these different simulated replicates is very small.

Mutation Rate	Kallisto	Hera	Selective Alignment	STAR-Salmon	Bowtie2-Salmon
0.01	0.906	0.935	**0.946**	0.942	**0.948**
0.02	0.871	0.925	**0.942**	0.939	**0.945**
0.03	0.844	0.910	**0.935**	0.933	**0.942**
0.04	0.817	0.880	**0.925**	**0.925**	**0.937**
0.05	0.793	0.845	0.904	0.909	**0.927**

Table 2.1: Synthetic dataset quantified against the mutated reference transcriptome with different mutation rates. The spearman correlation is calculated with respect to the ground truth.

Mutation Rate	Kallisto	Hera	Selective Alignment	STAR-Salmon	Bowtie2-Salmon
0.01	0.161	0.116	**0.100**	0.104	**0.096**
0.02	0.193	0.132	0.108	0.109	**0.100**
0.03	0.215	0.172	0.120	0.115	**0.107**
0.04	0.236	0.231	0.143	0.127	**0.118**
0.05	0.257	0.291	0.186	0.150	**0.142**

Table 2.2: Synthetic dataset quantified against the mutated reference transcriptome with different mutation rates. The MARD (mean absolute relative difference) is calculated with respect to the ground truth.

2.1.9 Experimental reads from human transcriptome

We have also benchmarked our proposed selective-alignment method on experimental data from SEQC(MAQC-III) consortium [40] samples (SRA accession `SRR1215996` - `SRR1216000`). Each of the five technical replicates consists of $\sim$11M, 100bp, paired-end reads, sequenced on an Illumina Hiseq 2000 platform.

We follow the same basic assessment methodology as discussed in section 2.1.8, and report the mean Spearman correlation and MARD value for each method. However, we note that, since this is experimentally-derived data, there is no knowledge of ground truth transcript abundances. Instead, we have measured the overall concordance between different approaches. Given the results obtained in all of our other testing, we expect the *Bowtie2*-based pipeline to be the most accurate, so we are generally looking for high

concordance with those quantifiaction estimates.

In table 2.3, we compare the quantification results produced by different methods. Each individual cell contains the average obtained across the five samples. High Spearman correlation and low MARD value between *Bowtie2* and selective-alignment show that selective-alignment produces results most similar to those based on *Bowtie2*. Interestingly, the concordance between the selective-alignment and *Bowtie2*-based pipelines is even higher than the concordance between the two pipelines based on more traditional alignment approaches (i.e. *Bowtie2* and STAR). While we cannot assess the accuracy with respect to known ground truth on these samples, we nonetheless believe assessments based on real data like this are important to perform, as the complexity of experimental data seems to be considerably higher than that of simulated data and its characteristics can be markedly different. Finally, Table 2.4 provides timing and memory assessments of all the methods running on sample `SRR1215996`. Since the mapping phase of selective-alignment is not distinct from the quantification phase, the memory and time footprints include the mapping part of the pipeline. Further, disk space is not comparable to alignment-based methods, since alignment files are not written directly as output of selective-alignment (rather, the selective-alignment algorithm informs the mappings and provides edit-distance-based scores — as described in eq. (2.1) — directly to the quantification algorithm).

2.1.10 Conclusion

Recently, fast non-alignment-based approaches have been developed for mapping RNA-seq reads to transcriptomes. Rather than generating full alignments, these approaches

Method	*kallisto*	*Hera*	selective	*STAR*	*Bowtie2*
kallisto	0.000 / 1.000	0.189	0.160	0.153	0.168
Hera	0.868	0.000 / 1.000	0.135	0.147	0.138
selective	0.898	0.902	0.000 / 1.000	0.129	**0.059**
STAR	0.898	0.896	0.913	0.000 / 1.000	0.129
Bowtie2	0.890	0.901	**0.966**	0.913	0.000 / 1.000

Table 2.3: The Spearman correlation and MARDS between transcript abundances computed by all methods on experimental data. Each number is the mean on 5 different samples; the numbers in the lower left triangle of the matrix are the Spearman correlations and the ones in upper right are the MARD values. "selective " refers to selective-alignment.

Method	time (s)	memory (KB)
kallisto	61.000	4,006,284.000
Hera	38.000	6,736,576.000
selective-alignment	65.000	7,994,324.000
STAR	398+96	max(8,342,444.000,5,513,432.000)
Bowtie2	977+125	max(1,020,032.000,9,949,380.000)

Table 2.4: Comparison of timing and memory foot-print of selective-alignment with other alignment and non-alignment methods on experimental sample SRR1215996. The timing performance for *STAR* and *Bowtie2* is the sum of mapping and quantification (with salmon) steps (first number is the mapping step) and memory footprint is the max memory footprint of these two steps (first number is for the mapping step).

compute "mapping" information that is often sufficient for a number of given analysis tasks (e.g., transcript quantification [11, 13, 14, 29, 30] or metagenomic abundance estimation [41]). Yet, there exist scenarios where such non-alignment-based approaches can go awry; either failing, by the greedy nature of their procedures, to find the true target of origin of a read, or by allowing spurious mappings to targets supported by exact matches that would nonetheless fail reasonable alignment scoring filters. Moreover, it is sometimes desirable to be able to produce, on demand, the edit distance or alignment that would

result from a given mapping location. The recently-introduced *Hera* validates mapping quality using alignment, which resolves spurious mappings, though it still suffers a loss of sensitivity compared to traditional alignment methods, and fails to process *de novo* assembled transcriptomes. In this paper, we introduce a selective alignment algorithm that attempts to bridge the gap between these non-alignment-based algorithms and more traditional alignment approaches. Selective-alignment improves upon both the sensitivity and specificity of these non-alignment-based algorithms while making very moderate concessions with respect to the computational budget. To achieve this level of efficiency, a number of algorithmic innovations were required, some of which may be of general interest. In the future, we hope to expand upon the notion of selective alignment even further, both by improving the algorithm and implementation, and by exploring use cases where selective alignment applies. Such situations are those where fast non-alignment-based approaches are inappropriate and traditional alignment approaches are too slow. In terms of improving the method, we hope to add functionality to automatically predict the optimal edit distance threshold in the read mappings based on the quality of the alignments, and for selective-alignment to self-tune to properly handle edge cases, such as soft clipping. The selective-alignment algorithm currently implements user specified edit distance threshold for filtering spurious reads. A more data-driven choice of filter can lead to a more resilient threshold that can perform gracefully while handling both adversarial reads as well as high-quality reads in heterogeneous read samples. In high quality samples, the edit distance bound can be set lower to further speed-up the algorithm. Future work will also include support for reporting the actual CIGAR strings for applications that require this information, such as RNA-seq based variant calling or allele identification.

2.2 Puffaligner: An efficient and accurate aligner based on the pufferfish index

Short-read aligners are a major workhorse of modern genomics. Given the importance of the alignment problem, a tremendous number of different tools have been developed to tackle this problem. Some widely used examples are BWA [42], *Bowtie2* [5], *HISAT 2* [43, 44] and STAR [45]. Existing alignment tools use a variety of indexing methods. Some tools, such as BWA, *Bowtie2*, and STAR use a full-text index over the reference sequences; BWA and *Bowtie2* use variants of the FM-index, while STAR uses a suffix array.

A popular alternative approach to full-text indices is to instead, index sub-strings of length k (k-mers) from the reference sequence. Trading off index size for potential sensitivity, such indices can either index all of the k-mers present in the underlying reference, or some uniform or intelligently-chosen sampling of k-mers . There are a large variety of k-mer -based aligners, including tools like the Subread aligner [46], SHRiMP2 [47], mrfast [48], and mrsfast [49]. To reduce the index size, one can choose to select specific k-mers based on a winnowing (or minimizer) scheme. This approach has been particularly common in tools designed for long-read sequence alignment like mashmap [50] and minimap2 [9].

Recently, a set of new indices for storing k-mers have been proposed based on graphs, specifically de Bruijn graphs (dBg). A de Bruijn graph is a graph over a set of distinct k-mers where each edge connects two neighboring k-mers that appear consequently

in a reference sequence and therefore, overlap on "$k-1$" bases. *kallisto* [23], deBGA [8], BGreat [51], BrownieAligner [52], and Pufferfish [10] are some tools which use an index constructed over the de Bruijn graph built from the reference sequences. Cortex [53], Vari [54], rainbowfish [55], and mantis [56] are also tools that use a colored compacted de Bruijn graph for building their index over a set of raw experiments. All these approaches cover a wide range of the possible design space, and different design decisions yield different performance tradeoffs.

Generally, the fastest aligners (like STAR) have very large memory requirements for indexing, and make some sacrifices in sensitivity to obtain their speed. On the other hand, the most sensitive aligners (like *Bowtie2*) have very moderate memory requirements, but obtain their sensitivity at the cost of a higher runtime. Maintaining the balance between time and memory is especially more critical while aligning to a large set of references, like a large collection of microbial and viral genomes which may be used as an index in microbiome or metagenomic studies. As both the collection of reference genomes and the amount of sequencing data grows quickly, it is import for alignment tools to achieve a time-space balance without loosing sensitivity.

Based on the compact Pufferfish [10] index, we introduce a new aligner called PuffAligner, that we believe strikes an interesting and useful balance in this design space. PuffAligner is designed to be a highly-sensitive alignment tool while, simultaneously, placing a premium on computational overhead. By using the colored compacted de Bruijn graph to factor out repeated sub-sequences in the reference, it is able to leverage the speed and cache friendliness of hash-table based aligners while still controlling the growth in the size of the index; especially in the context of redundant reference sequences.

By carefully exploring the alignment challenges that arise in different assays, including single-organism DNA-seq, RNA-seq alignment to the transcriptome, and metagenomic sequencing, we have engineered a versatile tool that strikes desirable balance between accuracy, memory requirements and speed. We compare PuffAligner to some other popular aligners and show how it navigates these different tradeoffs. PuffAligner is a free and open-source software and it is implemented in C++14 and can be obtained from `https://github.com/COMBINE-lab/pufferfish/tree/cigar-strings`.

2.2.1 Main pipeline in PuffAligner

PuffAligner is an aligner built on top of the Pufferfish indexing data structure. Pufferfish is a space-efficient and fast index for the colored compacted de Bruijn graph (ccdBg). A colored compacted de Bruijn graph is a graph whose vertices (strings) are the compacted non-branching paths of the underlying de Bruijn graph, with the restriction that each node also have the same color set (set of reference sequences in which it appears). The nodes in the colored compacted de Bruijn graph are referred to as unitigs. Each unitig can be mapped to a list of <reference ID, position, orientation> tuples that describe exactly how this subsequence appears in the unlderying collection of references. The basic query operation in the Pufferfish index is to query a k-mer from the input sequence against the index. Given this query, the pufferfish index returns the unique position (and orientation) where this k-mer appears in the colored compacted de Bruijn graph (or a sentinel value if this k-mer does not occur). This match between the query and the graph can then be easily "unpacked" into the implied list of matches with the

underlying references by finding all of the places that the matched unitig appears in the reference sequences and translating the relative position within the unitig into the corresponding reference position (and adjusting the orientation if necessary). The output of this step is then a list of all of the reference sequences, positions, and orientations where this exact match occurs. While k-mer query is the basic operation performed by the index, we actually do not use k-mer matches directly, and instead extend the initial match into unique maximal exact matches (uni-MEMs).

Specficially, each k-mer match is extended simultaneously in both the query and reference to obtain a longer exact match. The exact matches to the unitigs, called uni-MEMs, are then projected to the positions on the references associated to that unitig. Then, uni-MEMs are aggregated into MEMs (described below) on each reference, and the chains of MEMs with the highest score are selected. In the case of paired-end reads, the chains of the left and right ends are paired with respect to their distance, orientation, etc. Finally, rather than fully aligning each query sequence to the anchored position on the reference, only the sub-sequences from the query that are not part of the uni-MEMs (exact matches) are aligned to the reference; we call this procedure the between-MEM alignment. Each of these steps are explained in detail in the following sections.

2.2.2 Exact matching in the Pufferfish index

The pufferfish index provides PuffAligner with an efficient index for k-mer lookup within a list of references. Specifically, the core components of the index are (1) a minimal perfect hash function (MPHF), (2) a unitig sequence vector, (3) a unitig-to-reference table,

and (4) a vector storing the position associated with each k-mer in the unitig sequence vector. The unitig sequence vector contains all the unitigs in the ccdBg. The Pufferfish index admits efficient exact search for k-mers , as well as longer matches that are unique in both the query string and colored compacted de Bruijn graph. These matches, called uni-MEM, were originally defined in deBGA [8]. A uni-MEM is a Maximal Exact Match (MEM) between the query sequence and a unitig. Using the combination of the MPHF and the position vector, a k-mer is mapped to a unitig in the unitig sequence vector. The k-mer is then extended to a uni-MEM via a linear scan of the query sequence and the unitig sequence vector. Each uni-MEM can appear in multiple different references, and since uni-MEMs must be completely contained within a unitig, it is possible for multiple uni-MEMs to be directly adjacent on both the query and some references where the unitig appears.

uni-MEM collection: The first step in read alignment is to collect exact matches shared between the query (single-end or paired-end reads) and the reference. In PuffAligner, this is accomplished by collecting the set of uni-MEMs that co-occur between the query and reference. PuffAligner starts processing the read from the left-end and looks up each k-mer that is encountered until a match to the index is found. Once a match is discovered, it is extended in both query and the reference until one of these termination conditions occur: (1) a mismatch is encountered, (2) the end of the query is reached, or (3) the end of the unitig is reached. This process results in a uni-MEM match shared between the query and reference. uni-MEMs where extension is terminated as a result of reaching the end of a unitig must later be examined and potentially "collpased" together to form

MEMs with respect to the references on which they appear. If the uni-MEM extension is not terminated as a result of reaching the end of the query, then the position in the read is incremented by a small value and the same procedure is repeated for the next k-mer on the read. This process continues until either the uni-MEM extension terminates because the end of the query is reached, or because the last k-mer of the query is searched in the index. Here, we recall an important property of uni-MEM extension that is different from e.g. MEM extension or maximum mappable prefix (MMP) extension [45]. Due to the definition of the ccdBg, it is guaranteed that any k-mer appearing within a uni-MEM cannot appear in any other unintig in the ccdBg. Thus, extending k-mers to maximal uni-MEMs is, in some sense, safe with respect to greedy extension, as such extension will never cause missing a k-mer that would lead to another distinct uni-MEM shared between the query and reference. The concept of safe extension of kmer matches was introduced in [27].

Filtering highly-repetitive uni-MEMs: In order to avoid expending computation on performing the subsequent steps on regions of reads mapping to highly-repeated regions of the reference, any uni-MEM that appears more than a user-defined number of times in the reference is discarded. In this manuscript, we use the threshold of 1000. This filter has a strong impact on the performance, since, even if one k-mer from the read maps to a highly-repetitive region of the reference, the following expensive steps of the alignment procedure should be performed for every mapping position of the uni-MEM to find the right alignment for the read, while the less repetitive uni-MEMs also map to the true origin of the read on the reference as well. The drawback of this filter is that for a very small fraction of

the reads which are truly originating from a highly-repetitive region, all of the matched uni-MEMs will be filtered out and no hit remains for aligning the read. However, we find that in the case of aligning paired-end reads, usually one end of the read maps to a non-repetitive region, then, the alignment of the other end can be recovered using orphan recovery (explained in Section 2.2.6). Futheremore, we also provide a flag – *allowHighMultiMappers* that mitigates the effect of this filter for a slight tradeoff on the alignment performance.

uni-MEM compaction: For paired-end reads, PuffAligner aligns each end the read pairs individually. For each end, all the uni-MEMs are sorted on the basis of their positions on the reference. Consecutive uni-MEMs with no gap (both on the reference and the read) are merged into larger MEMs. The compactable uni-MEMs result from terminating the extension process due to reaching the end of a unitig. Such consecutive uni-MEMs can be safely compacted to form longer MEMs that will be used later in the MEM chaining algorithm. After the compaction of uni-MEMs, there is a list of MEMs which are shared sequences between the query and a set of reference positions, that are sorted based on the reference positions.

2.2.3 Finding promising MEM chains

As shown in fig. 2.6, having all the MEMs (maximal exact matches) from a read to each target reference, the goal of this step is to find promising chains of MEMs that cover the most unique bases in the read in a concordant fashion and that can potentially lead to a high quality alignment.

To accomplish this, we adopt the dynamic programming approach used in minimap2 [9] for finding co-linear chains of MEMs that are likely candidates to support high-scoring read alignments. As mentioned in minimap2, all the MEMs from a read r to the reference t, are sorted by the ending position of the MEMs on the reference. Then, this algorithm computes a score for each set of MEMs based on the number of unique covered bases in the read, the coverage score is also penalized by the length of the gaps, both in the read and reference sequence, between each consecutive pair of MEMs.

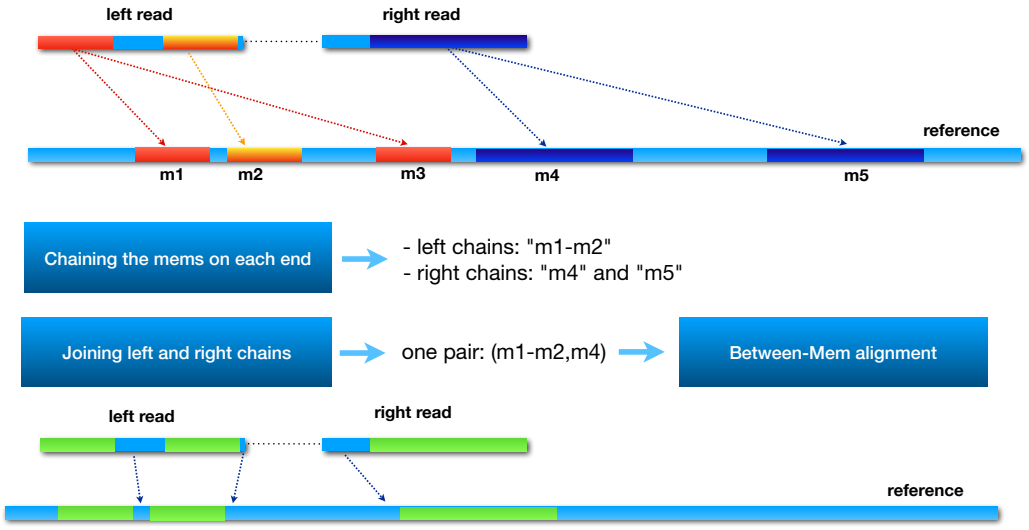


Figure 2.6: This figure shows the main steps of chaining and between-MEM alignment in the PuffAligner procedure via an example. In this example, m1, m2 and m3 are the projected MEMs from the left end of the read to the reference and m4 and m5 are the projected MEMs from the right end of the read. In the first step, the chaining algorithm chooses the best chain of MEMs that provide the highest coverage score for each end of the read, that is the m1-m2 chain for the left end and two single MEM chain for the right end. Then, the selected chains from each end are joined together to find the concordant pairs of chains, that is the (m1-m2, m4) pair for this read as m5 is too far from m1-m2. Then, the chain from each end will go through to the next step, between-MEM alignment. For the green areas (MEMs) no alignment is recalculated as they are exact matches. Only the un-matched blue parts of the chains (those nucleotides not occurring within a MEM) are aligned using a modified version of KSW2.

In PuffAligner, if the distance between two MEMs, m_1 and m_2, on the read and the reference is d_r and d_t respectively, these two MEMs should not be chained together if $|d_r - d_t| > C$, where C is the maximum allowed gap. So, the penalization term, the β value in [9], in the coverage score computation is modified accordingly to prevent pairing of such MEMs.

Also, unlike what is done in minimap2 [9], rather than considering together the MEMs that are discovered on both ends of a paired-end read, we consider the chaining and chain filtering for each end of the read separately. This is done in order to make it easier to enforce the orientation consistency of the individual chains. Specifically, the chaining algorithm that is presented in minimap2 [9] introduces a transition in the recursion that can be used to switch between the MEMs that are part of one read and those that are part of the other. However, such switching makes it difficult to enforce the orientation consistency of the chains that are being built for each end of the read. One solution to this problem is to add another dimension to the dynamic programming table, encoding if one has already switched from the MEMs of one read end to the other, and the recurrence can be modified to allow only one switch from the one read end to the other, allowing enforcement of orientation consistency. However, we found that, in practice, simply chaining the read ends separately led to better performance.

Finally, we also adopt the heuristic proposed by minimap2 [9] when calculating the highest scoring chains. That is, when a MEM is added to the end of an existing chain, it is unlikely that a higher score for a chain containing this MEM will be obtained by adding it to a preceding chain. Thus, we consider only a small fixed number of rounds (by default 2) of preceding chains once we have found the first chain to which we can add the current

MEM.

The chaining algorithm described above finds the best chains of MEMs shared between the read r and the reference t in orientation o. A chain is accepted if its score is greater than a configurable fraction, which we call the *consensusFraction*, times the maximum coverage score found for the read r to *any* reference. Throughout all the experiments in this manuscript the *consensusFraction* is set to 0.65. If a chain passes the consensus fraction threshold, we call it a *valid* chain. Additionally, rather than keeping all valid chains, we also filter highly-suboptimal chains with respect to the highest scoring chain *per-reference*. All valid chains shared between r and t are sorted by their scores, and chains having scores within 10% of the highest scoring chain for reference t are selected as potential mappings of the read r to the reference t. While these filters are essential for improving the throughput of the algorithm in finding the right alignment, they are carefully selected to have very little effect on the sensitivity of PuffAligner. For all the experiments in this manuscript, the same default settings of these parameters are used if not mentioned otherwise.

2.2.4 Computing base-to-base alignments between MEMs

After finding the high-scoring MEM chains for each reference sequence, a base-to-base alignment of the read to each of the candidate reference sequences is computed. Each selected chain implies a position on the reference sequence where the read might exhibit a high quality alignment. Thus, we can attempt to compute an optimal alignment of the read to the reference at this implied position, potentially allowing a small bit of padding

on each side of the read. This approach utilizes the positional information provided by the MEM chains. However, the starting position of the alignments is not the only piece of information embedded in the chains. Rather each chain of MEMs consists of sub-sequences of the read (of size at least k, though often longer) which match exactly to the reference. While the optimal alignment of the read to the reference at the position being considered is not *guaranteed* to contain these exact matches as alignments of the corresponding substrings, this is almost always the case.

