STONY BROOK UNIVERSITY

RESEARCH PROFICIENCY EXAMINATION — REPORT

RapMap: A new approach to fast read mapping algorithms with application to RNA-seq analysis

Author:
Avi SRIVASTAVA

Committee:
Prof. Rob PATRO(Advisor)
Prof. Joe MITCHELL
Prof. Adam SIEPEL
Prof. Michael BENDER

A RPE report submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Combine-Lab
Department of Computer Science

July 11, 2016

"Nature uses only the longest threads to weave her patterns, so that each small piece of her fabric reveals the organization of the entire tapestry."

Richard Feynman

STONY BROOK UNIVERSITY

Abstract

Research Proficiency Examination — Report

RapMap: A new approach to fast read mapping algorithms with application to RNA-seq analysis

by Avi SRIVASTAVA

Overview: DNA Sequencing technologies are evolving rapidly, and with them, the requirement for fast and accurate tools to analyze the generated data. The first step of many sequencing analyses requires us to solve the problem of read-alignment. This study discusses the evolution of the tools developed to handle the problem of read-alignment for second generation sequencing and their impact on RNA-seq analysis. The relative speed of these tools motivated the development of `RapMap`; a tool to map RNA-seq *reads* (sequences) to the reference sequence(s) (transcriptome).

Biological Motivation: The alignment of sequencing reads to a transcriptome is a common and important step in many RNA-seq analysis tasks. When aligning RNA-seq reads directly to a transcriptome (as is common in the *de novo* setting or when a trusted reference annotation is available), care must be taken to report the potentially large number of multi-mapping locations per read. This can pose a substantial computational burden for existing aligners, and can considerably slow downstream analysis.

Tool: We introduce a novel concept, quasi-mapping, and an efficient algorithm implementing this approach for mapping sequencing reads to a transcriptome. By attempting only to report the potential loci of origin of a sequencing read, and not the base-to-base alignment by which it derives from the reference, `RapMap`— our tool implementing quasi-mapping— is capable of *mapping* sequencing reads to a target transcriptome substantially faster than existing alignment tools. The algorithm we employ to implement quasi-mapping uses several efficient data structures and takes advantage of the special structure of shared sequence prevalent in transcriptomes to rapidly provide highly-accurate mapping information. We demonstrate how quasi-mapping can be successfully applied to the problems of transcript-level quantification from RNA-seq reads.

Availability: `RapMap` is implemented in `C++11` and is available as open-source software, under GPL v3, at github.

Acknowledgements

I would like to thank my advisor Dr. Rob Patro for his patience and continuous support throughout the project. My fellow students Hirak, Laraib, Nitish, Komal and Mohsin for patiently listening and commenting on the progress of the project in every group meeting. Geet Duggal, Richard Smith-Unna, and Owen Dando for useful discussions regarding various aspects of this work. Also, I'd like to thank the anonymous reviewers whose comments improved the `RapMap` manuscript and exposition.

Contents

List of Figures

List of Tables

List of Abbreviations

ARD	**A**bsolute **R**elative **D**ifference
BLAST	**B**asic **L**ocal **A**lignment **S**earch **T**ool
BWT	**B**urrows-**W**heeler **T**ransform
DFS	**D**epth-**F**irst **S**earch
DP	**D**ynamic-**P**rogram
Indel	**In**setion (and) **D**eletion
MARD	**M**ean **A**bsolute **R**elative **D**ifference
NGS	**N**ext **G**eneration **S**equencing
SA	**S**uffix **A**rray
SGS	**S**econd **G**eneration **S**equencing
SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata
SMEM	**S**uper **M**aximal **E**xact **M**atch
T-DBG	**T**ranscriptome - **d**e **B**ruijn **G**raph
TPEF	**T**rue **P**ositive **E**rror **F**raction
TPME	**T**rue **P**ositive **M**edian **E**rror
VBEM	**V**ariational **B**ayesian - **E**xpectation **M**aximization
wMARD	**w**eighted **M**ean **A**bsolute **R**elative **D**ifference

Chapter 1

Introduction

DNA sequencing is generally performed in three phases (Schadt, Turner, and Kasarskis, 2010): fragmentation, physical-sequencing, and assembly. The first phase of fragmentation breaks the reference DNA into several small pieces and amplifies it into multiple copies, based on the requirements of the analysis. In the physical sequencing phase, individual units (called *bases*) of the fragments are identified in the sequential order to create a read, the number of the contiguous sequence of bases in a read defines its length measured in base-pairs (bp). In the last phase of assembly, the collection of *reads* (called *library*) is analyzed by bioinformatics software to find overlapping regions in a *library* of *reads* and create "most of" the contiguous sequences of the genome. The phrase "most of" is particularly important because based on the species, sequencing technology, and many external factors sequencing can face many different biological and computational challenges and can give different results like collapsing the repetitive sequence of the genome.