In PuffAligner, we aim to exploit the information from the long matches to accelerate the computation of the alignments. In fact, since only chains with relatively high coverage score are selected, a large portion of the read sequences are typically already matched to the positions in the reference with which they will be matched in the final optimal alignment. For instance, in fig. 2.6, for the final chains selected on the reference sequence, it is already known for the light blue, dark blue and green sub-sequences on the left end of the read precisely where they should align to the reference. Likewise this is the case for the yellow and purple sub-sequences on the right read. The unmapped regions of the reads are either bordered by the exact matches on both sides, or they occur at the either ends of the read sequence. PuffAligner skips aligning the whole read sequence by considering the exact matches of the MEMs to be part of the alignment solution. As a result, it is only required to compute the alignment of the small unmapped regions, which reduces the computation burden of the alignments.

When applying such an approach, two different types of alignment problems are introduced, which we call bounded sub-sequence alignment and ending sub-sequence alignment. For bounded sub-sequence alignment, we need to *globally* align some interval

i_r of the read to an interval i_t of the reference. If i_r and i_t are of different lengths, the alignment solution will necessarily include insertions or deletions. If i_r and i_t are of the same length, then the optimal global alignment between them may or may not include indels. For each such bounded sub-sequence alignment, we determine the optimal alignment of i_r to i_t by computing a global pair-wise alignment between the intervals, and stitching the resulting alignment together with the exact matches that bound these regions.

Gaps at the beginning or the end of the read are symmetric cases, and so we describe, without loss of generality, the case where there is an unaligned interval of the read after the last MEM shared between the read and the reference. In this case, we need to solve the ending sub-sequence alignment problem. Here, the unaligned interval of the read consists of the substring spanning from the last nucleotide of the terminal MEM in the chain, up through the last nucleotide of the read. There is not a clearly-defined interval on the reference sequence. While the left end of the relevant reference interval is defined by the last reference nucleotide that is part of the bounding MEM, the right end of the reference interval should be determined by actually solving an extension or "end-free" alignment problem. We address this by performing extension alignment of the unaligned interval of the read to an interval of the reference that begins on the reference at the end of the terminal MEM, and extends for the length of the unaligned query interval plus the length of some problem-dependent buffer (which is determined by the maximum length difference between the read and reference intervals that would still admit an alignment within the acceptable score threshold).

An example of both of these cases is displayed in Figure 2.6. Specifically, an alignment of the read could be obtained by only solving two smaller alignment problems;

one is the ending sub-sequence alignment of the unmapped region after the green MEM on the left read and the other is the bounded sub-sequence alignment of region on the right read bordered by the yellow and purple MEMs.

PuffAligner uses KSW2 [9, 57] for computing the alignments of the gaps between the MEMs and for aligning the ending sequences. KSW2 exposes a number of alignment modes such as global and extension alignments. For aligning the bounded regions, KSW2 alignment in the global mode is performed, and for the gaps at the beginning or end of reads, PuffAligner uses the extension mode to find the best possible alignment of that region. PuffAligner, by default, uses a match score of 2 and mismatch penalty of 4. For indels, PuffAligner uses an affine gap scoring schema with gap open penalty of 5 and gap extension penalty of 3. In PuffAligner, after computing the alignment score for each read, only the alignments with a score higher than τ times the maximum possible score for the read are reported. The value of τ is controlled by the option *–minScoreFraction*, which is set to 0.65 by default.

2.2.5 Enhancing alignment computation

By only aligning the read's sub-sequences that are not included in the MEMs, the size of alignment problems being solved in PuffAligner are often much shorter than the length of the read. However, to further speed up alignment, we also incorporate a number of other techniques to improve the performance of the alignment calculation. We describe the most important of these below:

- **Skipping alignment calculation by recognizing perfect chains and alignment**

caching: It is possible to avoid the alignment computation completely in a considerable number of cases. In fact, as has been explained in previous work [27], the alignment calculation step can be completely skipped if the set of exact matches for each chain covers the whole read. PuffAligner skips alignment for cases where the coverage score of chains of MEMs is the length of the read, and assigns a total matched CIGAR string for that alignment. Alignment computation of a read might be also skipped if the same alignment problem has been already detected and computed for this read. For example, in the case of RNA seq data, reads often map to the same exons on different transcripts. In such cases, each alignment solution for a read is stored in a cache (a hash table) so that if the same alignment problem is detected, the solution can be directly retrieved from the cache, and no further computation is required (see table 2.5).

sample	Cache Hits	Perfect Chains	None Alignable	Total Skipped
DNA-seq experimental	52.894%	19.008%	0.710%	72.670%
RNA-seq simulated	28.692%	50.803%	0.970%	80.460%
Metagenomic simulated	61.096%	31.334%	0.000 %	92.430%

Table 2.5: The percentage of aligner engine calls skipped in the alignment calculation pipeline.

- **Early stopping of the alignment computation when a valid score cannot be achieved:** While care is taken to produce only high-scoring chains between the read and reference, it is nonetheless the case that the majority of the chains do not lead to an alignment of acceptable quality. Since the minimum acceptable alignment score is immediately known based on τ and the length of the read, the base-to-base alignment calculation can be terminated at any point where it becomes

imposible for the minimum required alignment score to be obtained. This approach can be applied both during the KSW2 alignment calculation, and also after the alignment calculation of each gap is completed. During this procedure, for each base at position i, starting from position 1 on the read of length n, if the best alignment score p up to the i-th position is s_i, we can calculate the maximum possible alignment score, s_{max}, that might be achieved starting at this location given the current alignment score by:

$$s_{max} = s_i + MS * (n - s_i), \tag{2.2}$$

where MS is the score assigned to each match. If s_{max} is smaller than minimum required score for accepting the alignment, the alignment calculation can be immediately terminated, since it is already known that this anchor is not going to yield a valid alignment for this read.

- **Full-sensitivity banded alignment:** KSW2 is able to perform banded alignment to make alignment calculation more efficient. In this mode, the dynamic programming matrix for the alignment problem is only filled out along the sub-diagonals out to a certain distance d away from the main diagonal. If one is guaranteed that any valid alignment must have fewer than d insertions or deletions, then the alignment must not exit these bands of the dynamic programming matrix. Note that alignments with $> d$ indels can be represented within these bands as insertions and deletions move in opposite anti-diagonal directions, but it is certainly the case that no alignment with $\leq d$ indels can exit these bands. By calculating the maximum number of gaps

(insertions or deletions) allowed in each sub-alignment probem, in a way that we are certain that any alignment having greater than this number of gaps must drop below the acceptable threshold, we utilize the banded alignment in KSW2 within each sub-alignment problem without losing any sensitivity with respect to non-banded alignment.

2.2.6 Joining mappings for read ends and orphan recovery

Finally, once alignments have been computed for the individual ends of a read, they must be paired together to produce valid alignments for the entire fragment. At this point in the process, on each reference sequence, there are a number of locations where the left end of each read or the right end of each read, or both, are mapped to the reference. For the purpose of determining which mappings will be reported as a valid pair, the mappings are joined together only if they occur on opposite strands of the reference, and if they are within a maximum allowed fragment length. There are two different types of paired-end alignments that can be reported by PuffAligner; concordant and discordant. If PuffAligner is disallowed from reporting discordant alignments, then the mapping orientation of the left and right end should agree with the library preparation protocols of the reads. PuffAligner first tries to find concordant mapping pairs on a reference sequence, and if no concordant mapping is discovered and the tool is being run in a mode where discordant mappings are allowed, then PuffAligner reports pairs that map discordantly. Here, discordant pairs may be pairs that do not, for example, obey the requirement of originating from opposite strands. While this is not expected to happen

frequently, it may occur if there has been an inversion in the sequenced genome with respect to the reference.

Orphan recovery: If there is no valid paired-end alignment for a fragment (either concordant or discordant, if the latter is allowed), then PuffAligner will attempt to perform orphan recovery. The term "orphan" refers to one end of paired-end read that is confidently aligned to some genomic position, but for which the other read end is not aligned nearby (and paired). To perform orphan recovery, PuffAligner examines the reference sequence downstream of the mapped read (or upstream if the mapped read is aligned to the reverse complement strand) and directly performs dynamic programming to look for a valid mapping of the unmapped read end. For this purpose, we use the "fitting" alignment functionality of edlib [34] to perform a simple Levenshtein distance based alignment that will subsequently be re-scored by KSW2. Finally, if, after attempting orphan recovery, there is still no valid paired-end mapping for the fragment, then orphan alignments are reported by PuffAligner (unless the "`--noOrphans`" flag is passed).

2.2.7 Assessing PuffAligner's performance

For measuring the performance of PuffAligner and comparing it to other aligners, we have designed a series of experiments using both simulated and experimental data from different sequencing assays. We compare PuffAligner with *Bowtie2* [5], STAR [45] and deBGA [8]. *Bowtie2* is a popular, sensitive and accurate aligner with the benefit of having very modest memory requirements. STAR requires a much larger amount of memory, but is much faster than *Bowtie2* and can also perform "spliced alignment"

against a reference (which PuffAligner, *Bowtie2*, and deBGA currently do not allow). deBGA, is most-related tool to PuffAligner conceptually, as it is an aligner with a colored compacted de Bruijn graph-based index that is focused on exploiting redundancy in the reference sequence.

We use different metrics to assess both the performance and accuracy of each method on a variety of types of sequencing samples. These experiments are designed to cover a variety of different use-cases for an aligner, spanning the gamut from situations where most alignments are expected to be unique (DNA-seq), to situations where each fragment is expected to align to many loci with similar quality (RNA-seq and metagenomic sequencing), and spanning the range of index sizes from small transcriptomes to large collections of genomes.

First, we show PuffAligner exhibits similar accuracy for aligning DNA-seq reads to *Bowtie2*, but it is considerably faster. In the case of experimental reads, since the true origin of the read is unknown, we use measures such as mapping rate and concordance of alignments to compare the methods. Furthermore, we evaluate the accuracy of aligners by aligning simulated DNA-seq reads that include variation (single-nucleotide variants and small indels with respect to the reference). For aligning RNA-seq reads, we compare the impact of alignments produced by each aligner on downstream analysis such as abundance estimatation. Finally, we show PuffAligner is very efficient for aligning metagenomic samples where there is a high degree of shared sequence among the reference genomes being indexed. We also illustrate that using alignments produced by PuffAligner yields the highest accuracy for abundance estimation of metagenomic samples.

2.2.8 Configurations of aligners in the experiments

The performance of each tool is impacted by the different alignment scoring schemes they use, e.g. different penalties for mismatches, and indels. To enable a fair comparison, we attempted to configure the tools so as to minimize divergences that simply result from differences in the scoring schemes. For the experiments in this paper, we use *Bowtie2* in a near-default configuration (though ignoring quality values), and attempt to configure the other tools, as best as possible, to operate in a similar manner.

The deBGA scoring scheme is not configurable, so we use this aligner in the default mode (unfortunately, the inability to disable local alignment and forcing just computation of end-to-end alignments in deBGA makes certain comparisons particularly difficult). For PuffAligner we use a scheme as close to *Bowtie2* as possible. The maximum possible score for a valid alignment in *Bowtie2* is 0 (in end-to-end mode) and each mismatch or gap subtracts from this score. *Bowtie2* uses an affine gap penalty scoring scheme, where opening and extending a gap (insertion or deletion) have a cost of 5 and 3 respectively. For DNA-seq reads, we configure STAR to allow as many mismatches as *Bowtie2* and PuffAligner by setting the options "`--outFilterMismatchNoverReadLmax 0.12`" and "`--outFilterMismatchNmax 1000`". Also, we use "`--alignIntronMax 1`" in STAR to perform non-spliced alignments while aligning genomic reads. For RNA-seq reads, STAR has a set of parameters which we change in our result evaluations, and which are detailed below in the relevant sections.

In *Bowtie2* we also use the option `--gbar 1` to allow gaps anywhere on the read except within the first nucleotide (as the other tools have no constraints on where

indels may occur). Furthermore, for consistency, we also run *Bowtie2* with the option "`--ignore-quals`", since the other tools do not utilize base qualities when computing alignment scores.

As explained in Section 2.2.2, for the sake of performance, highly repeated anchors (more than a user-defined limit) will be discarded before the alignment phase. This threshold is by default equal to 1000 in PuffAligner. We set the threshold to the same value for STAR and deBGA using options `--outFilterMultimapNmax 1000` and `-n 1000` respectively. There is no such option exposed directly in *Bowtie2*.

Since PuffAligner finds end-to-end alignments for the reads, we are also running other tools in end-to-end mode, which is the default alignment mode in *Bowtie2* as well. In STAR we enable this mode using the option `--alignEndsType EndToEnd`. In the case of deBGA, although the documentation suggests it is *not* supposed to find local alignments by default, the output SAM file contains many reads with relatively long soft clipped ends, so if a read is not aligned end-to-end, deBGA reports the local alignment for that. We were not able to find any option to force deBGA to perform end-to-end alignments for all reads, and so we have compared it in the configuration in which we were able to run it.

For aligning DNA-seq samples, each aligner is configured to report a single alignment, which is the primary alignment, for each read. *Bowtie2* outputs one alignment per read by default. To replicate this in the other tools, we use the option `--outSAMmultNmax 1` in STAR, `-o 1 -x 1` in deBGA, and `--primaryAlignment` in PuffAligner.

2.2.9 Alignment of whole genome sequencing reads

First, we evaluate the performance of PuffAligner with a whole genome sequencing (WGS) sample from the 1000 Genomes project [58]. We downloaded the `ERR013103` reads from sample `HG00190`, which is a low-coverage sample from a Finnish male, sequenced in Finland. [4]. There are $18,297,585$ paired-end reads, each of length 108 nucleotides in this sample. Using fastp [59], we remove low quality ends and adapter sequences from these reads. After trimming, there are $15,404,412$ reads remaining in the sample. Indices for each of the tools are built over all DNA chromosomes of the latest release of the human genome (v33) by gencode[5] [60].

In this experiment, all aligners are configured report only concordant alignments, i.e., only pairs of alignments that are cocordant and within the "maximum fragment length" shall be reported. The maximum fragment length in all aligners is set to 1000, using the option `--alignMatesGapMax 1000` in STAR, `--maxins 1000` in *Bowtie2* and `-u 1000 -f 0` in deBGA. The default value for the maximum fragment length in PuffAligner is set to 1000, the user can cofigure this value by using the flag `--maxFragmentLength`. This concordance requirements also prevents *Bowtie2*, PuffAligner, and STAR from aligning both ends of a paired end read to the same strand.

The alignment rate, run-time memory usage and running time for all the aligners are presented in table 2.6. The reason that deBGA has the highest mapping rate in table 2.6 compared to other tools is that it is local alignments for the reads that are not alignable end-to-end under the scoring parameters for the other tools. *Bowtie2* and PuffAligner

[4] https://www.internationalgenome.org/data-portal/sample/HG00190
[5] https://www.gencodegenes.org/human/release_33.html

are both able to find end-to-end alignments for about $\sim 95\%$ of the reads. STAR and PuffAligner are the fastest tools, with STAR being somewhat faster than PuffAligner. On the other hand, PuffAligner is able to align more reads than STAR, while requiring less than half as much memory. The memory usage of *Bowtie2* is the smallest, since *Bowtie2*'s index does not contain a hash table. However, this comes at the cost of having the longest running time compared to other methods. Overall, PuffAligner benefits from the fast query of hash based indices while its run-time memory usage, which is mostly dominated by the size of the index, is significantly smaller than other hash based aligners. Although deBGA's index is based on the de Bruijn graphs, similar to the Pufferfish index, the particular encoding for it is not as space-efficient as that of Pufferfish.

aligner	mapping-rate($\%$)	time (mm:ss)	memory (GB)
PuffAligner	95.583	6:14	13.091
deBGA	99.753	10:46	41.041
STAR	93.881	4:29	30.358
Bowtie2	95.443	16:15	3.501

Table 2.6: The performance of different tools for aligning experimental DNA-seq reads. The time reports are benchmarked after warming up the system cache so that the influence of index loading time is mitigated.

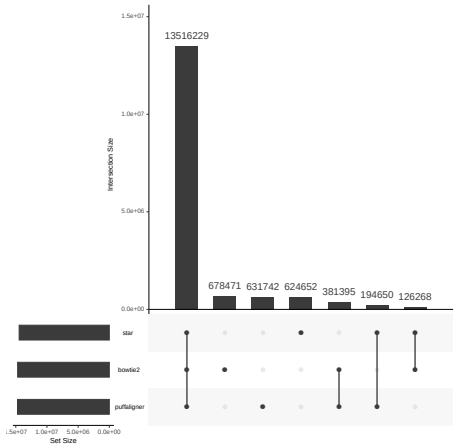


Figure 2.8: Caomparing the alignments in terms of agreement of the alignments found by different tools based on the location of the mappings.

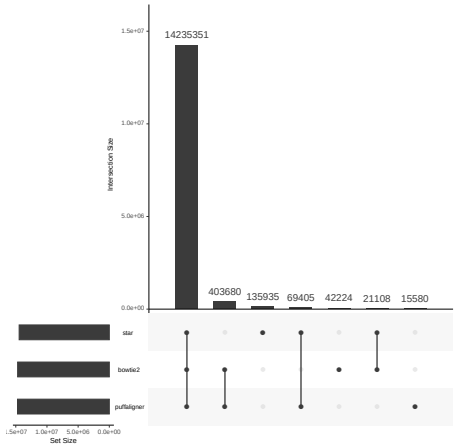


Figure 2.7: Caomparing the alignments in terms of agreement of the alignments found by different tools for each read.

To look more closely how the mappings between the tools differ, we investigate the agreement of the reads which are mapped by each tool and visualize the results in an upset plot in fig. 2.7 using the UpsetR library [61]. We are only comparing the three methods which perform end-to-end alignment in this plot, since outliers from the local alignments computed by deBGA would otherwise dominate the plot. The first bar shows that the majority of the reads are mapped by all three tools. The next largest set represents

the reads which are only mapped by *Bowtie2* and PuffAligner. All the other sets are much smaller compared to the first two sets. This fact illustrates that the highest agreement in the aligners is between *Bowtie2* and PuffAligner. Exploring a series of individual reads from the smaller sets in the upset plot, suggests that some of these differences happen as a result of small differences in the scoring configuration, while some result from different search hueristics adopted by the different tools. fig. 2.8 shows the coherence between the alignments reported by the tools by also including the exact location to which the reads are aligned in the reference.

2.2.10 Alignment of simulated DNA-seq reads in the presence of variation

To further investigate the accuracy of the aligners, we used simulated DNA-seq reads. One of the main differences between simulated reads and experimental reads is that simulated reads are often generated from the same reference sequences to which they are aligned, with the only differences being due to (simulated) sequencing error. While (simulated) sequencing error prevents most reads from being exact substrings of the reference, it actually does not tend to complicate alignment too much. On the other hand, while dealing with experimental data, the genome of the individual from which the sample is sequenced might include different types of variations with respect to the reference genome to which we are aligning [2]. Therefore, it is desirable to introduce variations in the simulated samples, and to measure the robustness and performance of the different aligners in the presence of the variation. Mason [62] is able to introduce different kinds of variations to the reference genome, such as SNVs, small gaps, and also

structural variants (SV) such as large indels, inversions, translocations and duplications. We use Mason to simulate 9 DNA-seq samples with different variation rates ranging from $1e-7$ to $1e-3$. Each sample includes $1M$ paired-end Illumina reads of 100bp length from chromosome 21 of the human genome, ensembl release 98 [6].

Figure 2.9: Comparing the accuracy of aligners in the presence of different rates of variations in the reference genome in terms of the precision of the alignments reported by each aligner. True positives (TP) are the compatible reads that are aligned to the original location, and the FP set consists of both the compatible reads aligned to sub-optimal locations (alignments with larger edit distance than the alignment to the original location) and the non-compatible reads that are aligned with high (> 25) edit distance.

[6] ftp://ftp.ensembl.org/pub/release-98/fasta/homo_sapiens/dna/

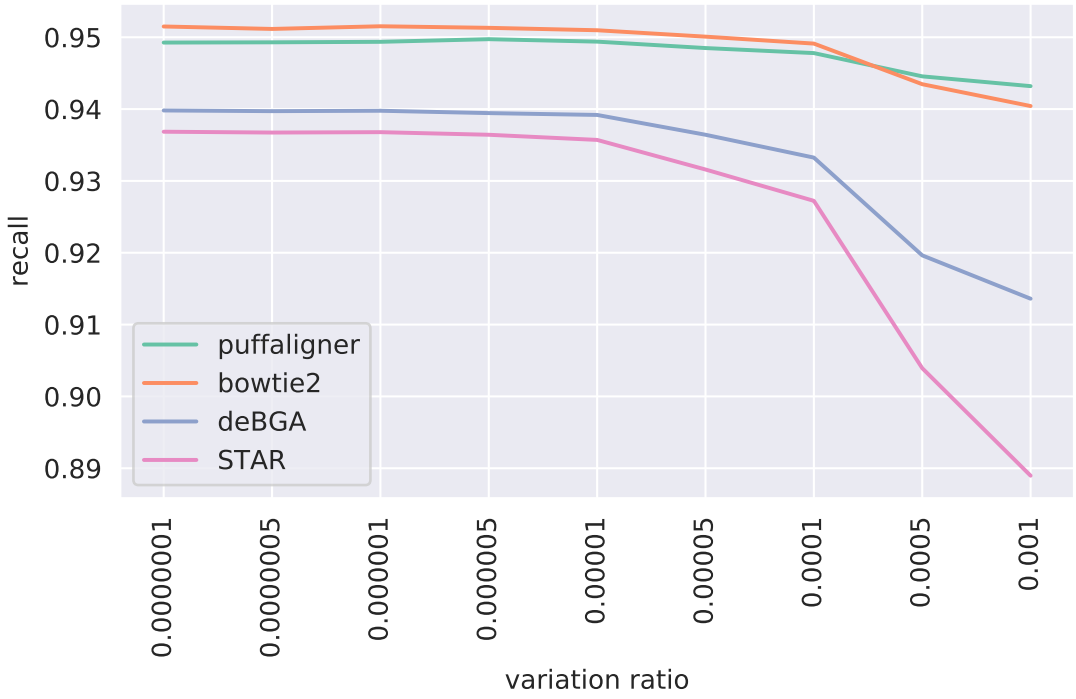


Figure 2.10: Comparing the accuracy of aligners in the presence of different rates of variations in the reference genome in terms of the ratio of the alignments in the true SAM file that are recovered by each aligner. The recall is the result of dividing the number of TP reads by the total number of compatible reads.

For this analysis, we do not restrict the aligners to only report concordant alignments, since the structural variations in the samples can lead to valid discordant alignments, such as those on the same strand or with inter-mate distances larger than the maximum fragment length. To be specific, we do not use the options which limit *Bowtie2* and PuffAligner to report only concordant alignments, in addition, we use the option "`--dovetail`" in *Bowtie2* to consider dovetail pairs as concordant pairs.

The alignments reported by deBGA already include discordant pairs and also orphan mappings. Furthermore, To remove any restrictions on the fragment length in the alignments reported by deBGA, we set the minimum and maximum insert size, respectively to 0 and the 50000, since setting a larger value resulted in the tool running into segmentation fault.

To allow dovetail pairs and also larger gaps between the pairs in STAR, we use the following options:

“--alignEndsProtrude 1000000 ConcordantPair”

“--alignMatesGapMax 1000000”

By default there is not a specific option in STAR for allowing orphan alignment of paired end reads. Instead, we can increase the number of allowed mismatches to be as large as one end of the read by using the following options:

“--outFilterMismatchNoverReadLmax 0.5”

“--outFilterMismatchNoverLmax 0.99”

“--outFilterScoreMinOverLread 0”

“--outFilterMatchNminOverLread 0”

For each sample, Mason produces a SAM file which includes the alignment of the simulated reads to the original, non-variant version of the reference — the version which was used for building the aligner’s indices in this experiment. Based on the alignments reported in the truth file, some reads did not have a valid alignment to the original reference. This was the result of a high rate of variations at some sequencing sites. We called the set of reads that, according to the truth SAM file, were aligned to the original reference as compatible reads.

We compared the performance of aligners based upon how well they are able to align the compatible reads. We computed the precision and recall of the alignments reported for these reads as follows. True positives are considered the reads that are mapped by the aligner to the same location stated by the truth file. Then, recall is computed by dividing the number of true positives by the number of all compatible reads.

Furthermore, we considered an alignment as a false positive in two different cases. First, an alignment was considered discordant if the reported alignment had a large edit distance (larger than 25) for the non-compatible reads. Second, in the case that an aligner reported an alignment to a location other than the one in the truth file, it was considered as a false positive if the edit distance of the reported alignment is greater than the edit distance of the true alignment. Having defined the set of TP and FP for the alignments, and also having considered the set of all compatible reads as the set we are trying to recover, we computed precision and recall for the set of alignments reported by each aligner.

Figure 2.10 shows the precision and recall of the aligners for different samples. According to fig. 2.10, for lower variation ratios up until $10e-5$, most of the tools are able to make accurate alignment calls with a high specificity. As the variation ratio introduced in the sample is increased, all the tools start to have lower precision and recall. deBGA and STAR perform worse in higher variation samples, as they fail to recover the true alignment for more reads, while *Bowtie2* and PuffAligner are able to align most of the reads to their true location on the original reference.

These results show that PuffAligner' accuracy is stable in the face of variation which makes the tool suitable for datasets that are known to have substantial variation, such as when aligning reads to microbial genomes where the specific sequenced strain may not be represented in the reference set.

2.2.11 Quantification of transcript abundance from RNA-seq reads

Mapping sequencing reads to target transcriptomes is the initial step in many pipelines for reference-based transcript abundance estimation. While lightweight mapping approaches [23, 24] greatly speed-up abundance estimation by, in part, eliding the computation of full alignment between reads and transcripts, there is evidence that alignments still yield the most accurate abundance estimates by providing increased sensitivity and avoiding spurious mappings [2, 27, 63]. Thus, the continued development of efficient methods for producing accurate transcriptome alignments of RNA-seq reads remains a topic of interest. In this section, we compare the effect of alignments produced by each tool on the accuracy of RNA-seq abundance estimation.

We generated $9{,}968{,}245.000$ paired-end RNA-seq reads using the polyester [64] read simulator. The reads are generated by the `simulate experiment countmat` module in polyester. The input count matrix is calculated based on the estimates from the *Bowtie2*-Salmon pipeline on the sample `SRR1085674` (where reads are first aligned with *Bowtie2* and then the alignments are quantified using Salmon). This sample is a collection of paired-end RNA-seq reads sequenced from human transcriptome using an Illumina HiSeq [65]. The human transcriptome from gencode release (33) is used to build all the aligners' indices. Also, for building STAR's index in the genome mode, the human genome and the comprehensive gene annotation (main annotation file) is obtained from the same release of gencode.

As the reads in this experiment are RNA-seq reads sequenced from the human transcriptome, it is important to account for multi-mapping, as often, a read might map

aligner	spearman	MARD	time (mm:ss)	memory (GB)
PuffAligner	0.920	0.052	1:17	2.543
deBGA	N/A	N/A	5:19	9.965
STAR- transcriptome	0.920	0.053	1:57	8.734
STAR- genome	0.901	0.064	3:30	32.573
Bowtie2	0.920	0.053	32:59	1.146

Table 2.7: Abundance estimation of simulated RNA-seq reads, computed by Salmon, using different tools' alignment outputs. The time and memory are only for the alignment step of each tool and the time for abundance estimation by Salmon is not considered.

to multiple transcripts which share the same exon or exon junction. This property makes the direct evaluation of performance at the level of alignments difficult. Therefore, a typical approach in evaluating the accuracy of the transcriptomic alignments is to assess the accuracy of downstream analysis such as abundance estimations by computing the correlation and relative differences of the estimates with the true abundance of the transcripts. To compare the accuracy of each tool we give the alignments produced by each aligner, which are in the SAM format, as input to Salmon to estimate the transcript expressions.

PuffAligner, by default, outputs up to 200 alignments with an alignment score greater than 0.65 times the best alignment score, i.e., the alignment for the read in the case that all bases are perfectly matched to the reference. To enable the multi-mapping to take into account the characteristics of alignment to the transcriptome, *Bowtie2* is run with the option `-k 200` which lets the tool output up to 200 alignments per read. The value of 200 is adopted from the suggested parameters for running RSEM [66] with *Bowtie2* alignments. We note that running *Bowtie2* with this option makes the tool considerably slower than the default mode, as many more alignments will be computed and output to the SAM file under this configuration. For both *Bowtie2* and PuffAligner, and also for STAR by default, orphan and discordant mappings are not allowed.

We ran STAR with the `'ENCODE'` options, which are recommended in the STAR manual for RNA-seq reads. STAR is also run in two different modes, one is by building the STAR index on human genome, while it is also provided a GTF file for gene annotation. In this mode, STAR performs spliced alignment to the genome, then projects the alignments onto transcriptomic coordinates. The other mode is building the STAR index on the human transcriptome directly, which allows STAR to align the RNA-seq reads directly to the transcripts in an unspliced manner. We chose to run STAR in the transcriptomic mode as well, since we find that it yields higher accuracy, though this increases the running time of STAR.

The deBGA index is built on the transcriptome, as are the *Bowtie2* and PuffAligner indices, since these tools do not support spliced read alignment. deBGA is run in the with options `-o 200 -x 200`, which nominally has the same effect as `-k 200` in *Bowtie2*, according to the documentation of deBGA.

Accuracy of abundance estimation by Salmon, when provided the SAM output generated by each aligner, is displayed in table 2.7. The timing and memory benchmarks provided in this table is only for the alignment step. Alignments produced by PuffAligner, *Bowtie2* and STAR in the transcriptomic mode produce the best abundance estimates. deBGA's output alignments are not suitable for any abundance estimation as many reads are aligned only to the same strand which are later filtered during the abundance estimation by Salmon, so we could not provide a meaningful correlations for abundance estimation using deBGA's alignments. Aligning the reads by STAR to genome and then projecting to transcriptomic coordinates does not generate as high correlation as directly aligning the reads to the transcriptome by STAR. However, we note that, as described by Srivastava

et al. [2], there are numerous reasons to consider alignment to the entire genome that are not necessarily reflected in simulated experiments. While the memory usage by PuffAligner is only 2 fold larger than memory used by *Bowtie2*, it computes the alignments much more quickly.

According to the results in table 2.7 PuffAligner is the fastest aligner in these benchmarks, and the accuracy as high as *Bowtie2* and STAR for aligning RNA-seq reads. Here, PuffAligner leads to the most accurate abundance estimates, while being 30 times faster than *Bowtie2*. Moreover, The memory usage is much less than other fast aligners such as STAR.

2.2.12 Alignment to a collection of microorganisms — simulated short reads

To demonstrate the performance and accuracy of PuffAligner for metagenomic samples, we designed two different experiments. One main property of metagenomic samples is the high similarity of the reference sequences against which one typically aligns, where a pair (or more) of references may be more than 90% identical. The first experiment we designed for this scenario, to specifically evaluate issues related to this challenge, we call the "single strain" experiment. Additionally, metagenomic samples also have the property of containing reads from a variety of genomes, some of which are not even assembled yet – and hence unknown. This leads to the second experiment, which we call the "bulk" experiment, that compares the aligners in the presence of a high variety of `species` in the sample in addition to the high similarity of references.

For simplicity and uniformity, all the experiments have been run in the concordant mode for both PuffAligner and *Bowtie2* (both of which support such an option), disallowing orphans and discordant alignments. All aligners are run in three different confiurations, allowing three specific maximum numbers of alignments per fragment; 1 (primary output with highest score, breaking ties randomly), 20, and 200. PuffAligner and STAR, as the only tools that support this option, also are run in the *bestStrata* mode. In this mode, the aligner outputs all *equally-best* alignments for a read with highest score without the limitation on number of reported alignments. This option is inspired by the similarly-named option in Bowtie1 [67]. However, unlike Bowtie1, PuffAligner and STAR only make a best-effort attempt to find the score of the best stratum alignments, and do not guarantee to find the best stratum (though the cases in which they fail to seem to be exceedingly rare). This option is especially useful in the metagenomic analyses, as we will report only the best-score alignments without having an arbitrary limitation on the number of allowed alignments. This allows proper handling of highly multi-mapping metagenomic reads. In other words, using this option, one can achieve a high sensitivity without the need to hurt specificity. The details of each experiment is explained in the following sections.

2.2.13 A single-strain Experiment

For this experiment, we download the viral database from NCBI, and choose three similar coronavirus genomes. This set includes one of the recently-uploaded samples from Wuhan [68, 69]. We select three very similar viral genomes to simulate reads

from, which are: NC_045512.2, NC_004718.3, and NC_014470.1. There are also a lot of literature discussing the similarity in sequence and behavior for these three species of coronavirus [70, 71, 72]. The first is the complete genome for severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1 known as Covid19 with length of $29,904$ bases. NC_004718.3 is the ID of SARS coronavirus complete genome (length: $29,752$) and finally, NC_014470.1 is a Bat coronavirus BM48-31/BGR/2008 complete genome (length: $29,277$).

We use Mason [62] to generate three simulated samples, each sample contains $500,000$ reads only from one of the three viral references we mentioned earlier. Then, reads were aligned back to the database of viral sequences using each of the four aligners. The results are shown in table 2.8 for the reads simulated from the covid19 strain.

As the results show, the alignments of all aligners, except for deBGA, are distributed only across the three references of interest out of all the reference sequences in the complete viral database. deBGA reports only a few alignments to a forth virus. In general, all of the aligners do a good job of reporting the correct alignment among the returned alignments for each read. Here, we are more interested in exploring how sub-optimal alignments are computed and filtered under different settings when aligning to a collection of very similar genomes. The results show that all tools have very high sensitivity even when considering only a single (primary) alignment per read. As we allow more alignments to be reported, the sensitivity increases and quickly levels off for all the tools. On the other hand, more alignments are generated and *Bowtie2*, in particular, generates a considerable number of extra alignments as the maximum number of allowed reported alignments is increased. However, the results do not change when allowing more

than 20 alignments, which means no more than 20 alignments ever pass the alignment score threshold for these reads in the viral database for any of the tools we are testing.

The results indicate that, when allowing more than one alignment to be reported for every read, *Bowtie2* tends to report a large number of sub-optimal (yet, still valid) alignments compared to other tools. These are alignments that are accepted within the alignment score threshold, but are to another target than the one from which the read truly originates. Generating these sub-optimal alignments is in no way wrong, but it has a non-trivial computational cost, as shown in fig. 2.11, even if these alignments are not used in downstream analysis. Further, the score of the best alignment for each read is specific to that read and not known ahead of time, meaning that this situation cannot be completely addressed simply by setting more stringent parameters for which alignment scores should be allowed. This behavior of *Bowtie2* gives the other tools a computational advantage when the user only truly requires the set of equally-best alignments for each read.

Interestingly, there is one read that all tools, except for PuffAligner fail to properly align. Inspecting this alignment reveals it is a valid alignment within the range of the acceptable scoring threshold, and it is unclear why it is not discovered by the other tools. Overall, the aligners tested perform very well here in reporting the true strain of origin without reporting too many extra alignments. Interestingly, despite changing the parameters to allow more alignments, STAR tends to return the same set of alignments under all configurations in this experiment. Figure 2.11 shows that PuffAligner has the lowest running time, even when the number of allowed alignments per read increases.

BestStrata Mode In this small example, all tools showed good sensitivity (and PuffAligner and STAR showed near-perfect sensitivity) even when reporting only a single-alignment per read. This experiment is, of course, an atypically small test for multi-mapping read. In in larger samples, with reads deriving from more organisms and a larger database of references, permitting more alignments usually yields non-trivial improvements in sensitivity. To control the rate of reporting sub-optimal alignments, PuffAligner supports the "best strata" option – also available to STAR, which allows only the alignments with the best calculated score to be reported (as a replacement for maximum allowed number of alignments). Using this option, PuffAligner achieves full specificity and sensitivity in this experiment table 2.8. We further demonstrate the positive impact of this option on the alignment of bulk metagenomic samples in the next section.

Alignment Mode	Tool	NC_045512.2	NC_004718.3	NC_014470.1	Others
Primary	PuffAligner	500,000.000	0.000	0.000	0.000
	Bowtie2	499,981.000	18	0.000	0.000
	STAR	499,999.000	0	0.000	0.000
	deBGA	499,991.000	0	0	9
Up to 20	PuffAligner	500,000.000	134.000	46.000	0.000
	Bowtie2	499,999.000	21,461.000	2,311.000	0.000
	STAR	499,999.000	0.000	0.000	0.000
	deBGA	499,991.000	0.000	0.000	9.000
Up to 200	PuffAligner	500,000.000	134.000	46.000	0.000
	Bowtie2	499,999.000	21,461.000	2,311.000	0.000
	STAR	499,999.000	0.000	0.000	0.000
	deBGA	499,991.000	0.000	0.000	9.000
Best strata	PuffAligner	500,000.000	0.000	0.000	0.000
	STAR	499,999.000	0.000	0.000	0.000

Table 2.8: Alignment Distribution for 500000 simulated reads from reference sequence NC_045512.2 (known as covid19). The best specificity is achieved by PuffAligner in `bestStrata` mode (as well as the primary mode). In this simulated sample, many alignments are not ambiguous, resulting in the good performance observed when using only primary alignments. However, typically in metagenomic analysis, many equally-good alignments exist, and selecting only one is equivalent to making a random choice.

2.2.14 Experiments with a mixture of organisms

We chose a random set of 4000 complete bacterial genomes downloaded from the NCBI microbial database and constructed the indices of PuffAligner, *Bowtie2*, STAR, and deBGA on the selected genomes. Figure 2.13 shows the time and memory required for constructing each of the indices, while the size of the final index on disk is displayed in fig. 2.12. Overall, PuffAligner and *Bowtie2* show a similar trend in time and memory requirements, while STAR and deBGA require an order of magnitude more memory. In terms of the final index size, *Bowtie2* has the smallest index, PuffAligner has the second-smallest, and STAR has the largets.

For simulating a bulk metagenomic sample, we generated a list of simulated whole genome sequencing (WGS) reads through the following steps:

- Select a real metagenomic WGS read sample
- Align the reads of the chosen real experiment to the 4000 genomes using *Bowtie2*, limiting *Bowtie2* to output one alignment per read.
- Choose all the references with count greater than C from the quantification results. This defines the read distribution profile that we will use to simulate data.
- For each of the expressed references, use Mason [62], a whole genome sequence simulator, to simulate $100bp$ paired-end reads with counts proportional to the reported abundance estimates so that total number of reads is greater than a specified value n. In this step we ran Mason with default options.

Table 2.9: Basic information for samples selected for simulating mock bulk metagenomic samples. The SRR10948222 sample is collected for Finding sub-biocrust soil microbial communities in Mojave Desert, California, United States. The SRR11283975 sample collected from the Jiaodong Peninsula, China to study the impact of different acidification degrees on the bacterial community. The SRR11496426 sample is collected from the oil site of Uzon Caldera to study the composition, genetics characteristics and structure of the microbial communities.

Accession	# of reads	# of reads aligned to 4k selected reference	# of simulated reads	# of references of origin for the simulated reads
SRR10948222	27,296,270	200k	5,550,650	98
SRR11283975	35.5k	8,333	1,012,176	92
SRR11496426	42.3k	30,203	1,029,382	179

- Mix and shuffle all of the simulated reads from each reference into one sample which is used as the mock metagenomic sample.

We selected three Illumina WGS samples that are publicly available on NCBI. A soil experiment with accession ID `SRR10948222` [73] from a project for finding sub-biocrust soil microbial communities in the Mojave Desert. The sample has $\sim 27M$ paired-end reads, containing a mixture of genomes from various genera and families. However, less than $200k$ of the reads in the sample were aligned to the strains present in our database, leading the selection of 98 species from a variety of genera. We scaled the read counts in the simulation to $\sim 50M$ reads. The other two selected samples are `SRR11283975` and `SRR11496426` the details of which are explained in table 2.9. In this section we only report the performance of the tools on the first sample.

The assessment of "accuracy" directly from the aligned reads is not a trivial task. Due to the high rate of multi-mapping in these simulated samples, and due to the fact that

multiple references can produce alignments of the same quality as the "true" origin of the read, we calculate the accuracy by comparing the true and estimated abundances using a quantification tool (in this case, Salmon) rather than by comparing the read alignments directly.

In table 2.10 the accuracy metrics are calculated over the abundance estimations obtained using the alignments produced by running the aligners in the different modes specified. The list of metrics for metagenomic expression evaluations have been chosen to be similar to previous work such as in Bracken [74] and Karp [75].

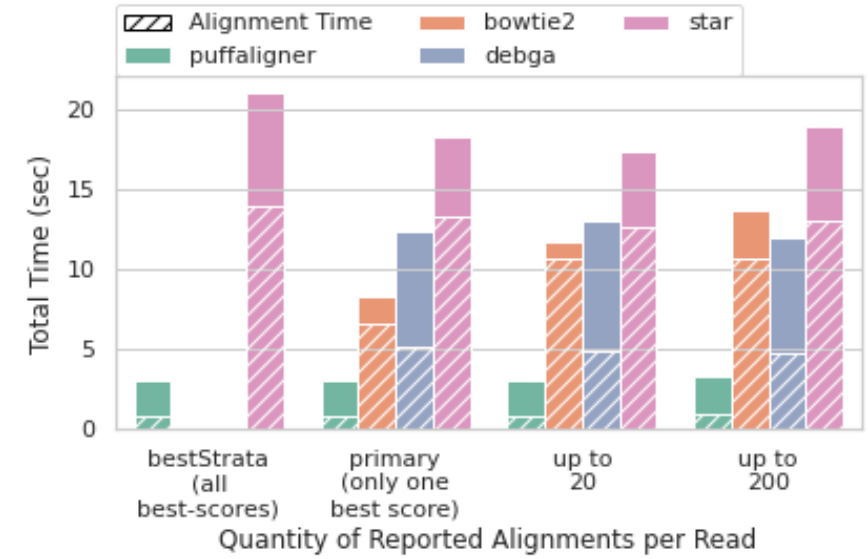
The metrics selected are *Spearman Correlation*, *Mean Absolute Relative Difference (MARD)*, *Mean Absolute Error (MAE)*, and *Mean Squared Log Error (MSLE)*. Each metric measures different characteristics of the predicted versus true abundance estimates. For example, lower MARD indicates better distribution of the reads among the references relative to the abundance of each reference, while MAE shows the quality of the distribution of the reads in a more absolute way regardless of the difference between the abundance of the references. In this case, one misclassified read has the same impact on the MAE metric both for a high-abundance and low-abundance reference.