FIGURE 1.1: From top to bottom: Single-End Sequencing, Paired-End Sequencing, information retained after Paired-End sequencing.

Physical-sequencing of the *reads* is generally performed in either of the two modes: namely single-end (SE) or paired-end (PE). To give the fragments a notion of direction as shown in Figure 1.1 fragments are sequenced from 5′ end to 3′ end (This refers to the 5′ and 3′ carbons on the sugar presents at each end). In SE sequencing, the fragments are sequenced only from one end. But, in PE sequencing, the fragments are first sequenced from one end for some fixed length, then in another from the opposite end to form a pair of *reads* (called *mate-pairs*). Knowing the distance between mate-pairs of a fragment in PE sequencing helps improve the specificity of the alignment of the fragment, especially when the matching region of the

read in the genome is not unique. In this case, the distance between its mate-pair can help resolve ambiguity.

Sequencing of the Human Genome (Venter et al., 2001) spawned many novel sequencing technologies such as 454 Life sciences (Roche), Illumina (also called Solexa Sequencing), Applied Biosystem's SOLiD, Pacific Biosciences and Oxford Nanopore etc. As a result of these sequencing technologies, scientists have found many biological applications for sequencing a genome, such as studying cancer (cancer genomics), gene regulatory network study and differential expression analysis etc. Most of these biological studies require that we solve the problem of read-alignment before further analysis can be done. Moreover, the accuracy of the downstream analyses greatly depends on how well the problem of read-alignment is solved, which from a computational perspective, makes the problem of read-alignment particularly interesting. The classical problem of read-alignment and its general mathematical notion can be defined as follows:

Given: A set of sequence (called *reads*) R, a reference sequence T, a distance function $d(u, v)$ which gives the distance between two sequence u and v, and ϵ (maximum edit distance) where $\epsilon \in \mathbb{Z}^+$.

$$R = \{r_i : r_i \in \Sigma^{l_i}, \Sigma \in \{A, C, G, T\}, i \in [1, n]\} \tag{1.1}$$

where l_i is the length of the *read* i and n is the total number of *reads*.

$$T = \Sigma^k, \Sigma \in \{A, C, G, T\} \tag{1.2}$$

where k is the length of the reference sequence.

$$d : \Sigma^{|u|} \times \Sigma^{|v|} \rightarrow \mathbb{Z} \tag{1.3}$$

Read Alignment Problem: Find $\forall r_i \in R$, a set of tuples $S = \{(s, p, c) : p \in \{[1, k] \cup \emptyset^1\}\}$ such that substring s of T starting from a position p, can be generated from r_i using the transformation defined by c (commonly encoded as *CIGAR string*) where $d(r_i, s) \leq \epsilon$ and c represents a edit script.

Sequencing technology has evolved very quickly and already has three major generation of sequencers. Sanger-sequencing (First Generation sequencing) (Sanger and Coulson, 1975) was developed by Sanger in 1975 and produces *reads* of 1000 bp . Second generation sequencing (SGS) (Reis-Filho et al., 2009) (sometimes called Next Generation Sequencing) technologies can produce 35bp-400bp (Schatz, Delcher, and Salzberg, 2010) *reads* with very high throughput and much lower per base cost than Sanger-sequencing. Third generation sequencing (Schadt, Turner, and Kasarskis, 2010)(TGS) has its own advantage of read length which is tens of thousand of bp. TGS suffers from a relatively high error rate and produces comparatively fewer reads than NGS, which is important in certain applications requiring high coverage.

The bioinformatics community has put tremendous efforts into building a wide array of different tools to solve the problem of read-alignment, based

[1] Represents no match condition.

on the generation of the sequencer these tools use many different strategies to quickly find potential alignment for the *reads*. Since the field is being explored intensively, it is very hard to discuss all the tools, we have restricted the literature of the report to SGS tools only and discussed them chronologically on the basis of major methodological advancement over previously developed tools in their time.