This experiment leads to three main observations. First, regardless of the alignment mode, quantifications derived from the deBGA alignments seem to lead to systematic underestimation of abundance. However, PuffAligner, STAR and *Bowtie2*, show very similar behavior with respect to accuracy. STAR is the best in primary mode as well as when allowing 20 alignments, closely followed by PuffAligner. When allowing up to 200 alignments per read, *Bowtie2* tends to yield the most accurate abundances, again with PuffAligner being the close runner-up. These results demonstrate that PuffAligner is a

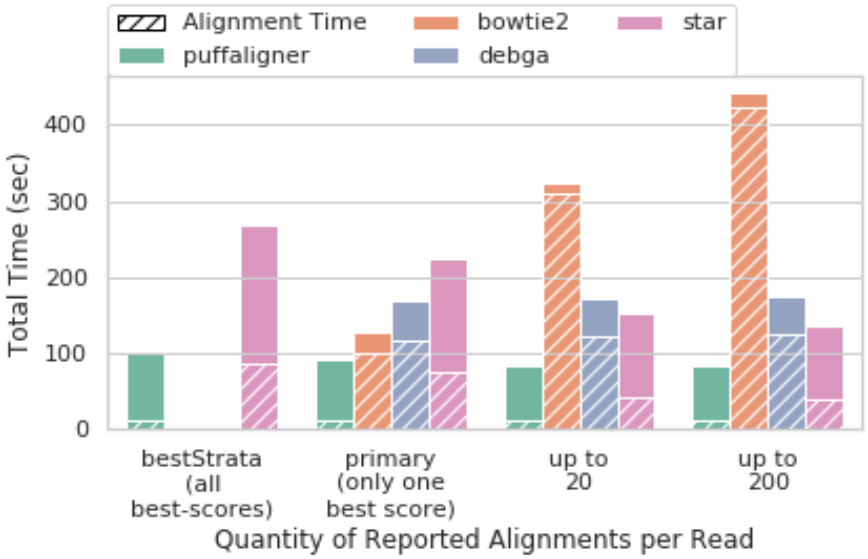
reliable alignment tool showing a stable pattern of being comparable to the best aligner under all the scenarios tested. That is, the good performance of PuffAligner is robust across a variety of different parameter settings.

Moreover, due to the nature of the metagenomic data — the high degree of ambiguity and multi-mapping — we expect to see improvement in the accuracy metrics as more alignments are reported per read, as this leads to a higher recall. While STAR's accuracy changes only slightly from 20 alignments to 200 alignments (only improving MAE) the results for PuffAligner and *Bowtie2* improve considerably when allowing more alignments per read. However, this higher accuracy comes in the cost of alignment time for *Bowtie2*. As shown in fig. 2.11, *Bowtie2* alignment time increases sharply when allowing more alignments per read, while PuffAligner exhibits only small changes in alignment time regardless of the maximum number of alignments being reported per read. The difference becomes especially evident when allowing up to 200 alignments per read, where PuffAligner is 4 times faster than *Bowtie2*. Additionally, in experimental data, many of the alignments reported do not necessarily have high quality, and only appear in the output as one of the 200 alignments for the read. In fact, we note the similar accuracy achieved by PuffAligner in *bestStrata* mode compared to when we allow up to 200 alignments per read. In the other two samples PuffAligner is the most accurate aligner in different modes for both samples.

Overall, these results along indicate that PuffAligner is a sensitive and fast aligner. Specifically PuffAligner exhibits similar accuracy (and is sometimes more accurate) as well-known aligners like *Bowtie2* and STAR. On these data, it exhibits memory requirements close to those of the memory-frugal *Bowtie2*, while being much faster.



(a)



(b)

Figure 2.11: Time performance of different aligners on the two microbiome experiments. In (a), the results are averaged over the three alignment processes for the samples covid19, sars, and bat200, each having $\sim 1M$ paired-end reads. The performance shown in (b) is for aligning reads in the mock sample simulated from SRR10948222 with $5M$ paired-end reads. As shown in the bulk experiment, the alignment for *Bowtie2* increases when asking for more alignments per read while the other tools show a constant alignment time scaling over number of reads. The dashed area shows fraction of the time spent purely on aligning reads where the remaining portion is the time required for index loading. PuffAligner is the fastest tool in this experiment, yet most of its time is still dedicated to loading the index.

2.2.15 Scalability

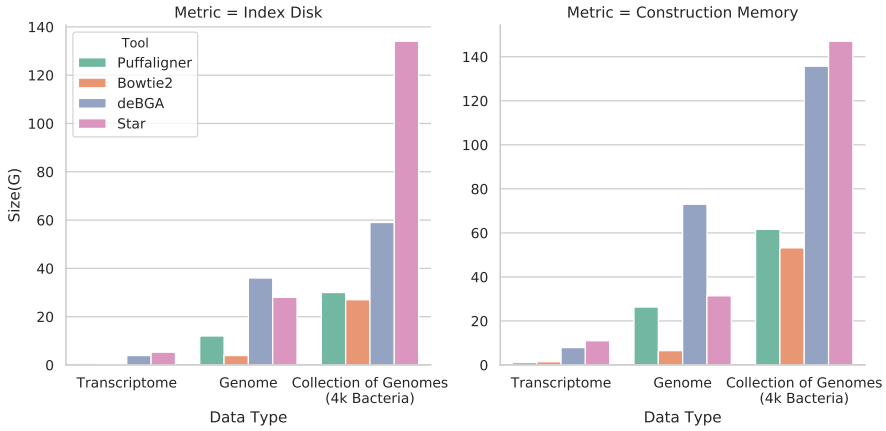


Figure 2.12: Scalability of different tools over the final index disk space, construction memory, for three different datasets, human transcriptome (gencode version 33), human genome (GRCh38 primary assembly), and collection of genomes (4000 random bacterial complete genomes). All tools are run with 16 threads.

Figure 2.13 and fig. 2.12 represents how the construction time and index size of each tool scales over different types of sequences. The trend shows the effect of database size as well as redundancy and sequence similarity on the scalability of each of the tools. Tools such as PuffAligner and deBGA, which build a de Bruijn graph based index on the input sequence, specifically compress similar sequences into unitigs and therefore scale well for databases with high redundancy such as microbiomes. It is worth mentioning that *Bowtie2* requires a switch from a 32-bit index to a 64-index as the total count of the input bases increases, which is another reason why the size is growing super-linearly.

2.2.16 Discussion & Conclusion

In this paper we introduce PuffAligner, an aligner suitable for the contiguous alignment of short-read sequencing data. We demonstrate its use in aligning DNA-seq reads to the

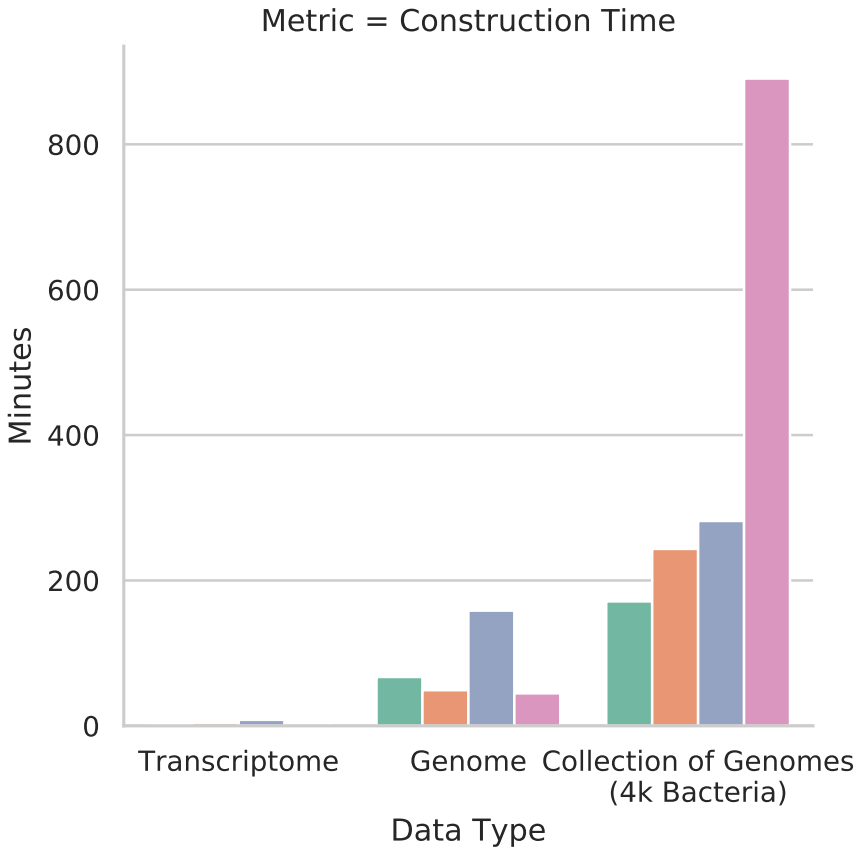


Figure 2.13: Scalability of different tools over the construction running time for three different datasets, human transcriptome (gencode version 33), human genome (GRCh38 primary assembly), and collection of genomes (4000 random bacterial complete genomes). All tools are run with 16 threads.

genome of a single species, aligning RNA-seq reads to the transcriptome, and aligning DNA-seq reads from metagenomic samples to a large collection of references. It is built on top of the Pufferfish index, which constructs a colored compacted de Bruijn graph using the input reference sequences. PuffAligner begins read alignment by collecting unique maximal exact matches, querying k-mers from the read in the Pufferfish index. The aligner then chains together the collected uni-MEMs using a dynamic programming approach, choosing the chains with the highest coverage as potential alignment positions for the reads. Finally, PuffAligner is able to efficiently compute alignment, exploiting

information from long matches in the chains and making use of an alignment cache to avoid redundant work.

We compared the accuracy and efficiency of PuffAligner against two widely-used alignment tools, *Bowtie2* and STAR, that perform unspliced and (optionally) spliced alignments of reads, respectively. We also compare the results against deBGA, an aligner that also utilizes an index built over the compacted de Bruijn graph.

We analyze the performance of these tools on both simulated and experimental DNA and RNA sequencing datasets. The accuracy of PuffAligner is comparable to *Bowtie2*, which exhibits very high alignment. PuffAligner generally performs better than STAR and deBGA (though, unlike STAR, none of these other tools currently support spliced read alignment). In terms of speed and memory, PuffAligner reaches a tradeoff between the relatively high memory usage of STAR and deBGA and the slower speed of *Bowtie2*. Hence, while the memory requirement of PuffAligner is more than that of *Bowtie2*, the speed gain is significant. In the tests performed in this manuscript, PuffAligner is almost always the fastest tool (with the exception being that STAR is faster when aligning unspliced DNA-seq reads to a single human genome).

An additional advantage of the Pufferfish index utilized in PuffAligner is that it can be built on a mixed collection of genomes, transcriptomes, or both. This feature is already utilized in a specific pipeline for RNA-seq quantification that makes use of a joint index over the genome and transcriptome [2]. The analysis shows that specificity of alignments in such a case can be improved by filtering from quantification reads that are better aligned to some genomic locus that is not present in the transcriptome.

Furthermore, the nature of the Pufferfish index, that explicitly factorizes out highly-

repetive sequence, coupled with the fast (and repetition-aware) alignment procedure of PuffAligner makes it a particularly useful for indexing and aligning to a highly similar collection of sequences. This potentially makes it a good match for metagenomic analyses.

We have provided a proof of concept for such a PuffAligner-based metagenomic analysis pipeline, and plan to build a more sophisticated and fully-featured metagenomic analysis framework around PuffAligner in the future.

Accession ID	Alignment Mode	Tool	Spearman	MARD	MAE	MSLE
SRR11283975 Jiandong Peninsula	Primary	PuffAligner	0.71	0.024	0.426	0.044
		Bowtie2	0.615	0.04	0.640	0.071
		STAR	0.727	0.02	0.406	0.039
		deBGA	0.274	0.521	106.776	3.788
	Up to 20	PuffAligner	0.942	0.003	0.074	0.002
		Bowtie2	0.909	0.005	0.049	0.004
		STAR	0.946	0.003	0.087	0.002
		deBGA	0.277	0.489	101.385	3.366
	Up to 200	PuffAligner	0.979	0.001	0.068	0.000
		Bowtie2	0.97	0.002	0.039	0.001
		STAR	0.951	0.003	0.086	0.001
		deBGA	0.278	0.483	100.961	3.293
	Best strata	PuffAligner	0.979	0.001	0.063	0.000
		STAR	0.951	0.003	0.086	0.001
SRR11496426 Uzon Caldera	Primary	PuffAligner	0.568	0.112	32.552	0.953
		Bowtie2	0.53	0.14	38.062	1.101
		STAR	0.559	0.118	31.823	0.825
		deBGA	0.367	0.566	115.882	3.569
	Up to 20	PuffAligner	0.789	0.03	7.426	0.239
		Bowtie2	0.74	0.042	10.834	0.304
		STAR	0.713	0.049	6.939	0.165
		deBGA	0.368	0.554	109.289	3.317
	Up to 200	PuffAligner	0.865	0.017	5.635	0.105
		Bowtie2	0.879	0.015	7.208	0.134
		STAR	0.724	0.045	6.496	0.133
		deBGA	0.369	0.549	108.986	3.273
	Best strata	PuffAligner	0.85	0.019	5.571	0.092
		STAR	0.723	0.046	6.544	0.134
SRR10948222 Mojave Desert	Primary	PuffAligner	0.69	0.028	1.390	0.075
		Bowtie2	0.58	0.053	2.910	0.153
		STAR	0.727	0.023	1.493	0.048
		deBGA	0.28	0.616	656.080	6.530
	Up to 20	PuffAligner	0.9	0.006	0.400	0.006
		Bowtie2	0.85	0.01	0.220	0.012
		STAR	0.929	0.004	0.303	0.002
		deBGA	0.28	0.573	637.600	5.650
	Up to 200	PuffAligner	0.97	0.002	0.360	0.001
		Bowtie2	0.99	0.001	0.190	0.000
		STAR	0.929	0.004	0.299	0.002
		deBGA	0.28	0.571	637.830	5.550
	Best strata	PuffAligner	0.97	0.002	0.36	0.001
		STAR	0.929	0.004	0.3	0.002

Table 2.10: Accuracy of abundance estimation with Salmon using alignments reported by each aligner for the mock samples simulated from the real samples with accession IDs SRR10948222, SRR11283975 and SRR11496426. We have run all the aligners in three main modes; allowing only one best alignment with ties broken randomly (Primary), up to 20 alignments reported per read, and up to 200 alignments reported per read. PuffAligner and STAR also support a mode that allows reporting all equally best alignments (bestStrata).

Chapter 3: Improved Data-Driven Likelihood Factorizations for Transcript Abundance Estimation [1]

3.1 Introduction

Shortly after the RNA-seq assay became popular as a tool for transcriptome profiling and quantification, the computational community began developing principled inference methodologies to allow accurate transcript-level quantification in the presence of multi-mapping reads. Tools such as *Cufflinks* [76], *RSEM* [15], *mmseq* [17] and *IsoEM* [18] provided statistical models by which transcript-level abundance estimates could be inferred. These methodologies principally rely on maximum likelihood estimation to infer the transcript abundances that would be most likely given the observed data (i.e., the alignments of the sequenced fragments to the underlying genome or transcriptome). Bayesian methodologies such as *BitSeq* [77] and *Tigar* [78] were also developed and adopt different inferential approaches varying from fully-Bayesian approaches like collapsed Gibbs sampling [77] to approximate inference approaches like variational Bayesian optimization [78, 79, 80].

These methods vary widely in their details, though adopt a similar generative model of the underlying RNA-seq experiment; one which is well-represented by the generative model of *RSEM* [15, 36]. In this paper, we shall refer to this as the *full model*. It is a

[1] This work is presented in the proceedings of ISMB 2017

generative model of an RNA-seq experiment that considers the likelihood of observing a collection of alignments as dependent upon the parameters of interest (i.e., the transcript abundances), as well as the details of each alignment of a sequenced fragment to the reference transcriptome. In this way, the full model provides very high fidelity, and is capable of incorporating a tremendous amount of information into the inference procedure (e.g., the implied fragment length under each alignment, details about the alignment and the fragment's quality values, the probability of different start positions for the sampled fragment, etc.).

Unfortunately, however, this means that straightforward inference procedures that adopt this full model scale in the number of considered alignments *per-iteration*. For example, a 30 million fragment RNA-seq experiment may produce 100 million fragment alignments, all of which are considered by the inference procedure in each of its (typically) hundreds to thousands of iterations. This approach, then, poses two problems. First, inference is typically slow since each iteration must consider a large number of independent probabilities. Second, so as to prevent the inference algorithm from becoming even slower, these per-alignment probabilities are typically retained in memory, which can lead memory requirements to scale linearly with the number of alignments. One approach to mitigate the cost associated with optimizing the *full model* is to alter the actual inference algorithm that is used. For example, *eXpress* [16] uses an online-EM algorithm, rather than a batch-EM algorithm (by default), to infer transcript abundances. This eliminates the need to cache alignments in memory for efficiency, resulting in constant memory usage. However, a single pass over the data is not always sufficient to achieve the same accuracy as methods that run batch algorithms to convergence.

One of the more popular approaches for reducing the computational burden and speeding up the inference procedure is to form an approximate factorization of the likelihood function (see Section 3.2.1). For example, *mmseq* introduced a notion of fragment equivalence classes, which treats as equivalent any fragments that align to exactly the same set of transcripts. This leads to a likelihood function in which the counts of fragments compatible with subsets of transcripts serve as sufficient statistics. The likelihood defined over these counts is typically orders of magnitude faster to evaluate, but it can discard certain fragment-level information encoded in the alignments. Distinct but related notions of equivalence classes were also introduced by Salzman et al. [81] and Nicolae et al. [18].

Because of the computational economy of this approximate factorization, it (or similar variants) were later adopted by new lightweight approaches for transcript quantification like *Sailfish* [11, 12] and *kallisto* [14]. By coupling a very fast inference approach with techniques that removed the requirement of computing traditional alignments for each sequenced fragment, such approaches reduced the time required to obtain transcript-level quantification estimates by orders of magnitude over existing approaches. These lightweight methods have proven an important and popular development. Recently, Patro et al. [13] introduced a new lightweight approach, Salmon, that uses a "dual-phase" inference algorithm, which combines an online stochastic inference method with an efficient offline inference algorithm. While adopting a similar approximate factorization as *mmseq*, *Sailfish* and *kallisto*, Salmon also maintains aggregate (i.e., average) weights per equivalence class that allow retaining some information about fragment-level probabilities during the offline inference algorithm. However, this information is restricted to a single scalar value per transcript-equivalence class pair, and so is necessarily limited in its ability to represent

the full model with complete fidelity.

In this paper, we argue that the dual-phase algorithm introduced by Salmon allows one to derive a data-driven approximate factorization of the full model likelihood function. The online phase of the algorithm assess each individual fragment probability, and uses this information to build a highly-reduced but accurate proxy for the *full model* likelihood that can be efficiently optimized during the offline phase. While only slightly increasing the per-iteration cost of the underlying inference algorithm, this data-driven factorization can represent the fragment-level likelihood function with much higher fidelity. In fact, we demonstrate that a data-driven likelihood factorization can produce transcript-level abundance estimates that display essentially no loss in accuracy compared to what is obtained under the full model. Thus, such a factorization is preferable to the more common compatibility-based approximate factorization, since it can provide a substantial improvement in accuracy while introducing only a small increase in the computational burden. We note that we focus in this paper on how to factorize the likelihood function, and not, specifically, the algorithm by which this function is best optimized. Thus, we expect the approaches we introduce here to easily translate to other likelihoods or optimization approaches; e.g., to variational Bayesian optimizations [78], or natural gradient-based optimization algorithms [80]. Our data-driven factorizations are incorporated into the Salmon transcript quantification tool.

3.2 Approach

3.2.1 The likelihood function and its factorizations

We begin by considering the basic generative model laid out by Li et al. [15]. We consider a transcriptome T to consist of a set of M transcripts, $t_1, t_2, \ldots, t_M$. In a given sample, there are c_i copies of the i^{th} transcript. Further, we can assign to each transcript its length, such that the length of t_i is given by ℓ_i. The generative model of an RNA-seq experiment states that the expected number of fragments sequenced from each transcript type t_i is proportional to the total number of sequencable nucleotides that it constitutes in the underlying mixture — that is we expect that $\alpha_i \propto \eta_i = \frac{c_i \cdot \ell_i}{\sum_j c_j \cdot \ell_j}$ — where α_i is the number of fragments drawn from transcripts of type t_i. Assuming that each fragment is drawn independently, the likelihood of a collection $\mathcal{F}$ of fragments can be written as:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{F}) = \prod_{f_j \in \mathcal{F}} \sum_{i=1}^{M} \Pr(t_i \mid \boldsymbol{\theta}) \Pr(f_j \mid t_i), \tag{3.1}$$

where $\boldsymbol{\theta}$ denotes the parameters of the model, which are the underlying transcript abundances. We note that, throughout this manuscript, we use the term "fragment" as a generic term which is represented by a single read (in single-end protocols) and a read pair (in paired-end protocols). The methods we propose in Section 3.3 work only in terms of the conditional fragment probabilities, and so are equally-applicable in both single and paired-end protocols, though the definition of these conditional probabilities will be protocol dependent.

The primary quantity of interest, with respect to the factorizations being proposed

in this paper, are the $\Pr(f_j \mid t_i)$ terms — that is, the conditional probability of drawing a particular fragment f_j, given transcript t_i. This term encodes, given parameters of the model and experiment, how likely it is to observe a specific fragment f_j arise from transcript t_i. Many terms can be included in such a conditional probability, some common terms include:

$$\Pr(d_j \mid f_j, t_i) = \frac{\Pr_D(d_j)}{\sum_{k=1}^{\ell_i} \Pr_D(k)}, \tag{3.2}$$

the probability of observing a mapping of implied length d_j for f_j given that it derives from t_i, where $\Pr_D(k)$ is the probability of observing a fragment of length k under the empirical fragment length distribution D;

$$\Pr(p_j \mid d_j, f_j, t_i) = \frac{1}{\ell_i - d_j + 1}, \tag{3.3}$$

the probability of a observing a mapping starting at position p_j for fragment f_j given that it has implied length d_j and is derived from t_i;

$$\Pr(o_j \mid f_j, t_i) = \begin{cases} \begin{cases} 0.5 & \text{if unstranded} \end{cases} \\ \begin{cases} 1.0 & \text{if compatible orientation} \\ \epsilon & \text{if incompatible orientation} \end{cases} & \text{if strand-specific} \end{cases}, \tag{3.4}$$

the probability of observing a mapping with a specific orientation o_j (i.e., forward or antisense) with respect to the underlying transcript for f_j, given t_i, ϵ (a user-defined constant), and knowledge of the underlying protocol, and

$$\Pr\left(a_j \mid f_j, o_j, d_j, p_j, t_i\right), \tag{3.5}$$

the probability of observing the particular alignment (e.g., `CIGAR` string) a_j for f_j given it is sampled from transcript t_i, has orientation o_j, implied length d_j and starts at position p_j—such a probability is calculated from a model of alignments, like those presented in [13, 15, 16].