1.1 Smith-Waterman (Smith and Waterman, 1981)

The Smith-Waterman (Smith and Waterman, 1981) algorithm is a classical algorithm for solving the problem of sequence-alignment which allows both mismatches and indels under a given scoring scheme. It guarantees finding optimal alignment under the time bound of $\boldsymbol{O(|T|l_i)}$ for each read r_i (notations as described before Equation (1.1)). For a set R of *reads* this time bound becomes $\sum_{\boldsymbol{r_i \in R}} \boldsymbol{O(|T|l_i)}$ of which mostly spend on filling DP sub-solutions that have no hope of satisfying the edit distance bound of ϵ and if we use Smith-Waterman directly on the large reference genomes (like Humans) for read-alignment, it becomes infeasible. Due to the large computational cost of the algorithm and the presence of huge numbers of *reads*, motivated scientists to use heuristics (greedy type approach) to solve the problem.

The family of BLAST (Altschul et al., 1990; Altschul et al., 1997) algorithms was one such initiative, initially designed for comparing two long sequences but used also for read-alignment. These algorithms attempt to optimize a specific local similarity measure which uses heuristics to give the similarity $d(u, v)$ between two sequences u and v based on their biological significance. BLAST give approximate alignment results and was more than an order magnitude faster than previous methods with similar accuracy. BLAST follows a hash table based approach for preprocessing the set of *reads* and a simple seed-and-extend strategy on the reference sequence to solve the problem of read-alignment. BLAST maintains a database of a hash table on all the k-means (k length substrings) of the *reads* and traverses the reference sequence to find a hit for seed(s) by checking the database of the hash table (called the *index*). BLAST first joins the seeds without gaps and then refines them by Smith-Waterman for gapped alignment. Generally, in a big reference sequence, only a few subsequences have sufficient similarity to the query; BLAST finds these by filtering irrelevant matches using two threshold parameters S and T. Each seed's similarity score is checked to determine if it's greater or equal to than the parameter T. Then, BLAST tries to extend seed(s) exceeding the threshold using the Smith-Waterman algorithm to get the maximum matching score (under their similarity measure) and reports only those alignments exceeding the similarity threshold S.

Even though BLAST approach was faster than existing methods for read-alignment, it has some shortcomings when viewed in the context of NGS. The index needs to be built for every new read, and for alignment BLAST scans through the entire reference sequence many times (once for each query) resulting in large computational requirements.

1.2 MAQ (Li, Ruan, and Durbin, 2008)

MAQ was initially developed for NGS technologies which produced tens of millions of 30-40bp *reads*. With *reads*, this short, most genomes contain repetitive regions or nearly repetitive regions (edit distance $< \epsilon$), at least as long as these *reads*. Because of the repetitive regions *reads* can align to the reference on multiple positions (called ambiguous alignments) and even a few mutations can make the read align to the wrong location. One of the major shortcomings of the read-alignment tools was that for a better accuracy a lot of *reads* are ignored based on the ambiguity of their alignment. Although conservatively discarding ambiguous *reads* simplifies the read-alignment problem, it leaves out the important information from repetitive regions, which is essential for many biological applications. Instead of just ignoring ambiguous *reads*, MAQ (Li, Ruan, and Durbin, 2008) was one of the first tools to use this information. MAQ uses an approach similar to phred (Ewing and Green, 1998) base-calling, i.e. not to discard the ambiguous *reads* as soon as they are discovered. Instead, for each ambiguously aligned read, MAQ calculates the likelihood of the read being wrongly positioned and assign a quality score to each alignment. Using the posterior error probability of each alignment, more information is retained than just discarding them. In a nutshell, MAQ uses a seed-and-extend type approach for aligning *reads*. It builds a hash on *reads*, parses the reference for a hits, but with modification in the alignment strategy of assigning a mapping quality score for each alignment (which is a measure of the confidence that a read actually comes from the position it is aligned to). Also, MAQ doesn't scan reference sequence for every read, instead, it builds the index on a set of read and scans the reference a fixed (small) number of times.