In fact, one can conceive of many such general models of "fragment-transcript agreement" [13]. The framework we propose in Section 3.3 can naturally account for such conditional probabilities that one might consider as part of $\Pr\left(f_j \mid t_i\right)$. However, in this manuscript, we consider that $\Pr\left(f_j \mid t_i\right)$ is simply the product of the conditional probabilities defined in Equations (3.2) to (3.5), appropriately normalized.

3.2.2 Equivalence classes and approximate likelihood factorizations

Here, we describe the most common definition of fragment equivalence classes, and explain how they are used to derive an approximate factorization of the likelihood function, we adopt a notation similar to Patro et al. [13].

Let $A\left(\mathcal{T}, f_j\right)$ be the set of all alignments of fragment f_j to the transcriptome $\mathcal{T}$ and let $\Omega\left(f_j\right) = \{\langle i, t_i\rangle \mid t_i \in A\left(\mathcal{T}, f_j\right)\}$ be the tuple of transcripts to which f_j maps—considering the t_i are ordered by their index i. Fragment equivalence classes are defined in terms of the equivalence relation $\sim$, such that $f_m \sim f_n$ if and only if $\Omega\left(f_m\right) = \Omega\left(f_n\right)$. Thus, fragment equivalence classes consider as equivalent (for the purposes of inference), sequenced fragments that align to the same set of transcripts. We will refer to $\Omega\left(f_j\right)$ as

the *label* of f_j for all $f_j \in [q]$, where $[q]$ is the equivalence class to which f_j belongs. We will also refer to $\Omega([q]) = \Omega(f_j), \forall f_j \in [q]$ as the label of f_j's equivalence class. Finally, it will be convenient to define the total size of each such equivalence class as $N^q = |[q]|$, which is the total number of equivalent fragments in the class $[q]$.

Now, we can write the equivalence class-based approximation to the likelihood function as:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{F}) \approx \prod_{[q] \in \boldsymbol{\mathcal{C}}} \left(\sum_{\langle i, t_i \rangle \in \Omega([q])} \Pr(t_i \mid \boldsymbol{\theta}) \cdot \Pr(f \mid [q], t_i) \right)^{N^q}, \tag{3.6}$$

where $\boldsymbol{\mathcal{C}}$ is the set of all equivalence classes, and $\Pr(f \mid [q], t_i)$ is the probability of generating a fragment f given that it comes from equivalence class $[q]$ and transcript t_i. The key to the efficiency of likelihood evaluation (or optimization) under this factorization, is that the probability $\Pr(f \mid [q], t_i)$ is assumed to be identical for each of the N^q fragments in each equivalence class $[q]$—hence, we do not subscript f in Equation (3.6). This allows one to replace the product over all fragments f_j in Equation (3.1) with a product over all equivalence classes in Equation (3.6). The approximation, of course, stems from the fact that, under the full model, a fragment f_j may have a probability $\Pr(f_j \mid t_i)$ that is arbitrarily different from $\Pr(f \mid [q], t_i)$. Moreover, the most common approximations, like those adopted in *mmseq*, *Sailfish*, and *kallisto* consider this probability to be fixed and essentially independent of any fragment-level information (e.g., it is set to one divided by the effective length of t_i).

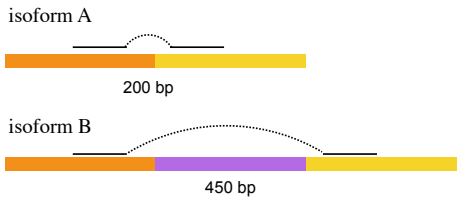


Figure 3.1: A fragment multimapping between two different isoforms (A,B) of a gene. Depending on the parameters of the fragment length distribution of the underlying sample, either multi mapping locus could be more probable *a priori*. Under the approximate likelihood factorization that considers only compatibility-based equivalence classes, such information is necessarily hidden from the resulting inference algorithm. We note that, of course, such multi-mapping can also happen between different genes (e.g., paralogs).

3.2.3 What approximate factorizations elide

Figure 3.1 provides an illustrative example why considering conditional fragment probabilities can be important. Consider a multi-isoform gene, and a single fragment f_j, which aligns equally well (i.e., the sequence of both ends of the fragment match the sequence of the underlying transcripts equally well) to isoforms A and B of this gene. If we consider only transcript-fragment compatibility, then both of the alignments illustrated in fig. 3.1 are delineated only in that isoform A has fewer potential start locations. However, considering the implied length of this fragment, given the expected insert size distribution of the experiment (either provided as input to the model, or inferred from the collection of previously aligned fragments), can provide strong evidence that one or the other of these isoforms was more likely to have generated f_j. For example, were the mean of the fragment length distribution 250, then we would expect isoform A to be much more likely to have generated f_j. Conversely, were the mean of the fragment length distribution 400, then we would expect that, in fact, isoform B might have been more likely to generate this fragment. Standard (i.e., compatibility-based) approximate factorizations of

the full likelihood function into equivalence classes discard (or collapse) this fragment-level information. For example, compatibility-only factorizations of the likelihood into equivalence classes simply treat $\Pr(d_j \mid f_j, t_i)$ as equal for all transcripts in the equivalence class to which fragment f_j belongs. The factorization adopted by Salmon attempts to maintain slightly more information by computing these conditional probabilities and averaging them; maintaining a single extra scalar per transcript-equivalence class pair, that represents the conditional probability that any fragment coming from a particular equivalence class would derive from a particular transcript. Though this maintains some extra information, it is not always enough to faithfully approximate the full-model likelihood function.

Below, we describe a data-driven approach that allows for a much more faithful representation of the *full model* likelihood function, while still greatly reducing the amount of information that must be maintained for inference. A broad overview of how these factorizations relate to each other is given in Figure 3.2, and the specific factorizations are described in more detail below.

3.3 Methods

As illustrated in Figure 3.2 and described above, the approximations that rely on *compatibility-based* factorizations can discard information that may be useful for correct transcript abundance estimation. Specifically, such notions of equivalence classes sacrifice per-fragment information encoded in the conditional probabilities $\Pr(f_j \mid t_i)$. We propose here alternative notions of equivalence classes that take into account both the transcripts with which a fragment is compatible, as well as the vector of conditional probabilities

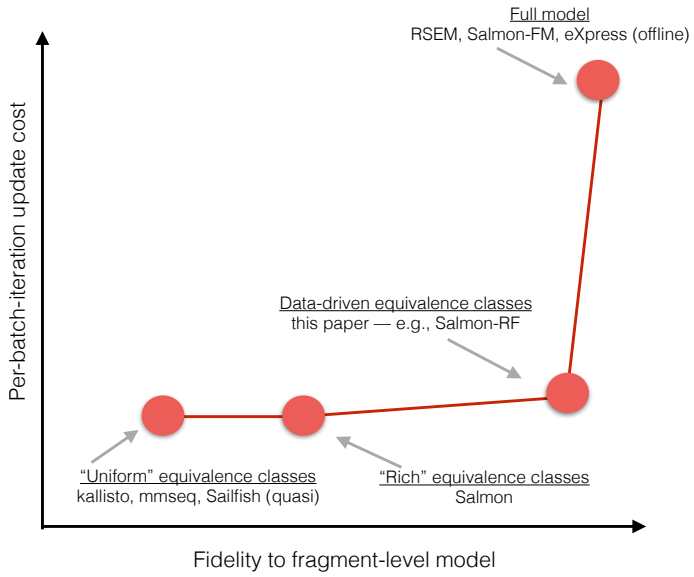


Figure 3.2: There is a conceptual tradeoff between the computational efficiency of an inference technique, and the fidelity with which it models the full, fragment-level likelihood function. *kallisto*, *Sailfish* (using quasi-mapping [12]) and *mmseq* simply consider the compatibility of fragments with transcripts, and thereby discard the conditional fragment-level probabilities completely. Salmon collapses the fragment-level conditional probabilities to a single scalar (their average value) per-equivalence class; this recovers some of the fidelity lost in the other approaches, but can still discard useful fragment-level information. Approaches that consider each fragment independently in each round of the optimization algorithm (e.g. *RSEM* and *Salmon-FM* and *eXpress* (offline)) sacrifice no fidelity, but each iteration scales with the total number of aligned / mapped fragments. Our proposed data-driven clustering approach (*Salmon-RF*) captures most of the important fragment-level probabilities of the *full model*, while retaining an update time very similar to Salmon's standard model in its offline rounds. The online rounds of Salmon and *eXpress* are not directly comparable to the batch rounds considered in this figure (they update the parameters more frequently), but they do consider the conditional probability of each fragment individually.

that encodes how likely the fragment is to have been sequenced from each such transcript. That is, these factorizations account both for the set of transcripts $t_1, \ldots, t_k$ to which a fragment f_j maps, as well as the conditional probabilities $\Pr(f_j \mid t_1), \ldots, \Pr(f_j \mid t_k)$ that f_j was sampled from each of these transcripts. Our approach is agnostic to how $\Pr(f_j \mid t_i)$ is computed, but, as stated in Section 3.2.1, we consider here each conditional probability to be the product of Equations (3.2) to (3.5), appropriately normalized.We accomplish this by defining new equivalence relations over fragments that consider and summarize these conditional probabilities in a data-driven manner.

As one divides each equivalence class into smaller sub-classes of fragments, the factorized likelihood approaches the likelihood (and hence fidelity) of the *full model*. Conversely, as the number of equivalence classes increases so does the complexity of evaluating and optimizing the likelihood.

Here, we introduce two different factorization methods that refine the *compatibility-based* notion of equivalence classes. These approaches are a refinement in the strict sense that each sub-cluster of fragments that fall within the newly-defined equivalence classes align to the same set of transcripts as all other fragments in the original, *compatibility-based* definition of the equivalence class. However, in these factorizations, the conditional fragment probabilities (with respect to the set of transcripts) tend to exhibit smaller distance to mean; i.e., the approximate weight used to summarize the conditional probability of all fragments within these refined equivalence classes is much closer to the individual conditional probabilities of all the fragments placed in the class. Subsequently, this leads to a more accurate approximation of the likelihood function. Moreover, we find that only a small number of such refined equivalence classes is required to approximate the

full likelihood very closely, meaning that the computational complexity of evaluating and optimizing the likelihood function is very close to what is required when considering the original *compatibility-based* equivalence class factorization (table 3.3).

3.3.1 Rank-based factorization

We call the first factorization method that we consider to refine the notion of equivalence classes the "rank-based factorization". We consider all transcripts to which a fragment aligns, and sort the transcripts based on the conditional probability values of the fragment given each transcript. Then, the equivalence class for a fragment is determined by the set of transcripts to which it maps, *and* the rank-order of the conditional probabilities for this fragment given those transcripts. For instance, consider $1{,}000.000$ fragments which all align to the transcripts t_1 and t_2, where 250 of these fragments align to t_1 with a higher conditional probability than that with which they align to t_2 (and vice-versa for the rest). In this case, the rank-based equivalence relation will induce 2 equivalence classes (whereas the *compatibility-based* relation would have induced 1), the first 250 fragments will become members of one equivalence class with label $\{\langle 1, t_1\rangle, \langle 2, t_2\rangle\}$ and the rest will be assigned to another equivalence class with the label $\{\langle 1, t_2\rangle, \langle 2, t_1\rangle\}$. As with the original notion of *rich* equivalence classes in Salmon [13], a single scalar value per transcript is saved in each equivalence class, which is the mean of all conditional probabilities of the fragments given each transcript. Of course, in this factorization, the total number of equivalence classes is typically larger than the number of *compatibility-based* equivalence classes. Formally, we define the rank-based equivalence relation $\sim_<$ as

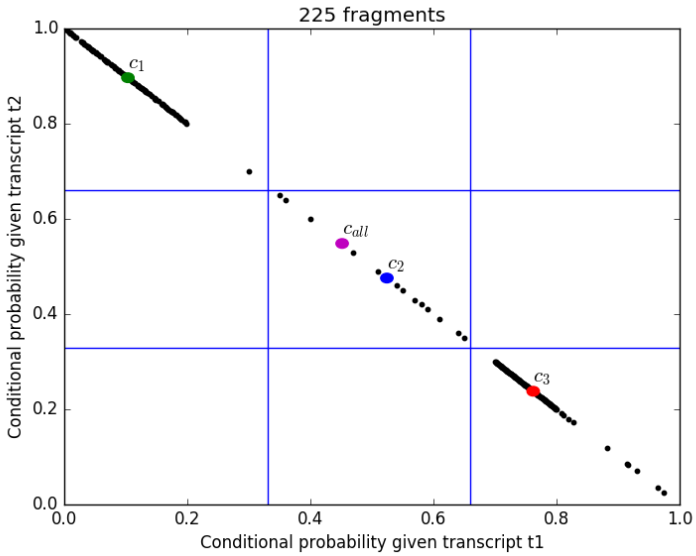


Figure 3.3: Factorizing an equivalence class consisting of 225 fragments and 2 transcripts into k=3 bins. Each dot represents one fragment. The vertical lines indicate borders of bins for transcript t_1 and the horizontal lines show borders of bins for transcript t_2. The purple circle with label c_{all} shows the center for original equivalence class. The rest of the circles are indicators of the centers for each cluster after the factorization.

follows: let $r\left(f, \{\langle i_1, t_{i_1}\rangle, \langle i_2, t_{i_2}\rangle, \ldots, \langle i_j, t_{i_j}\rangle\}\right)$ be a function that returns a permutation σ of $\left(t_{i_1}, t_{i_2}, \ldots, t_{i_j}\right)$ such that $\Pr\left(f \mid t_{\sigma_1}\right) \leq \Pr\left(f \mid t_{\sigma_2}\right) \leq \cdots \leq \Pr\left(f \mid t_{\sigma_j}\right)$, with ties broken arbitrarily in favor of the transcript having the smaller index. We define two fragments f_m and f_n to be equivalent ($f_m \sim_< f_n$) if and only if $\Omega\left(f_m\right) = \Omega\left(f_n\right)$ and $r\left(f_m, \Omega\left(f_m\right)\right) = r\left(f_n, \Omega\left(f_n\right)\right)$.

3.3.2 Range-based factorization

We consider a second factorization approach that we call "range factorization" (*Salmon-RF*). In this approach, we seek equivalence classes that have fragments which both align to the same set of transcripts *and* which have similar conditional probabilities with respect to these transcripts. To motivate this approach, consider, first, the case

of two fragments that have exactly the same conditional probabilities for the same set of transcripts, then one can safely group them together without any loss of accuracy with respect to the original likelihood function. In fact, this is the equivalence relation proposed by Nicolae et al. [18]. However, this particular factorization can have a negative impact on performance since most of the time probabilities of fragments are not exactly proportional. Hence, this can lead to a model similar to the full model that considers all fragment-transcript likelihood values. However, we can compromise the "exact" proportionality of probabilities for the sake of performance. Instead of clustering fragments that have exactly proportional probabilities, we place fragments with the same conditional probability "range" into the same equivalence class. We first divide the valid range of probabilities $[0, 1]$ into k bins, and then consider two conditional probabilities equal if their values are in the same bin. Two fragments are considered equivalent under this definition, denoted $\sim_r$, if they fall into the same set of bins with respect to all transcripts to which they align. Formally, let $b_k\left(f, \{\langle i_1, t_{i_1}\rangle, \langle i_2, t_{i_2}\rangle, \ldots, \langle i_j, t_{i_j}\rangle\}\right)$ be a function that returns a vector of bin values (one for each transcript, and each between 0 and $k-1$). We define two fragments f_m and f_n to be equivalent ($f_m \sim_r f_n$) if and only if $\Omega\left(f_m\right) = \Omega\left(f_n\right)$ and $b_k\left(f_m, \Omega\left(f_m\right)\right) = b_k\left(f_n, \Omega\left(f_n\right)\right)$.

We can tune the parameter k to tradeoff of the number of such equivalence classes versus the accuracy they provide. As k approaches infinity (or, rather, machine precision), the fidelity provided by this factorization approaches that of the full model, because all fragments will end up in either single-member equivalence classes, or in equivalence classes of fragments having conditional probabilities exactly proportional to theirs. On the other hand, as k gets smaller, the number of clusters gets closer to a small constant

times the number of *compatibility-based* equivalence classes, but each cluster consists of fragments with the wider range of conditional probabilities. In this approach, we do not simply replace each conditional probability with the center of the bin into which it falls. Rather, for each bin, we record the sum and a total number of conditional probabilities stored in this bin. After processing all fragments, the centroid of each bin is computed and used as the representative conditional probability for this bin. This model is a natural extension of the rich equivalence class model used in Salmon, and the models coincide when $k = 1$. Throughout this paper, range-based equivalence classes have a number of bins equal to $4 + \left\lceil \sqrt{|\Omega([q])|} \right\rceil$.

Figure 3.3 provides a good example of this factorization and its impact on the average of conditional probabilities for each transcript. There are 225 fragments that all are aligned to the two transcripts in this equivalence class. Each dot represents a fragment with its x value equal to $\Pr(f \mid t_1)$ and y value equal to $\Pr(f \mid t_2)$. c_{all} shows the average value of conditional probabilities of all fragments for transcript t_1 and t_2. As can be observed, the deviation of c_{all} from many of the conditional probabilities is large since the conditional probabilities are widely distributed over the range from zero to one. However, when we divide the range into three bins and then separate fragments based on the bin into which their conditional probabilities fall, we obtain three clusters containing fragments whose within-cluster conditional probabilities fall into much smaller ranges. So, in this case, all fragments that have the same bin for their conditional probability given t_1 and their conditional probability given t_2 end up in the same cluster. Lines show the borders of each bin and colored circles show the centroids used to represent the conditional probabilities in each bin. In this case, we expect to obtain results closer to

the *full model*; yet, the number of clusters over which one must iterate to apply the EM algorithm is sill much smaller than the total number of fragments (see Table 3.3).

Though we have implemented and experimented with both of these alternative factorizations, in this paper we will focus on the *range-based* factorization, as we observe that it almost always provides a better approximation of the likelihood than the *rank-based* factorization.

3.4 Results

We test the ability of our proposed factorization to improve the approximation of the *full model* likelihood on both synthetic and experimental data. We demonstrate that, as expected, the *range-based* factorization almost always provides a very good approximation of the *full model* likelihood. Interestingly, we also observe that it sometimes leads to a slightly more accurate solution than when no factorization is applied (i.e., when the likelihood is evaluated for each fragment independently). Though we have not investigated this in depth, it is likely that, in some cases, a small degree of smoothing of the conditional probabilities can lead to a more stable and accurate solution.

We consider both small-scale and transcriptome-wide simulated data. In Section 3.4.1 we consider simulations over the transcripts from families of paralogous genes. Such situations represent the most challenging abundance estimation problems for transcript quantification tools since high levels of multi-mapping are prevalent. We conduct the simulations over many random settings of the abundances of these transcripts, and look at how well different methods are able to recover the true abundances at different average

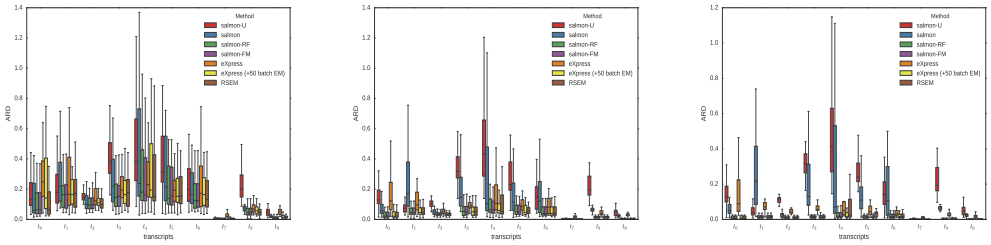


Figure 3.4: Applying different methods of transcript abundance estimation in alignment mode on two sets of data in 3 depth of fragment sequencing. Top (a) are all isoform transcripts of gene `RAD51`. The bottom (b) is from transcripts of four different paralogs of `RAD51`, `RAD51B`,`RAD51C`, `RAD51D`. In each row the left most plot refers to experiment with counts of 1X coverage, the middle one to 10X and the most right plot refers to the experiment with fragment counts of 100X coverage. Distribution of ARDs on isoforms of `RAD51`.

coverage levels. We directly observe how, in the most adversarial situations, the proposed factorization allows us to recover important information that leads to improved abundance estimates.

In Section 3.4.2 we explore the effect that different factorizations have on abundance estimates transcriptome-wide. Here, we observe that, while the data-driven factorizations lead to improved abundance estimates, the differences between methods becomes much smaller, since the statistics are aggregated over the entire transcriptome and since many transcripts fall into the "easy" case of abundance estimation. The differences between methods, while still moderate, are larger when we restrict our assessment to a more difficult subset of transcripts.

Finally, in Section 3.4.3, we examine the effect of different factorization methods over experimentally sequenced data. We explore how closely different factorizations approach the abundance estimates derived by *RSEM*—though we note (as observed in some of the simulated data) that *RSEM* is not necessarily more accurate than the alternative methods or factorizations.

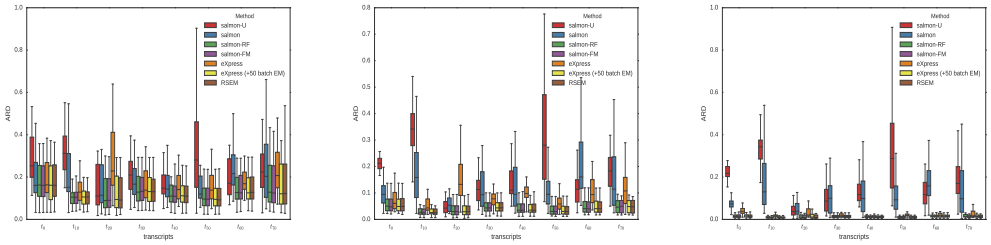


Figure 3.5: Applying different methods of transcript abundance estimation in alignment mode on two sets of data in 3 depth of fragment sequencing. Top (a) are all isoform transcripts of gene `RAD51`. The bottom (b) is from transcripts of four different paralogs of `RAD51`, `RAD51B`,`RAD51C`, `RAD51D`. In each row the left most plot refers to experiment with counts of 1X coverage, the middle one to 10X and the most right plot refers to the experiment with fragment counts of 100X coverage. Distribution of ARDs on isoforms of a family of 4 `RAD51` paralogs.

In Sections 3.4.1 to 3.4.3 we consider the transcript abundance estimates generated by *RSEM*, *eXpress* (both in default mode and with 50 batch EM rounds) and variants of Salmon (using different factorizations). We focus on the performance of these tools when quantifying abundances using alignments, instead of mappings [12]. We keep the input data as close as possible, since the purpose of this paper is not an investigation of the effect of alignment versus mapping on expression estimation, but rather the effect of the factorization of the likelihood and how that factorization affects inference. We noticed that, regardless of the factorization used, there was a small but persistent gap between non-alignment-based tools (*kallisto* and mapping-based variants of Salmon) compared to *RSEM* and alignment-based variants of Salmon on the *RSEM-sim* data. It is not clear that this is due to any fundamental superiority of alignment compared to mapping, but rather, may be a result of the fact that the specific error model, learned by *RSEM* and used to simulate reads in *RSEM-sim*, acts as a "side-channel" of information for alignment-based approaches. However, this question, though outside the scope of this paper, deserves further consideration and analysis.

In the supplementary material, we explore the effect of these factorizations on mapping-based solutions by comparing different variants of mapping-based Salmon with *kallisto* (which only allows using pseudoalignment for quantification).

Alternative factorization variants: Salmon (i.e., without any modification) uses a *compatibility-based* notion of equivalence classes called "rich" equivalence classes. Under this notion, the equivalence classes themselves are *compatibility-based*, but each transcript-equivalence class pair is associated with a scalar weight which is computed as the mean conditional probability of all fragments in this equivalence class to derive from this transcript. We also consider a variant of Salmon (denoted as *Salmon-U* herein) that adopts a purely *compatibility-based* notion of equivalence classes. That is, it stores no extra information about the conditional probability of deriving the fragments in each equivalence class from the different transcripts, and during inference considers only that $\Pr(f \mid t) = \Pr(p \mid f, t) = 1/\tilde{\ell}_t$, where $\tilde{\ell}_t$ is the effective length of transcript t and is defined as $\tilde{\ell}_t = \ell_t - \mu_d^{\ell_t}$. $\mu_d^{\ell_t}$ is the mean of the truncated empirical fragment length distribution as described in [13].

We also consider a variant of Salmon, *Salmon-FM*, that performs no additional factorization. Instead, like *RSEM*, it considers each fragment and its relevant conditional probabilities independently. In this case, the only difference between *Salmon-FM* and *RSEM* is that the former computes the conditional fragment probabilities using an *online* stochastic inference algorithm, while *RSEM* recomputes the conditional fragment probabilities after updating auxiliary model parameters during the first 10 iterations of an offline (i.e., batch) EM procedure.

Finally, we consider a variant of Salmon, *Salmon-RF*, that uses the range-factorization

described in Section 3.3.2 to generate equivalence classes based on $\sim_r$ and compute the associated weights.

We use both the mean absolute relative difference (MARD) and Spearman correlation to assess performance. We define the absolute relative difference (ARD) as:

$$\text{ARD}_i = \begin{cases} 0 & \text{if } x_i + y_i = 0 \\ \frac{|x_i - y_i|}{(x_i + y_i)} & \text{otherwise} \end{cases}, \tag{3.7}$$

Where y_i is the estimated number of reads originating from t_i and x_i is the true (or assumed) number of reads originating from t_i. The MARD is simply defined as $\text{MARD} = \frac{1}{M}\sum_{i=1}^{M} \text{ARD}_i$, where M is the total number of transcripts.

Experimental setup and software parameters: In the tests below, Salmon v0.8.0 was run in alignment mode with the `--useErrorModel` flag. *Salmon-RF* consists of Salmon run with `--useRangeClusterEqClasses 4`. *Salmon-U* consists of Salmon run with `--noRichEqClasses`. *RSEM* v1.3.0 was run with default parameters. *eXpress* v1.5.1 was run with `--no-bias-correct` and other parameters were left as default (the extra parameter `--additional-batch 50` was used to produce the *eXpress (+50)* results). All alignments were generated using *Bowtie 2* version 2.2.9 with the default parameters chosen by *RSEM*. We note that these default parameters disallow indels in the resulting alignments (though Salmon and *eXpress* allow indels in the alignments the process, *RSEM* does not). Further, we note that since we examine simulated data without bias and since we compare against *RSEM* (which does not model sequence-

specific or fragment-GC bias) in the experimental data, we run all other methods without bias correction. On experimental RNA-seq data, one might expect bias correction alone to substantially improve the accuracy of a given method. Though those accuracy improvements should be orthogonal to those obtained by improving the fidelity of the likelihood function. All the tests are performed on a 64-bit Linux server with 256GB of RAM and 4 x 6-core Intel Xeon E5-4607 v2 CPUs running at 2.60GHz.

3.4.1 Small-scale simulations on RAD51 and its paralogs

We first consider a few small-scale simulations to motivate how the conditional probabilities considered by the *full model* (and approximated closely by the *range-based* equivalence classes) might improve abundance estimates. We note that these simulations are specifically constructed to represent adversarial and difficult-to-quantify mixtures of highly related isoforms. We consider the transcripts from large families of paralogous genes, under many random distributions of abundances. Often, the fragments will align to many different transcripts with few-or-no nucleotide differences, and sometimes even with similar implied insert sizes. Thus, we expect that closely approximating the conditional fragment probabilities might have a large effect in this case. We note, however, that such adversarial abundance configurations are likely rare in experimental data.

We consider two different, small-scale tests focusing around the gene RAD51 and members of its paralogous family in *Homo Sapiens*. The RAD51 family includes eight paralogous genes including RAD51 itself. RAD51 codes for a 339-amino acid protein that has a significant role in repairing double strand breaks of DNA [38].

In the first experiment we apply *RSEM* and all varieties of Salmon on all isoforms of the `RAD51` gene. We extracted all (10) reference transcripts of `RAD51` from the Ensembl (release 80) reference transcriptome. True reads counts for all transcripts were generated by sampling a read count for each transcript uniformly over $[1, 200]$; these counts represent base-depth coverage (left) in Figure 3.4. These counts were multiplied by 10 to derive the input read counts at $10X$ coverage (Figure 3.4, center) and by 100 to derive the counts at $100X$ coverage (Figure 3.4, right).

Given these read counts, the Polyester simulator [39] was then used to simulate 5 different read sets (replicates) from the same input distribution. This entire procedure was repeated 30 times, setting `R`'s random seed from 1 to 30 in sequence.

Since the reads are simulated, we can assess the deviation of the estimated abundances from the exact abundances for each transcript. We use the absolute relative difference (ARD) of estimated versus true read counts (Equation (3.7)) as the metric to evaluate the accuracy of different methods for each transcript over replicates, and Figure 3.4 shows a box plot of the distribution of ARD values over the 30 simulations.

As we expect, *Salmon-U* generally yields the largest ARDs, failing to utilize the information contained in the conditional fragment probabilities. Salmon generally performs better, suggesting that, even in this complex scenario, the aggregate weight maintained in the rich equivalence classes helps to recover some (but not all) of the fidelity of the full model. However, *Salmon-RF*, while only slightly increasing the number of equivalence classes considered, produces ARDs very close to those of *RSEM*, *eXpress (+50)* and *Salmon-FM*. This suggests that, even in this adversarial scenario, the *range-based* equivalence classes allow us to recover the inferential accuracy of the *full model*.

To further explore difficult abundance estimation scenarios, we consider the case of the presence of high abundance isoforms from more than one gene in the reference. Therefore, in the second set of experiments we consider 4 paralogs of *RAD51* (`RAD51`, `RAD51B`, `RAD51C` and `RAD51D`). We extract all transcripts corresponding to these genes and we run the same simulation as above with respect to all of these transcripts. Evaluation of ARDs for every transcript in all genes is displayed in Figure 3.5. The results in this case are similar to what was observed in the single gene experiment. In some cases, like transcript `ENST00000553595` from `RAD51B` (which is displayed as t_{10} in Figure 3.5), both *Salmon-U* and Salmon fail to estimate an accurate abundance. In other cases Salmon performs better than *Salmon-U*, e.g., transcript `ENST00000585947` from `RAD51D` (displayed as t_{50} in Figure 3.5). For almost every transcript, *Salmon-RF*, *Salmon-FM*, *eXpress (+50)* (*eXpress* under default settings performs a bit worse) and *RSEM* all perform similarly and better than the methods that adopt a purely *compatibility-based* factorization of the likelihood. As this simulation contains a large number of transcripts, we plot, in Figure 3.5, the box plots for only every 5^{th} transcript to make the plot more interpretable. The complete plot containing the ARD values for all transcripts of this paralogous family is provided in the supplementary materials of the published version of this work [82].

3.4.2 Transcriptome-wide analysis on synthetic data

To assess the performance of the proposed model on a large dataset of RNA seq reads, we generate synthetic data using *RSEM-sim*, and adopting the procedure used

by Bray et al. [14]. *RSEM* model parameters were generated by running *RSEM* on sample `NA12716_7` from the GEUVADIS [83] study. Using these model parameters, *RSEM-sim* was then used to generate a sample consisting of 30M 100bp paired-end RNA-seq reads.

Again, we explore the performance of *RSEM*, *eXpress* (both in default mode and with 50 rounds of batch EM) and 4 different variants of Salmon (*Salmon-U*, Salmon, *Salmon-RF* and *Salmon-FM*). We compute the Spearman correlation and MARD metrics of each of these methods compared with the true (i.e., simulated) abundances which are shown in the first two columns of Table 3.1 (MARD-all and Spearman-all). As we observe here, discarding all weight information in equivalence classes (*Salmon-U*) causes a drop in performance compared to the case with a single scalar per equivalence class-transcript pair (Salmon). Using the range-factorization proposed in this paper improves both the correlation and MARD measures even further, and bring its accuracy on par with that of *RSEM* and *Salmon-FM*, which adopt no factorization and run an EM algorithm that scales in the number of alignments in each iteration. In the default mode (i.e., using a single online pass), *eXpress* produces a larger MARD and lower correlation than any of the tools that run the batch EM until convergence. With 50 extra batch EM rounds (*eXpress (+50)*), *eXpress* performs more similarly to the other tools. We note that, in this data, the number of equivalence classes produced by the *range-based* factorization is $\sim 586,000$, only $\sim 150,000$ greater than the $\sim 438,000$ *compatibility-based* equivalence classes. Both of these numbers are orders-of-magnitude smaller than the $\sim 100,000,000$ distinct alignments for this dataset. The number of equivalence classes for all methods is shown in Table 3.3. This table also reports the number of "hits". The number of hits is the sum, over each equivalence class, of the number of transcripts in this equivalence class—

i.e., $\sum_{[q]\in\mathcal{C}} |\Omega([q])|$. This is the total number of items processed during each round of the EM algorithm. This small number of equivalence classes and hits allows the *Salmon-RF* model to run as fast as Salmon, which runs considerably faster than *Salmon-FM*, which, in turn, runs considerably faster than *RSEM*. With the exception of *eXpress*, which implements a constant-memory algorithm by design, the memory usage profiles for these different tools track the timing results (as expected). For more details, refer to figs. 3.6 and 3.7.

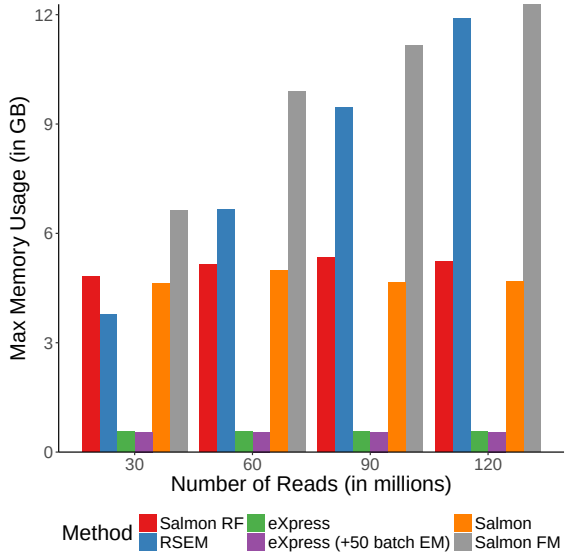


Figure 3.6: Maximum memory used by each method while running on input read files of sizes from 30M paired-end read counts to 120M. Memory usage of *RSEM* and *Salmon-FM* keeps increasing linearly by the read input counts since those tools provide full fidelity by storing conditional probabilities of each fragment-transcript pair. Salmon and *Salmon-RF* keep memory usage constant by using the notion of equivalence classes and clustering reads together. *eXpress* and *eXpress (+50)* also have constant memory usage, which is the lowest among all methods. However, this low memory usage is obtained at the expense re-loading all the alignments from the input `BAM` file in each iteration of the EM algorithm. This induces a considerable runtime burden, per EM-round, compared to other tools as displayed in fig. 3.7.

Though we observe an improvement for *Salmon-RF* and *Salmon-FM* over Salmon and especially *Salmon-U* in this case, we note that it is relatively small in scale. This is

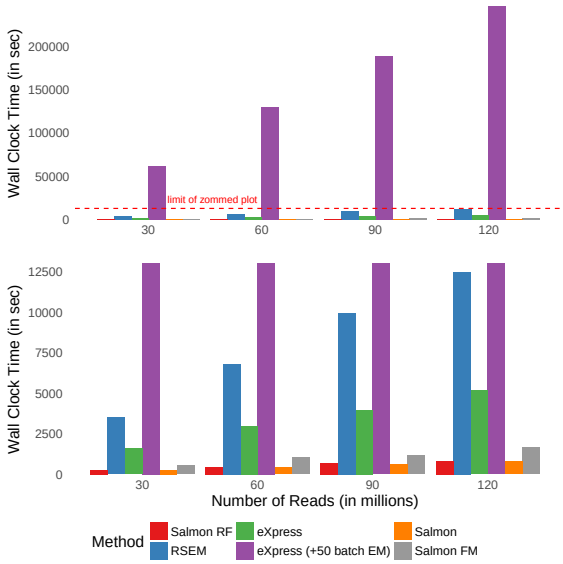


Figure 3.7: Total wall-clock time of each method as the number of paired-end reads is scaled from 30M to 120M. All methods receive the alignment BAM file as pre-computed input (i.e., alignment time is not considered in these evaluations). *Salmon-RF* is as fast as Salmon, and both are twice as fast as *Salmon-FM* which is the result of the reduced complexity of their batch EM rounds. *RSEM*, which also uses the full model in each iteration, takes longer than the Salmon variants. **Note**, unlike other methods, *eXpress* does not scale beyond 2 threads (when bias correction is disabled). Thus, we have placed an asterisk next to *eXpress* in the legend here to designate that the results should not be taken as a direct comparison to the other methods (which are using 16 threads). *eXpress (+50)* is considerably slower than other methods since *eXpress* is a constant-memory algorithm that iterates through the entire alignment BAM file (reading the alignments from disk) once per EM round.

because, while the aggressive *compatibility-based* factorizations do give up information, common expression patterns may not be complex or difficult enough to be greatly affected by the lossy factorization of the likelihood. Also, however, these aggregate metrics are computed over the entire transcriptome, and so, difficulties of these factorizations in deconvolving particularly complex scenarios may become lost in the noise of the vast number of good predictions.

To focus on the more difficult cases, we computed our accuracy metrics on a subset of the simulated data. Specifically, retaining the original abundance estimates over the

	MARD-all	Spearman-all	MARD-subset	Spearman-subset
Salmon-U	0.239	0.800	0.460	0.561
Salmon	0.223	0.813	0.432	0.575
Salmon-RF	0.214	0.825	0.412	0.644
Salmon-FM	0.214	0.825	0.411	0.647
eXpress	0.289	0.778	0.526	0.543
eXpress (+50)	0.227	0.827	0.477	0.593
RSEM	0.212	0.820	0.408	0.654

Table 3.1: Spearman correlation and MARD of quantification results compared to true abundances for synthetic data on all transcripts (MARD-all and Spearman-all), and the subset of transcripts (MARD-subset and Spearman-subset) where *RSEM*'s ARD is in $[0.25, 0.75]$.

	MARD-all	Spearman-all	MARD-subset	Spearman-subset
Salmon-U	0.239	0.800	0.460	0.561
Salmon	0.223	0.813	0.432	0.575
Salmon-RF	**0.214**	**0.825**	**0.412**	**0.644**
Salmon-FM	**0.214**	**0.825**	**0.411**	**0.647**
eXpress	0.289	0.778	0.526	0.543
eXpress (+50)	0.227	**0.827**	0.477	0.593
RSEM	**0.212**	**0.820**	**0.408**	**0.653**

Table 3.2: Spearman correlation and MARD of quantification results compared to true abundances for synthetic data on all transcripts (MARD-all and Spearman-all), and the subset of transcripts (MARD-subset and Spearman-subset) where *RSEM*'s ARD is in $[0.25, 0.75]$.

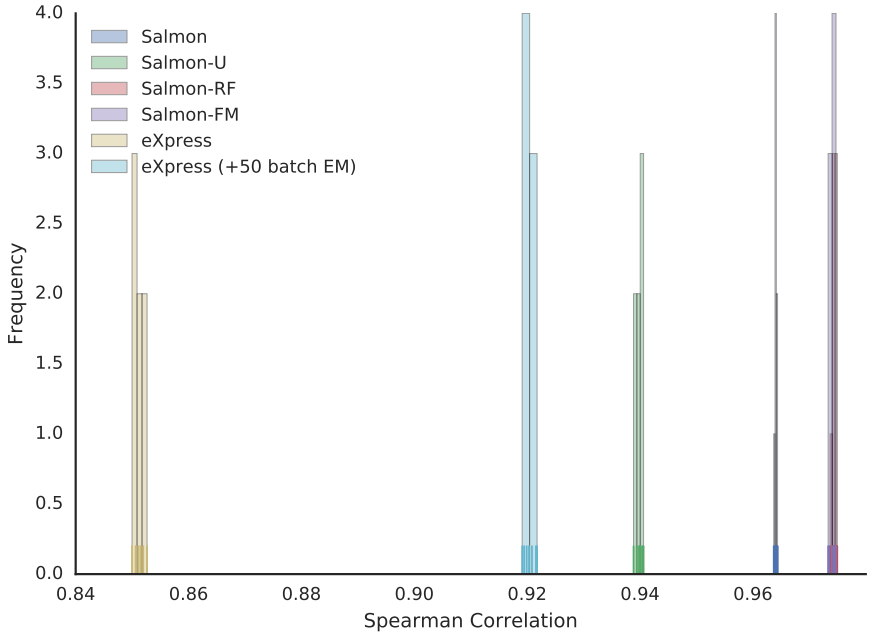


Figure 3.8: The Spearman correlation of transcripts abundance estimations with *RSEM* results reveals that *Salmon-FM* is highly correlated with *RSEM*. Very similar correlation with *RSEM* is observed by the proposed data-driven factorization, *Salmon-RF*. Salmon displays a lower correlation than *Salmon-RF*, but a higher correlation than *Salmon-U*. The variants of *eXpress* show a lower correlation than *Salmon-U*, with the offline EM iterations increasing *eXpress*' correlation considerably.

entire transcriptome, we restricted our analysis to those transcripts for which *RSEM* obtained an ARD between 0.25 and 0.75. The motivation for choosing these values is to discard the particularly "easy" to quantify transcripts (where the *full model* is likely neither necessary nor particularly helpful) as well as the "hopeless" transcripts (those where the inference exhibits significant error even under the reference implementation of the *full model*). The results of this analysis are shown in second two columns of Table 3.1 (MARD-subset and Spearman-subset). While the trend is similar to that observed on the full data, the difference between methods (and the impressive performance of *Salmon-RF*) becomes more clear. Specifically, we observe that Salmon outperforms *Salmon-U*, but this time the gap between Salmon and *Salmon-RF*, *Salmon-FM* and *RSEM* is larger.

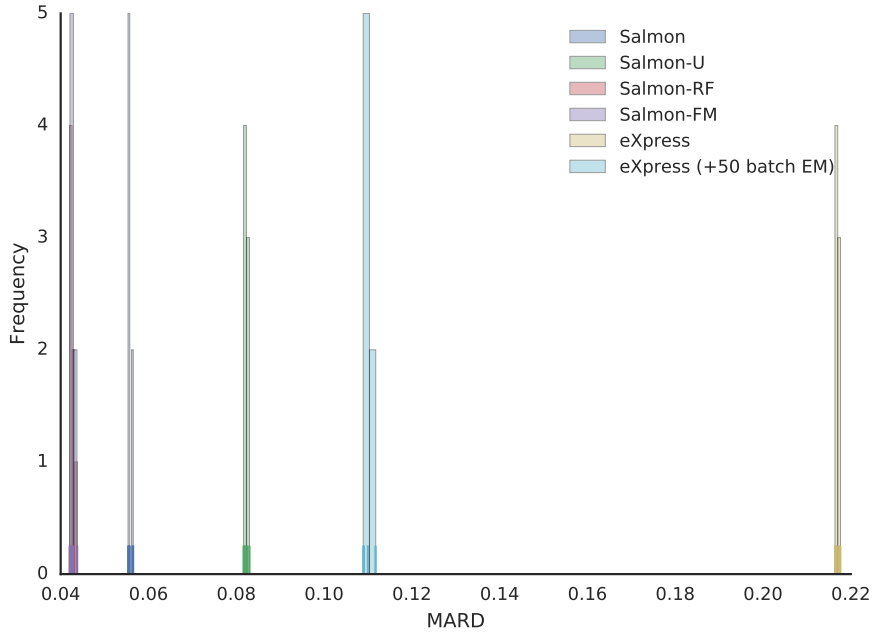


Figure 3.9: Comparing the MARD of estimated transcript fragment counts with respect to *RSEM* results shows similar trend to that observed with the Spearman correlations;i.e., *Salmon-FM* has the least error rate using RSEM abundances as the truth while *Salmon-RF* perform equally well. Salmon exhibits a lower MARD than *Salmon-U*, which is followed by both variants of *eXpress*.