To select a seed, the first 28bp of each read are hashed using six templates (12 for paired-end reads), sorted and stored in the form of a hash-table (called the *index*). Then the subsequences of the reference sequence are scanned for a hit in the hash-table of all six templates. For each hit, MAQ assigns a mapping quality score, which is the sum of the qualities of the mismatching bases over the length of the whole read, extending from the initial 28bp seed without a gap and with at most 2 mismatches. Later, orphaned alignments (mate-pairs where only one mate gets aligned) are searched for gapped alignment using Smith-Waterman in the region of mapped mate-pair. The region for gapped alignments is taken to be two standard deviations of the distribution of the distance between aligned PE *reads*. MAQ was also special in a sense that it utilizes the mate-pair information of paired-end *reads* to correct potentially wrong alignments and accurately align *reads* to repetitive sequences. The idea used by MAQ represented an important advancement but it was still far from satisfactorily solving the problem of read-alignment. For example, MAQ always reports a single alignment and in the case of equally good alignments, it randomly picks one and assigns a score of 0 (identifiable by downstream analysis). Also, no special indexing technique is used for the reference, and even for single end read the template hashing ensures only 2 mismatch hits, as every k-mismatch hit requires $\binom{2k}{k}$ templates. Overall, it takes 1100 CPU hours for MAQ to align 100Million 35bp PE *reads*. This is still a high computational cost given the exponentially increasing pace of sequencing.

At the same time, SOAP (Li et al., 2008) was developed to handle the

problem of both ungapped and gapped alignment. Unlike MAQ, SOAP hashes the reference sequence into the memory and builds an index table for the reference sequence. It uses 2 bits per base encoding[2] to convert the read and the reference to numeric data-type and to obtain a matching score, a simple XOR of the reference subsequence with the *reads* is done. It allows either a certain number of mismatches or one continuous gap for aligning a read to the reference sequence. As an example, to acquire hits with two mismatches, a read is split into four fragments and with the use of the pigeon-hole principle it made sure that at least two fragments must have no mismatches. By using all six combinations of possible fragments from a read, we can find all the hits with two mismatches. Since mismatches and gapped are not allowed simultaneously, for gapped hits SOAP uses an enumeration algorithm which tries to insert a continuous gap or deletes a fragment at each possible position in a read. At the end, the best hit for each read having the minimal number of mismatches or smaller gap is reported. Moreover, an option for specifying the number of mismatches makes it more versatile. However, tools at this time were facing a dilemma: if an index is built on *reads*, then there is overhead for scanning the whole genome many times when the number of alignments is small , but, if the index is built on the genome, then the memory requirements can become immense.

1.3 Bowtie (Langmead et al., 2009)

The next significant advancement in the world of read-alignment came with Bowtie (Langmead et al., 2009) which, astonishingly, at the time was able to map 25M *reads* per CPU hour on the human genome using just 1.3G of memory. Bowtie's speed derived largely from using a compressed index of the reference genome. Bowtie index uses the Burrows-Wheeler transform (BWT) (Burrows and Wheeler, 1994) and FM-index (Ferragina and Manzini, 2000; Ferragina and Manzini, 2001). The small size of the resulting index makes it possible to load it into the memory of a computer with as little as 2G of RAM. The main advantage of building the index on the reference instead of *reads* is that the index can be build once and reused many times (e.g. for different sequencing experiments). Bowtie can find exact matches of any length l in $O(l)$ time and can report them all in $O(l) + k$ time where k is number of occurrences, using the "backward search" (Ferragina and Manzini, 2000) algorithm. For the inexact match(with mismatches) problem bowtie essentially builds inexact alignment by finding multiple supporting exact matches.

Inexact match search (done only when no exact match is found) is performed by using depth-first search (DFS) on the query, and backtracking by replacing and inducing mismatches. Replacement is done using the minimum quality value at that base and ties are broken randomly. An exhaustive search for a read has an exponential time bound in the number of mismatches m. To avoid full-length backtracking, Bowtie uses a BWT index on both the forward and reversed reference and stops backtracking after reaching the middle of the read which makes backtracking faster. The reverse reference can work well for one mismatch, but is not able to avoid excessive

[2]Since we have only 4 different bases to encode i.e. A, C, T, G

backtracking for more than one mismatch. To prevent further backtracking, Bowtie avoids low-quality alignments by stopping backtracking at around 125 backtracks and allowing default mismatch of 2. One of the other advantages of Bowtie was that one can increase the sensitivity of the tool by increasing the default mismatch rate but with the trade-off that more time will be required because DFS will go deeper during the backtracking.

As much as the strategy of Bowtie was novel (though similar approach was taken by SOAP2 (Li et al., 2009)) and vastly improve the state-of-the-art in terms of running time, it started to fall short with the improvements in the sequencing throughput and read-length of the NGS, and also the launch of big research projects required much faster short-read alignment methods to handle the data analysis of large-scale sequencing experiments. With large reads ($\geq$100bp) becoming more common, Bowtie allowing any number of mismatches in the non-seed portion of the read becomes less well-suited to the data. Also, Bowtie does not allow gapped alignment which can be very important in some contexts.