This is most likely because this particular subset of transcripts presents a more difficult inference challenge, where the conditional probabilities provide useful evidence. In the case of these transcripts, running the EM algorithm until convergence seems particularly important, as we observe that *eXpress* (and even *eXpress* $(+50)$) trail the other methods, especially in terms of the MARD. This makes it evident that further refinement of the abundance estimates (i.e. more rounds of the EM) over a representation of the data encoding conditional fragment probabilities (as done in *RSEM*, *Salmon-FM* and *Salmon-RF*) is necessary to obtain improved accuracy on these transcripts.

We further investigate the performance of tools in non-alignment mode as well. Spearman correlation and MARD of transcript quantification with different tools on RSEM

	# eq. classes	# hits
Salmon-U	438393	5986371
Salmon	438393	5986371
Salmon-RF	625638	8212669
Salmon-FM	29447710	103663423

Table 3.3: The number of equivalence classes and hits, in the simulated data, under different likelihood factorizations.

	kallisto	Salmon	*Salmon-RF*	*Salmon-FM*
MARD	0.231	0.227	0.227	0.227
Spearman Correlation	0.805	0.810	0.811	0.811

Table 3.4: The Spearman correlation and MARD of estimations by different tools in "mapping" mode on synthetic data simulated by RSEM-sim as described in the main paper. We observe that, when using mappings instead of full alignments, the factorization being used has a smaller effect (an observation worthy of further consideration in future work). Still, we observe that Salmon performs slightly better than *kallisto* (which has a factorization akin to *Salmon-U* from the main paper), while *Salmon-RF* and *Salmon-FM* perform best.

simulated data is presented in Tables 3.4 and 3.5.

3.4.3 Transcriptome-wide analysis on experimental data

Finally, we explore the effect of our data-driven factorization method with the different versions of Salmon using experimental data from the SEQC(MSEQ-III) consortium [40] (NCBI GEO accession `SRR1215996` - `SRR1217002`). Specifically, the library is prepared

	kallisto	Salmon	*Salmon-RF*	*Salmon-FM*
MARD	0.441	0.433	0.430	0.429
Spearman Correlation	0.591	0.603	0.614	0.614

Table 3.5: The Spearman correlation and MARD of the estimates of different methods in mapping mode on synthetic data simulated by RSEM-sim. These metrics are computed only over the "difficult" transcripts—the set of transcripts for which *RSEM* achieves ARD values between 0.25 and 0.75.

on Universal Human Reference RNA (UHRR) from Stratagene and ERCC Spike-In controls and consists of $\sim$11M 100bp, paired-end reads sequenced on an Illumina HiSeq 2000 platform. The experiment consists of seven replicates with the same flowcell and barcodes but on different lanes.

As defined previously in section 3.4.1 we compare the performance of Salmon, *Salmon-FM*, *Salmon-RF*, *Salmon-U*, *eXpress*, *eXpress (+50)* with *RSEM*. However, unlike in previous sections, here, we lack a ground truth. Thus, we measure the accuracy of each method on the estimated number of reads, treating *RSEM*'s estimations of the number of reads for each transcript (which is observed to be among the most accurate on synthetic data in previous sections) as the truth. We perform a comparison across all seven replicates and consider the Spearman correlation and MARD metrics. Since these are technical replicates, we expect the performance over each replicate to be very similar, though we plot the results as a distribution in Figure 3.8 and Figure 3.9. The results on experimental data follow the same trend as we observed on synthetic data. That is, *Salmon-FM* correlates well with *RSEM* (as expected) because of the availability of full fragment level transcript probabilities. Likewise, we again observe that our proposed data-driven factorization method, *Salmon-RF*, performs essentially the same as the *full model*. Both of these methods agree more closely with *RSEM* than does Salmon, and again, *Salmon-U*, ignoring all fragment-level conditional probabilities, is further from *RSEM*'s results. The number of equivalence classes for each factorization are shown in Table 3.6. We also observe that *eXpress*, in its default mode, performs most differently from *RSEM* of the methods we considered. As expected, running additional rounds of the batch EM (*eXpress (+50)*) increases the similarity of *eXpress*' estimations with those of

	Salmon-U	Salmon	*Salmon-RF*	*Salmon-FM*
# eq. classes	427,611.000	427,611.000	624,340.000	9,077,708.000
# hits	5,737,414.000	5,737,414.000	8,318,638.000	50,325,595.000

Table 3.6: The number of equivalence classes and hits, in the experimental data, under different likelihood factorizations.

RSEM; though it is still less similar than the other methods.

3.5 Conclusion

While compatibility-based equivalence class factorizations [11, 12, 14, 17, 18] have paved the way in terms of substantially improving the efficiency of the iterative optimization procedures used for transcript-level quantification from RNA-seq data, they nonetheless make sacrifices in modeling fidelity to achieve this. While these methods generally perform adequately in terms of transcriptome-wide assessments, there are still important situations in which their compatibility-centric factorization of the underlying likelihood function discards information that can be important for accurate abundance estimates. Salmon [13] uses a dual-phase inference algorithm that allows it to recover some of the information discarded by other approaches. It improves upon the approximate factorization of the full likelihood function by incorporating a notion of *rich* equivalence classes of fragments. In some, but not all cases, this improved factorization is sufficient to recover the lost accuracy of the full model.

In this paper, we have introduced a data-driven factorization of the likelihood function that makes use of Salmon's dual-phase inference algorithm (*Salmon-RF*). We have shown that this improved factorization is able to match the accuracy of the *full model* while still

maintaining a reduced representation that is orders of magnitude smaller than the total number of fragment alignments.

We believe that this data-driven factorization represents the right tradeoff between efficiency and accuracy. Specifically, it demonstrates an almost indistinguishable sacrifice in efficiency beyond the factorization already employed by Salmon (which, itself, is similar in size to those employed by *mmseq*, *Sailfish* and *kallisto*), while producing no perceptible loss in accuracy compared to the full per-fragment likelihood function used by *RSEM* and similar methods.

In this paper, we have focused on the effect that the adopted factorization of the likelihood function can have on the ability of a method to accurately estimate transcript abundance. However, we note that there still remain small but interesting differences between methods that employ alignment and those that rely on mapping (i.e., quasi-mapping or pseudoalignment). Fully exploring the nature of these differences, and how they interact with the factorizations proposed herein, is an interesting direction for future work.

Finally, while we have investigated the effect different factorizations have on maximum likelihood estimates, fully-exploring the effect they have in estimating the variance of these estimates (e.g., via bootstrapping) or even in estimating the full posterior distribution of abundances (e.g., via Gibbs sampling) is another interesting direction for future work.

Chapter 4: Estimating the posterior with an improved version of bootstrap sampling based on Equivalence Classes

4.1 Overview

Rubin [84] introduced the Bayesian Bootstrap procedure, which generalizes the bootstrap and introduces a procedure for placing a prior over the sampling weights used in bootstrap resampling. The classic bootstrap is the posterior mean of the Bayesian bootstrap, and Rubin demonstrated they have quite similar estimation properties, generality, and similar limitations.

In particular, in the discussion of the paper introducing the Bayesian Bootstrap, when discussing model specification, Rubin muses:

"is it reasonable to use a model specification that effectively assumes all possible distinct values of X have been observed?"

Specifically, both the non-parametric bootstrap and the Bayesian Bootstrap make this assumption — no unobserved value will ever be included in a bootstrap replicate. This renders unobserved values "impossible" under the model, and prevents understanding the effect they might have on the inference procedure or the estimator being computed.

Here, we introduce the notion of the Augmented Bootstrap. This procedure follows

the general framework of the bootstrap (or, optionally, the Bayesian Bootstrap), but augments the observed data with additional "pseudo-observations" that represent values that are possible given a conceptual model for data generation, but which were not observed in the sample.

4.2 Methods

We will explain this idea in the underlying context of the problem of quantifying the abundance of transcripts from RNA-sequencing (RNA-seq) data. In this problem, we observe a collection of sequenced fragments, from which we can then estimate the abundance of the distinct transcripts using an expectation-maximization procedure (our "estimator" in the case of the bootstrap). Consider that we observe a collection $\mathcal{F}$ of N sequenced fragments. Each fragment aligns to a set of distinct locations which we denote as $a(f_i) = \{(j_1, k_1), (j_2, k_2), \dots\}$ signifying that fragment f_i aligns to transcript j_1 at position k_1, to transcript j_2 at position k_2, and soforth. For a given set η of transcript abundances, we can write the probability of observing the set $\mathcal{F}$ of fragments as:

$$\mathcal{L} = \prod_{i=1}^{N} \sum_{(j,k) \in a(f_i)} P(t_j|\eta) P(f_i|t_{j,k} = 1)$$

where $P(t_j|\eta)$ is the probability of selecting transcript j for sequencing conditional on the transcript abundances and $P(f_i|t_{j,k} = 1)$ is the probability of generating fragment i from transcript j at position k.

While there is no closed-form solution to determine the η that maximize $\mathcal{L}$, we can, at least locally, optimize this quantity using an EM algorithm. We are then interested in

assessing the accuracy of $\hat{\eta}$, our maximum likelihood estimator for η.

To assess this accuracy, we can use the non-parametric bootstrap. In this framework, we will resample (with replacement) from $\mathcal{F}$ to produce a series of bootstrap replicates $\{\mathcal{F}_\infty, \mathcal{F}_\in, \ldots \mathcal{F}_\lfloor\}$, and for each we can use the EM procedure to obtain a maximum likelihood estimate of the transcript abundances given this bootstrap replicate; we denote these esitimates as $\{\hat{\eta_1}, \hat{\eta_2}, \ldots, \hat{\eta_b}\}$. We can then assess e.g. the variance of the estimate for the abundance of transcript j, denoted as $\hat{\eta}^{(j)}$, by assessing the sample variance of $\{\hat{\eta_1}^{(j)}, \hat{\eta_2}^{(j)}, \ldots, \hat{\eta_b}^{(j)}\}$.

Because we are resampling *sequenced fragments* with replacement, we will never observe in our resampling a pattern of alignments for a fragment different from what we saw in our original sample $\mathcal{F}$. This leads to some interesting, and perhaps undesirable behavior of the bootstrap.

Consider a pair of alleles of a transcript that differ only at a single locus. Further, imagine that these transcripts are sequence distinct from the rest of the transcriptome (i.e. they share no multimapping reads with other transcripts apart from their sibling allele). Let there be N_t reads that map to both of these transcripts, and 0 reads that map uniquely to either (i.e. 0 reads that overlap the variant locus).

From the perspective of our estimator, these transcripts are inferentially indistinguishable. Specifically, with no prior information on whether one of these alleles is *a priori* more likely than the other, we have no information about how the N_t fragments should be allocated among these transcripts. Perhaps t_p and t_m each give rise to $\frac{N_t}{2}$ fragments (what the EM will likely tell us), or perhaps t_p gives rise to 0 fragments and t_m gives rise to all N_t fragments. In fact, any combination between the two alleles that sums to N_t is feasible

and has equal likelihood.

The crux of the issue in uncovering this uncertainty using the non-parametric bootstrap (either the traditional or the Bayesian variant), is that no observation was made that distinguishes between these alleles. Thus, no matter how we resample the original observations, we will never be able to recover the underlying uncertainty in the abundance of these transcripts. Our estimator will demonstrate some variance over the bootstrap replicates, of course, but only related to what fraction of the original N_t reads we sample within each replicate (with the expected value, of course, being N_t).

4.2.1 The Augmented Bootstrap

To address this issue, we propose the augmented bootstrap. This procedure is applicable in sutations where the data over which inference is being performed have a finite (and ideally "small") support **Mike: I think of the parameter having support, not data** We describe the procedure in full generality, and then explain the heuristics that we adopt to make the procedure computationally expedient in our use case.

The main idea behind the augmented bootstrap is that we will augment our observed data with some set of "pseudo-observations" — data values that *might* have been observed, but which were not observed in our sample. This is analogous to placing a prior over the discrete set of possible observations that may be made. The prior may be informative or noninformative, and this can be represented by means of the sampling weights assigned to each of these pseudo-observations.

Let us consider our chosen problem of transcript abundance from RNA sequencing

data. To simplify the exposition, let us further assume we are dealing with single-end reads, and that we will ignore the possibility of sequencing error when generating our "pseudo-observations". Then, in this case we may consider producing a set of pseudo-observations by drawing, from every transcript, a fragment starting at every position. Let this set of pseudo-observations be denoted as $\mathcal{P}$ and let us denote by $\mathcal{F}_{\mathcal{A}}$ the set $\mathcal{F} \cup \mathcal{P}$. This is our collection of augmented observations — the set of samples over which we will perform our augmented bootstrapping procedure. While we could consider every observation $f \in \mathcal{F}_{\mathcal{A}}$ to be sampled with replacement with equal probability, this introduces an obvious dependency between $|\mathcal{F}|$ and $|\mathcal{P}|$ where the effect of the pseudo-observations will be relatively larger when the original sample is small and less important when the original saple is very large. Thus, we will consider modifying the sampling weights between "true" observations (those in $\mathcal{F}$) and pseudo-observations (those in $\mathcal{P}$). While we can consider giving each pseudo-observation a distinct sampling weight, let us consider here the simpler case where we define $w < 1$ to be the sampling weight applied to every pseudo-observation. In fact, this gives us a direct way to interpret the weight of all pseudo-observations as a proportion of the weight of the true observations. Consider that we want the pseudo-observations to account for 1% of the samples in a given bootstrap sample — then we can set the weights in the following way.

Let $|\mathcal{F}| = N$ and $|\mathcal{P}| = N'$ so taht $|\mathcal{F}_{\mathcal{A}}| = N + N'$. We would like the weight of each pseudo-observation to be proportional to w times the weight of each true datum, where $0 \leq w \leq 1$.

Now, when we perform a bootstrap replicate, we wish to resample with replacement from $\mathcal{F}_{\mathcal{A}} = \langle f_1, f_2, \ldots, f_N, p_1, p_2, \ldots, p_{N'} \rangle$ where we will use the sampling weights

$\langle y_1, y_2, \ldots, y_N, z_1, z_2, \ldots, z_{N'} \rangle$ where

$$y_i = \frac{(1-w)(N+N')}{N}$$

$$z_i = \frac{w(N+N')}{N'}$$

This gives the relative sampling weight of every pseudo-observation and every true observation equal to w and $1-w$ respectively. Thus, in expectation, $(100 \times w)\%$ of our sampled data in each bootstrap replicate will consist of pseudo-observations, while the rest will consist of true observations.

The effect of adding these pseudo-observations to augment our bootstrap sampling is that we can now observe outcomes in our estimates that previously would have not been possible due to plausible observations that were missing from our specific sample.

Returning to the running example of the alleles of a single transcript; in addition to the N_t fragments compatible with both alleles, there will now be pseudo-observations compatible only with t_m and pseudo-observations compatible only with t_p. In a given bootstrap replicate the inclusion or exclusion of these pseudo-observations will result in a different relative estimate between the abundances of these two alleles than we will ever arrive at under a bootstrap replicate of the original data. In a sense, this augmentation is enabling us to approach the bootstrap procedure from a more Bayesian perspective, where data are *possible* even when they are not observed. The cost for this, of course, is that we must make some decision about their prior probability.

4.2.2 Heuristics for augmenting the bootstrap

We have defined the augmented bootstrap procedure as augmenting the observed sample with pseudo-observations for *all possible* observations. This can immediately pose some challenges. First, it requires that the set of possible observations is finite and sufficiently small to be enumerated. Second, many possible pseudo-observations, though technically possible given the imposed prior, may have little to no effect on the resulting inference of interest.

Again, consider our running example of transcript abundance estimation. Here, under observations of the original sample, many transcripts will both have an estimated abundance of 0 and, further, will simply have no observed sequenced fragments compatible with them. Generating and possibly sampling pseudo-observations from these transcripts may lead to small fluctuations in the estimated abundance of these transcripts across bootstrap replicates, but it is unlikely to have any substantial effect on the "main" inference problem (i.e. the estimated maximum likelihood abundances of non-trivially expressed transcripts) — and since no "true" observations are compatible with these transcripts, we'd expect their posterior samples to be rather uninteresting.

This immediately suggests one potential heuristic for limiting the number of pseudo-observations with which we will augment our true samples. Let $\mathcal{T}$ be our complete set of transcripts and let $\mathcal{T}_{\mathcal{F}}$ be the set of transcripts having at least one fragment in $\mathcal{F}$ that aligns to them. Rather than generating pseudo-observations from all of $\mathcal{T}$, we may consider generating pseudo-observations only from $\mathcal{T}_{\mathcal{F}}$ — that is from transcripts that we predict to be expressed and from those that have compatible observations (but which

we may not predict to be expressed in the maximum likelihood estimate). In general, this will produce far fewer pseudo-observations than if we generate them from all of $\mathcal{T}$. Furthermore, the sampling of *these* pseudo-observations are much more likely to lead to alternative high-likelihood estimates across bootstrap-replicates, because they are most likely to change the balance of observations in highly-ambiguous components within the inference problem. Thus, we are, in effect, selecting a smaller set of pseudo-observations that are more likely to uncover the relevant uncertainty in our estimator.

4.3 Results

We have created a simulated dataset of Drosophila with an expression of the reads from two alleles of each gene. To create the allelic reads, we flip a random base in the original genes and then generate reads with Polyester RNA-seq simulator[39] from both allelic genes. Based on the procedure for generating the allelic reads, most, if not all, the reads mapping to a transcript, map to both its paternal and maternal alleles. We demonstrate how the augmented Bootstrap improves the accuracy of abundance estimation on a sample with high levels of ambiguity.

4.3.1 Estimating the allelic expression of a simulated sample

To investigate how the posterior sampling could improve the accuracy of the abundance estimation, we follow a specific procedure for creating the multi-allelic RNA-seq sample from the Drosophila genome. We assume the original genome is one parental allele, and generate the second allele by adding a random mutation to 5 exons of each gene.

To increase the uncertainty in the abundance estimations, we simulate reads from both alleles of every transcript in the reference transcriptome. Each transcript in the paternal allele is expressed by two reads. Then we select 1000 transcripts from each gene with 3 to 6 isoforms and alter the expression of the maternal allele of the transcript randomly in a way that the final gene count of the maternal allele will have the same expression level as the paternal allele.

We should acknowledge that the way this data is simulated is unrealistic because each transcript has multiple alleles in the sample and all the transcripts are expressed. The reason, we follow the procedure is to include a very high level of uncertainty so that estimating the posterior distribution becomes more challenging.

First of all, we demonstrate how the regular Bootstrap sampling fail to capture the entire posterior distributions for many $\mathcal{T}$ in this sample. We create inferential replicates by Bootstrap sampling available in Salmon and *kallisto*. We also create Bootstrap samples using the augmented Bootstrap sampling with different weights. We Salmon in two different optimization modes of 'EM' and 'VBEM'. The 'EM' mode is the regularly expected maximization algorithm and the 'VBEM' is a variational bayesian version of the EM with non-zero priors for the model's parameters which induces further sparsity on the final solution.

Table 4.1 shows the performance of allelic expression quantification with both point estimates and the mean of the posterior distributions. We observe that using the augmented Bootstrap sampling with both 0.01 and 0.1 weights increase the accuracy of the abundance estimation. This suggests that augmenting the sample with one unique read for each possible transcript increases the ability of the uncertainty estimation with

Method	Spearman-all
kallisto	0.939
kallisto Bootstrap	0.942
Salmon EM point-estimates	0.954
Salmon VBEM point-estimates	0.797
Salmon VBEM regular Bootstrap	0.901
Salmon VBEM augmented Bootstrap (w=0.01)	0.985
Salmon VBEM augmented Bootstrap (w=0.1)	0.992
Salmon EM regular Bootstrap	0.956
Salmon EM augmented Bootstrap (w=0.01)	0.986
Salmon EM augmented Bootstrap (w=0.1)	0.992

Table 4.1: Accuracy of abundance estimation based on the Spearman correlation. Using the mean of the posterior distribution generated by augmented Bootstrap sampling with weight 0.01 and 0.1 significantly improves the accuracy in the allelic expression quantification with RNA-seq.

Bootstrap sampling.

We also evaluated the performance of the two optimization methods 'EM' and 'VBEM'. Because of the sparsity imposed by 'VBEM' and the specific characteristics of the simulated sample that almost all the transcripts are expressed, the point estimates calculated by 'VBEM' become less correlated with the truth. However, we observe that using the mean of the posterior distribution, both 'EM' and 'VBEM' reach a very high correlation with the truth.

To further investigate the effect of the augmented Bootstrap sampling for recovering the posterior distribution, we evaluated the confidence intervals calculated based on both regular Bootstrap sampling and augmented Bootstrap sampling. In this analysis, we look at confidence intervals with different lengths (0 to 100%) and at each level we calculate how many times the truth is found in that interval. If we have the true posterior distribution, the truth should be found in each interval relative to the length of the interval, e.g., if we are looking at the 95% confidence interval, we should be able to find the true

Confidence interval length	5%	25%	50%	75%	95%
Salmon VBEM regular Bootstrap	0.064	0.296	0.540	0.746	0.888
Salmon VBEM augmented Bootstrap (w=0.01)	0.068	0.332	0.613	0.820	0.942
Salmon VBEM augmented Bootstrap (w=0.1)	0.028	0.145	0.299	0.479	0.684
Salmon EM regular Bootstrap	0.061	0.287	0.523	0.725	0.876
Salmon EM augmented Bootstrap (w=0.01)	0.068	0.326	0.608	0.818	0.941
Salmon EM augmented Bootstrap (w=0.1)	0.028	0.147	0.303	0.480	0.686

Table 4.2: Evaluating the confidence intervals achieved by each Bootstrap sampling method. Augmenting the bootstrap samples with '0.01' of unique reads increases the accuracy of each interval considerably.

value of the abundance at least 95% of the time.