Extending the concept and using the Suffix-Array (SA), BWT and FM-index; BWA (Li and Durbin, 2009) solved the problem of inexact matching by allowing gapped alignment. For exact matching, it simply finds the interval of the suffix array where the query substring occurs (which can be done in linear time using backward search (Ferragina and Manzini, 2000)) independent to the length of the reference. For inexact matches, the process is more subtle. Given the maximum no. of indels/mismatches allowed, BWA preprocesses the query string with the BWT of the reverse (not reverse complementation) of the reference. Which helps BWA to avoid unnecessary backward depth first traversal resulting in faster and accurate alignment. Many other implementation details like limiting the number of differences in the first few tens of bases of the read were done by BWA to increase the alignment rate and accuracy even further.

1.4 BWA-mem (Li, 2013)

In the timeline of read-sequencing, sequencers started producing Terabytes of data and handling them became tedious even for optimized tools like BWA. The need for faster methods resulted in even faster tools like `Bowtie 2`, BWA-mem (Li, 2013) and MrsFast (Hach et al., 2010). `Bowtie 2` improved over Bowtie, by allowing gapped alignment using the seed and extend (difference in extension stage uses the intelligent dynamic program) approach, `Bowtie 2` benefits extensively from using single-instruction-multiple-data (SIMD) parallel processing to vectorize many computations. It is $\geq$ 3 times faster than BWA. BWA also continued to evolve, and BWA-mem introduced a new seed-chain-extend paradigm. In this approach, one first finds a seed for the alignment using super maximal exact matches (SMEM) based on the method of BWA, and a chain of seeds is formed based on the inter-seed distances. This often avoids useless and costly seed extension step. At the end, alignment is performed using the affine gap penalty dynamic program on the relevant chains. MrsFast is another notable tool which is cache-oblivious and indexed both reference and *reads* using kmer-based approach to provide read alignments.

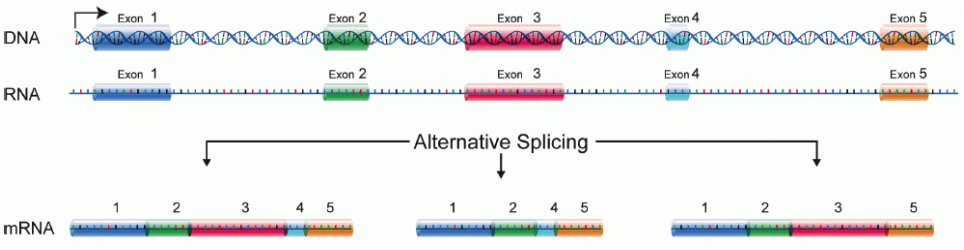

FIGURE 1.2: From Top to bottom (*Alternative Splicing*). : DNA, RNA and mRNA.

RNA-seq (Wang, Gerstein, and Snyder, 2009) or transcriptome sequencing, produces *reads* from transcribed (exonic regions) regions of the RNA. As explained in Figure 1.2 first double stranded DNA is converted to a single strand of RNA. Then, to generate mRNA from RNA different coding regions (called *exon*) of the RNA are fused together to form different types of mRNA (the process is called *alternative splicing*). In mRNA, the fusion point of two different exonic regions of the RNA is called as *splice junction*. If *reads* from RNA-seq are aligned to the genomic reference sequence, a read can map to splice junction of mRNA which in the genomic region could be tens of thousands of bases apart and current methods of read-alignment will fail.

This divided the alignment problem into two different regimes namely spliced aligners and unspliced aligners. Spliced read aligners try to solve the problem of aligning to a target reference genome where some *reads* can span exons junctions. These tools are called splice read aligners since they have to find alignments to exon-exon junctions which may be tens of thousand bases apart in the reference. On the other hand, they typically encounter very little multi-mapping since most *reads* have a single genomic locus of origin. Some of the notable spliced aligners are like TopHat (Trapnell, Pachter, and Salzberg, 2009) , `STAR` (Dobin et al., 2013), Subread aligner (Liao, Smyth, and Shi, 2013) which uses novel seed and vote strategy and the recently published HISAT (Kim, Langmead, and Salzberg, 2015) which uses a hierarchical FM index to allow very fast mapping of both DNA-seq and RNA-seq *reads* with only small memory requirements.