Table 4.2 shows the percentage of the transcripts at each level for which the truth is recovered in the corresponding interval. We observe that either optimization method using a pseudo-observation with 1% weight improves the accuracy of the confidence intervals. The highest accuracy is achieved by the 95% confidence interval of the augmented Bootstrap sampling with $w = 1\%$. As we observed earlier, increasing the pseudo-observation weight to 10% increased the spearman correlation of the mean with the truth, however, in this evaluation we observe that sampling too many reads from the pseudo-observation do not lead to a better posterior distribution.

4.4 Discussion

We demonstrated that following a Bayesian approach to augment the Bootstrap sampling for generating inferential replicates increase the reliability of the posterior distribution. Our approach selects all the possible transcripts in a sample based on the existence of at least one read aligning to a transcript. We augment the set of observed reads with an extra unique read per possible transcript. Then, the pseudo-observations will be sampled in

each Bootstrap sample based on a weight that is typically much smaller than the weight that observed reads are sampled from. In the allelic expression experiment, we observed that having 1% of the reads coming from the added set impros the accuracy of the posterior distribution.

Chapter 5: Conclusion

Throughout this dissertation, we have explored improving the lightweight approaches employed in various steps of the RNA-seq analysis pipeline, i.e., mapping or alignment of the reads to a known reference, estimating the abundance of transcripts, and assessing the accuracy of the point estimates by evaluating the posterior distribution. The recurring theme of all the methods we have introduced here is improving the accuracy of lightweight methods by maintaining their efficiency.

In chapter 2, we introduced selective-alignment as a new algorithm for efficiently aligning the reads to the reference transcriptome. This approach increases both sensitivity and specificity of quasi-mapping. Selective-alignment increases the sensitivity by performing safe skips for querying each k-mer in the reference. It also relaxes other constraints imposed on merging the mappings discovered for a read. It also introduces the concept of co-mapping for further refining the candidate mappings. Selective-alignment also computes an alignment score for each mapping to filter spurious hits further. The alignment-score could also be used in the quantification step to improve the accuracy of the estimations. We show that selective-alignment improves the accuracy of the quantification with lightweight methods without sacrificing the performance.

Furthermore, we have introduced PuffAligner in chapter 2. PuffAligner is built on

top of the Pufferfish index, which is an efficient colored compacted de Bruijn graph base index of a collection of reference sequences. PuffAligner is a multi-purpose aligner that can be utilized for aligning DNA-seq, RNA-seq, and metagenomic reads. PuffAligner finds high-quality alignments for short reads, which are similar to those discovered by accurate tools, e.g., *Bowtie2*, in a significantly shorter amount of time.

We have investigated the effect of the factorization of the likelihood employed by lightweight RNA-seq quantification tools on the accuracy of the estimations. These tools treat all the fragments (reads) which are mapped to the same set of reference sequences as identical and represent all the fragments compatible with the same set of transcripts as one equivalence class in the likelihood function. This factorization approximates the likelihood function because of the differences in the characteristics of each fragment in an equivalence class, e.g., different fragment length and alignment compatibilities lead to different conditional probabilities. I have proposed an improved factorization in chapter 3 which groups the fragments in an equivalence class that is not only similar in terms of the set of transcripts by which they are compatible but also similar in terms of the conditional probabilities to those set of transcripts. We observed that this improved factorization leads to greater accuracy of the abundance estimation with almost no effect on the speed of the lightweight methods.

In chapter 4, we have tackled the problem of estimating the posterior distribution for RNA-seq abundance estimations. The posterior distribution is necessary to assess the accuracy of the quantification results. Bootstrap sampling is a popular approach for creating the inferential replicates from the original observed sample. However, the existing Bootstrap approaches only consider the observed set of reads (equivalence classes)

for generating the inferential replicates. We introduced the concept of augmented Bootstrap sampling, which augments the original sample with additional observations. We illustrated that augmenting the Bootstrap samples with reads uniquely mapping to each possible transcript, i.e., the set of transcripts with at least one aligned read from the observed sample, improves the uncertainty estimation.

5.1 Future Work

Most RNA-seq quantification tools rely on the generative model proposed by [15], which we have discussed in detail throughout this dissertation. A shortcoming of this model is that it assumes the set of reference sequences against which we are quantifying is complete and that all the fragments in the sample come from one of the transcripts in the reference. RSEM [15] tries to address this issue by including an extra transcript as the noise transcript to which all the fragments not aligned to any other transcripts would map. However, based on how the reference transcripts are created by alternative splicing, there might exist a fragment that is completely compatible with an existing transcript in the reference sequence, but it comes from a transcript missed in our reference transcriptome. This might happen because the exon that the fragment comes from is present in one of the existing transcripts. Such fragments possibly lead to coverage anomalies of the transcripts present in the reference, which is completely ignored by the generative model of RNA-seq quantification.

Recently, different post quantification analyses have been proposed to detect and possibly fix such coverage anomalies. For instance, in a recent paper [85], the authors

develop a method for detecting anomalies in the coverages of transcripts in the RNA-seq quantification results. As they state, such anomalies could be evidence of misquantification of transcripts. They try to resolve the anomalies by moving around reads between transcripts which they map to obtain results with less expression of anomalies while still maintaining a high likelihood. This study also suggests that the cases where the anomalies could not be resolved by simply transferring reads between the existing transcripts could indicate some mis-annotated transcripts in the reference transcriptome.

We believe this issue could be addressed more properly during the quantification step rather than post analysis of the abundance estimations. It is possible to detect coverage anomalies during both the online and offline phases of the quantification procedure. During the online phase, reads are assigned only once; therefore, scanning all the transcripts once at the end of the online phase could highlight the anomalies. However, the reads are redistributed during each iteration of the offline EM. Therefore, keeping track of the anomalies, in that case, is more challenging. Another challenge is that the mapping positions are not available in this phase in the current implementation of offline EM. One possible solution is keeping track of mapping positions by increasing the resolution of the equivalence classes. In addition to these challenges, some anomalies will never be resolved by redistributing the reads, probably because the reads are not sequenced from any of the sequences in the reference set. This suggests the existence of novel isoforms which are missing from the set of known reference sequences. Therefore, the next step in improving RNA-seq quantification model would be detecting such cases and reporting possible existing transcripts in the reference at the end of the quantification procedure.

When using Natbib, use the following three lines:

Bibliography

[1] Ali Mortazavi, Brian A Williams, Kenneth McCue, Lorian Schaeffer, and Barbara Wold. Mapping and quantifying mammalian transcriptomes by RNA-Seq. *Nature Methods*, 5(7):621, 2008.

[2] Avi Srivastava, Laraib Malik, Hirak Sarkar, Mohsen Zakeri, Fatemeh Almodaresi, Charlotte Soneson, Michael I Love, Carl Kingsford, and Rob Patro. Alignment and mapping methodology influence transcript abundance estimation. *BioRxiv*, page 657874, 2019.

[3] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

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

[5] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357, 2012.

[6] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem, 2013.

[7] Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.

[8] Bo Liu, Hongzhe Guo, Michael Brudno, and Yadong Wang. debga: read alignment with de bruijn graph-based seed and extension. *Bioinformatics*, 32(21):3224–3232, 2016.

[9] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

[10] Fatemeh Almodaresi, Hirak Sarkar, Avi Srivastava, and Rob Patro. A space and time-efficient index for the compacted colored de bruijn graph. *Bioinformatics*, 34 (13):i169–i177, 2018.

[11] Rob Patro, Stephen M Mount, and Carl Kingsford. Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms. *Nature Biotechnology*, 32(5):462–464, 2014.

[12] Avi Srivastava, Hirak Sarkar, Nitish Gupta, and Rob Patro. RapMap: a rapid, sensitive and accurate tool for mapping RNA-seq reads to transcriptomes. *Bioinformatics*, 32(12):i192–i200, 2016. doi: 10.1093/bioinformatics/btw277. URL `http://dx.doi.org/10.1093/bioinformatics/btw277`.

[13] Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417–419, mar 2017. doi: 10.1038/nmeth.4197.

[14] Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology*, 34(5):525–527, 2016.

[15] Bo Li, Victor Ruotti, Ron M Stewart, James A Thomson, and Colin N Dewey. RNA-Seq gene expression estimation with read mapping uncertainty. *Bioinformatics*, 26 (4):493–500, 2010.

[16] Adam Roberts and Lior Pachter. Streaming fragment assignment for real-time analysis of sequencing experiments. *Nature Methods*, 10(1):71–73, 2013.

[17] Ernest Turro, Shu-Yi Su, Ângela Gonçalves, Lachlan JM Coin, Sylvia Richardson, and Alex Lewin. Haplotype and isoform specific expression estimation using multi-mapping RNA-seq reads. *Genome Biology*, 12(2):1, 2011.

[18] Marius Nicolae, Serghei Mangul, Ion I Mandoiu, and Alex Zelikovsky. Estimation of alternative splicing isoform frequencies from RNA-Seq data. *Algorithms for Molecular Biology*, 6(1):9, 2011.

[19] Marius Nicolae, Serghei Mangul, Ion I Măndoiu, and Alex Zelikovsky. Estimation of alternative splicing isoform frequencies from RNA-Seq data. *Algorithms for Molecular Biology*, 6(1):1, 2011.

[20] The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL `https://doi.org/10.5281/zenodo.3509134`.

[21] Anqi Zhu, Avi Srivastava, Joseph G Ibrahim, Rob Patro, and Michael I Love. Nonparametric expression analysis using inferential replicate counts. *Nucleic Acids Research*, 47(18):e105–e105, 2019.

[22] Bradley Efron. Computers and the theory of statistics: thinking the unthinkable. *SIAM review*, 21(4):460–480, 1979.

[23] Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology*, 34(5):525, 2016.

[24] Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417, 2017.

[25] Hui Y Xiong, Leo J Lee, Hannes Bretschneider, Jiexin Gao, Nebojsa Jojic, and Brendan J Frey. Probabilistic estimation of short sequence expression using rna-seq data and the "positional bootstrap". *bioRxiv*, page 046474, 2016.

[26] Peter Glaus, Antti Honkela, and Magnus Rattray. Identifying differentially expressed transcripts from RNA-seq data with biological variation. *Bioinformatics*, 28(13):1721–1728, 2012.

[27] Hirak Sarkar, Mohsen Zakeri, Laraib Malik, and Rob Patro. Towards selective-alignment: Bridging the accuracy gap between alignment-based and alignment-free transcript quantification. In *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 27–36, Washington DC, USA, 2018. ACM. URL `http://doi.acm.org/10.1145/3233547.3233589`.

[28] Fatemeh Almodaresi, Mohsen Zakeri, and Rob Patro. Puffaligner: A fast, efficient, and accurate aligner based on the pufferfish index. *Bioinformatics*, 2021.

[29] Zhaojun Zhang and Wei Wang. RNA-Skim: a rapid method for RNA-Seq quantification at transcript level. *Bioinformatics*, 30(12):i283–i292, 2014.

[30] Chelsea J-T Ju, Ruirui Li, Zhengliang Wu, Jyun-Yu Jiang, Zhao Yang, and Wei Wang. Fleximer: Accurate quantification of rna-seq via variable-length k-mers. In *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 263–272, 2017.

[31] Michael J Axtell. Butter: High-precision genomic alignment of small RNA-seq data. *bioRxiv*, page 007427, 2014.

[32] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948, 1993.

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

[34] Martin Šošić and Mile Šikić. Edlib: a C/C++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, 33(9):1394–1395, 2017.

[35] Cyan. xxHash — Extremely fast hash algorithm. `http://cyan4973.github.io/xxHash/`, 2018. Accessed: 2018-04-30.

[36] Bo Li and Colin N Dewey. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics*, 12(1):1, 2011.

[37] Mohsen Zakeri, Avi Srivastava, Fatemeh Almodaresi, and Rob Patro. Improved data-driven likelihood factorizations for transcript abundance estimation. *Bioinformatics*, 2017. (Proceedings of ISMB 2017, in press).

[38] Andrew Yates, Wasiu Akanni, M Ridwan Amode, Daniel Barrell, Konstantinos Billis, Denise Carvalho-Silva, Carla Cummins, Peter Clapham, Stephen Fitzgerald, Laurent Gil, et al. Ensembl 2016. *Nucleic acids research*, page gkv1157, 2015.

[39] Alyssa C Frazee, Andrew E Jaffe, Ben Langmead, and Jeffrey T Leek. Polyester: simulating RNA-seq datasets with differential transcript expression. *Bioinformatics*, 31(17):2778–2784, 2015.

[40] Seqc/Maqc-Iii Consortium et al. A comprehensive assessment of rna-seq accuracy, reproducibility and information content by the sequencing quality control consortium. *Nature biotechnology*, 32(9):903–914, 2014.

[41] L. Schaeffer, H. Pimentel, N. Bray, P. Melsted, and L. Pachter. Pseudoalignment for metagenomic read assignment. *Bioinformatics*, Feb 2017. doi: https://doi.org/10.1093/bioinformatics/btx106.

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

[43] Daehwan Kim, Ben Langmead, and Steven L Salzberg. Hisat: a fast spliced aligner with low memory requirements. *Nature methods*, 12(4):357–360, 2015.

[44] Daehwan Kim, Joseph M Paggi, Chanhee Park, Christopher Bennett, and Steven L Salzberg. Graph-based genome alignment and genotyping with hisat2 and hisat-genotype. *Nature biotechnology*, 37(8):907–915, 2019.

[45] Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.

[46] Yang Liao, Gordon K Smyth, and Wei Shi. The subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic acids research*, 41(10):e108–e108, 2013.

[47] Matei David, Misko Dzamba, Dan Lister, Lucian Ilie, and Michael Brudno. Shrimp2: sensitive yet practical short read mapping. *Bioinformatics*, 27(7):1011–1012, 2011.

[48] Can Alkan, Jeffrey M Kidd, Tomas Marques-Bonet, Gozde Aksay, Francesca Antonacci, Fereydoun Hormozdiari, Jacob O Kitzman, Carl Baker, Maika Malig, Onur Mutlu, et al. Personalized copy number and segmental duplication maps using next-generation sequencing. *Nature genetics*, 41(10):1061, 2009.

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

[50] Chirag Jain, Alexander Dilthey, Sergey Koren, Srinivas Aluru, and Adam M. Phillippy. A fast approximate algorithm for mapping long reads to large reference databases. *Journal of Computational Biology*, 25(7):766–779, July 2018. doi: 10.1089/cmb.2018.0036. URL `https://doi.org/10.1089/cmb.2018.0036`.

[51] Antoine Limasset, Bastien Cazaux, Eric Rivals, and Pierre Peterlongo. Read mapping on de bruijn graphs. *BMC bioinformatics*, 17(1):237, 2016.

[52] Mahdi Heydari, Giles Miclotte, Yves Van de Peer, and Jan Fostier. Browniealigner: accurate alignment of illumina sequencing data to de bruijn graphs. *BMC bioinformatics*, 19(1):311, 2018.

[53] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature genetics*, 44(2):226, 2012.

[54] Martin D Muggli, Alexander Bowe, Noelle R Noyes, Paul S Morley, Keith E Belk, Robert Raymond, Travis Gagie, Simon J Puglisi, and Christina Boucher. Succinct colored de bruijn graphs. *Bioinformatics*, 33(20):3181–3187, 2017.

[55] Fatemeh Almodaresi, Prashant Pandey, and Rob Patro. Rainbowfish: A succinct colored de bruijn graph representation. In *17th International Workshop on Algorithms in Bioinformatics (WABI 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[56] Prashant Pandey, Fatemeh Almodaresi, Michael A Bender, Michael Ferdman, Rob Johnson, and Rob Patro. Mantis: A fast, small, and exact large-scale sequence-search index. *Cell systems*, 7(2):201–207, 2018.

[57] Hajime Suzuki and Masahiro Kasahara. Introducing difference recurrence relations for faster semi-global alignment of long sequences. *BMC Bioinformatics*, 19(1):45, 2018.

[58] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015.

[59] Shifu Chen, Yanqing Zhou, Yaru Chen, and Jia Gu. fastp: an ultra-fast all-in-one fastq preprocessor. *Bioinformatics*, 34(17):i884–i890, 2018.

[60] Adam Frankish, Mark Diekhans, Anne-Maud Ferreira, Rory Johnson, Irwin Jungreis, Jane Loveland, Jonathan M Mudge, Cristina Sisu, James Wright, Joel Armstrong, et al. Gencode reference annotation for the human and mouse genomes. *Nucleic acids research*, 47(D1):D766–D773, 2019.

[61] Jake R Conway, Alexander Lex, and Nils Gehlenborg. Upsetr: an r package for the visualization of intersecting sets and their properties. *Bioinformatics*, 33(18): 2938–2940, 2017.

[62] Manuel Holtgrewe. Mason: a read simulator for second generation sequencing data. 2010.

[63] Hy Vuong, Thao Truong, Thang Tran, and Son Pham. A revisit of rsem generative model and its em algorithm for quantifying transcript abundances. *bioRxiv*, page 503672, 2018.

[64] Alyssa C Frazee, Andrew E Jaffe, Ben Langmead, and Jeffrey T Leek. Polyester: simulating rna-seq datasets with differential transcript expression. *Bioinformatics*, 31(17):2778–2784, 2015.

[65] John Lonsdale, Jeffrey Thomas, Mike Salvatore, Rebecca Phillips, Edmund Lo, Saboor Shad, Richard Hasz, Gary Walters, Fernando Garcia, Nancy Young, et al. The genotype-tissue expression (gtex) project. *Nature genetics*, 45(6):580, 2013.

[66] Bo Li and Colin N Dewey. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics*, 12(1):323, 2011.

[67] Ben Langmead. Aligning short sequencing reads with bowtie. *Current protocols in bioinformatics*, 32(1):11–7, 2010.

[68] Fan Wu, Su Zhao, Bin Yu, Yan-Mei Chen, Wen Wang, Zhi-Gang Song, Yi Hu, Zhao-Wu Tao, Jun-Hua Tian, Yuan-Yuan Pei, et al. A new coronavirus associated with human respiratory disease in china. *Nature*, 579(7798):265–269, 2020.

[69] Pavel V Baranov, Clark M Henderson, Christine B Anderson, Raymond F Gesteland, John F Atkins, and Michael T Howard. Programmed ribosomal frameshifting in decoding the sars-cov genome. *Virology*, 332(2):498–510, 2005.

[70] Yixuan Wang, Yuyi Wang, Yan Chen, and Qingsong Qin. Unique epidemiological and clinical features of the emerging 2019 novel coronavirus pneumonia (covid-19) implicate special control measures. *Journal of medical virology*, 92(6):568–576, 2020.

[71] Tao Zhang, Qunfu Wu, and Zhigang Zhang. Probable pangolin origin of sars-cov-2 associated with the covid-19 outbreak. *Current Biology*, 2020.

[72] Xiaolu Tang, Changcheng Wu, Xiang Li, Yuhe Song, Xinmin Yao, Xinkai Wu, Yuange Duan, Hong Zhang, Yirong Wang, Zhaohui Qian, et al. On the origin and continuing evolution of sars-cov-2. *National Science Review*, 2020.

[73] PI: Kirsten Fisher. Sub-biocrust soil microbial communities from mojave desert, california, united states - 8hms. Sequence Read Archive (SRA) [Internet]. Bethesda (MD): National Library of Medicine (US), National Center for Biotechnology Information; 2009, 1 2020. submitted to JGI at 2019-09-20; Available from: https://www.ncbi.nlm.nih.gov/sra/.

[74] Jennifer Lu, Florian P Breitwieser, Peter Thielen, and Steven L Salzberg. Bracken: estimating species abundance in metagenomics data. *PeerJ Computer Science*, 3: e104, 2017.

[75] Mark Reppell and John Novembre. Using pseudoalignment and base quality to accurately quantify microbial community composition. *PLoS computational biology*, 14(4):e1006096, 2018.

[76] Cole Trapnell, Brian A Williams, Geo Pertea, Ali Mortazavi, Gordon Kwan, Marijke J Van Baren, Steven L Salzberg, Barbara J Wold, and Lior Pachter. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology*, 28(5):511–515, 2010.

[77] Peter Glaus, Antti Honkela, and Magnus Rattray. Identifying differentially expressed transcripts from RNA-seq data with biological variation. *Bioinformatics*, 28(13):1721–1728, 2012.

[78] Naoki Nariai, Osamu Hirose, Kaname Kojima, and Masao Nagasaki. TIGAR: transcript isoform abundance estimation method with gapped alignment of RNA-Seq data by variational Bayesian inference. *Bioinformatics*, page btt381, 2013.

[79] Naoki Nariai, Kaname Kojima, Takahiro Mimori, Yukuto Sato, Yosuke Kawai, Yumi Yamaguchi-Kabata, and Masao Nagasaki. TIGAR2: sensitive and accurate estimation of transcript isoform expression with longer RNA-Seq readsonline. *BMC Genomics*, 15(Suppl 10):S5, 2014.

[80] James Hensman, Panagiotis Papastamoulis, Peter Glaus, Antti Honkela, and Magnus Rattray. Fast and accurate approximate inference of transcript expression from RNA-seq data. *Bioinformatics*, 31(24):3881–3889, 2015.

[81] Julia Salzman, Hui Jiang, and Wing Hung Wong. Statistical modeling of RNA-Seq data. *Statistical science: a review journal of the Institute of Mathematical Statistics*, 26(1), 2011.

[82] Mohsen Zakeri, Avi Srivastava, Fatemeh Almodaresi, and Rob Patro. Improved data-driven likelihood factorizations for transcript abundance estimation. *Bioinformatics*, 33(14):i142–i151, 2017.

[83] Tuuli Lappalainen, Michael Sammeth, Marc R Friedländer, Peter A. C. 't Hoen, Jean Monlong, Manuel A Rivas, Mar Gonzàlez-Porta, Natalja Kurbatova, Thasso Griebel, Pedro G Ferreira, et al. Transcriptome and genome sequencing uncovers functional variation in humans. *Nature*, 501(7468):506–511, 2013.

[84] Donald B Rubin. The bayesian bootstrap. *The annals of statistics*, pages 130–134, 1981.

[85] Cong Ma and Carl Kingsford. Detecting, categorizing, and correcting coverage anomalies of rna-seq quantification. *Cell Systems*, 9(6):589–599, 2019.