TABLE 1.1: Short-Read Aligners/Mappers property. MAQ represents if mapping quality is used by tool or not

Tool	Indexing	Gapped	Paired-end	MAQ
BLAST	hash *reads*	No	No	Yes
MAQ	hash *reads*	Yes	Yes	Yes
SOAP	hash reference	Yes	Yes	Yes
Bowtie	FM-index	No	Yes	No
BWA	FM-index	Yes	Yes	No
BWA-mem	FM-index	Yes	Yes	No
pseudo-mapping	T-DBG	Yes	Yes	No
quasi-mapping	Suffix Array	Yes	Yes	No

Transcriptome aligners are same as unspliced aligners(i.e. they work on transcriptome level) but they also intelligently handle high multi-mapping

rate in the alignments due to the presense of isoforms of the same exonic regions in mRNA. As can be seen in Figure 1.2, *exon1* (in blue) is present in all the mRNA and if a read is generated from the region of exon1 from any one mRNA it'll get aligned to each and everyone of them equally well. The modified problem of read-alignment for transcriptome aligners can be defined as follows:

Given: A set of sequences (called *reads* or RNA-seq *reads*) R, a set (called transcriptome) T of reference sequences (called transcripts) t, a distance function $d(u, v)$ which gives the distance between two sequence u and v, and ϵ (maximum edit distance) where $\epsilon \in \mathbb{Z}^+$.

$$R = \{r_i : r_i \in \Sigma^{l_i}, \Sigma \in \{A, C, G, T\}, i \in [1, n]\} \tag{1.4}$$

where l_i is the length of the *read* i and n is the total number of *reads*.

$$T = \{t_i : t_i \in \Sigma^{\ell_{t_i}}, \Sigma \in \{A, C, G, T\}, i \in [1, k]\} \tag{1.5}$$

where ℓ_{t_i} is the length of the transcript i and k is the total number of transcripts.

$$d : \Sigma^{|u|} \times \Sigma^{|v|} \to \mathbb{Z} \tag{1.6}$$

Read Alignment Problem for Transcriptome: Find $\forall r_i \in R$, a set of tuple $S = \{(s, p, t, c) : p \in \{[1, k] \cup \emptyset^3\}\}$ such that substring s of transcript t starting from a position p, can be generated from r_i using the transformation defined by c (also called *CIGAR string*) where $d(r_i, s) \leq \epsilon$ and c represents a collection of match, mismatch, insertion and deletion (indel).

Transcriptome aligners focus primarily, on a different problem from spliced-aligners (though some of the spliced aligners can be used as transcriptome aligner) since they have to align *reads* to the transcriptome instead of the genome. Specifically, they face a huge rate of multi-mapping in *reads* but they don't have to handle spliced events (since they assume a reference that consists of the already spliced (i.e. mature) transcripts). Transcriptome alignment can also be used after *de novo* transcriptome assembly and does not rely on the knowledge of the reference genome of the organism. Table 1.1 (Li and Homer, 2010) give a gist of some of the tools and their constituent properties.

1.5 Salmon (Patro, Duggal, and Kingsford, 2015)

Salmon solved an important problem of quantification by using a novel concept of lightweight-alignemnt (Patro, Duggal, and Kingsford, 2015). The main motivation behind lightweight alignment was the realization that in some special biological application (in the case of Salmon it was the quantification of transcript abundances from RNA-seq *reads*), we do not require the actual alignment between the query sequence (i.e. c in our problem formulation) and the reference. Rather, simply knowing the transcripts

[3] Represents no match condition.

(and the position in that transcript) where each match reasonably well is sufficient to solve the problem. Salmon attempts to find these mapping locations using BWA-mem type seed-chain-extend approach i.e. seeding through SMEM, chaining MEM and SMEM together and extending it to only exact matches.

The introduction of lightweight-alignment further divided the transcriptome aligners into two different regimes, namely aligners and mappers. Aligners (as discussed in Section 1.4) works by generating CIGAR string which is a base-to-base correspondence between two query strings. But, in comparison to alignment, mappers does not find CIGAR strings, instead for each read r_i they just report transcript t and the position p in t where r_i gets aligned "sufficiently well" (*Alignment vs. Mapping*)[4].

Using a different, alignment-free approach, a recent quantification tool `Kallisto` (Bray et al., 2015) makes use of Transcriptome-de-Bruijn graph(T-DBG) to compute so-called pseudo-alignments. `Kallisto` builds a de-Bruijn graph on the transcriptome and labels each unique k-mer region on the graph with its constituent transcripts. During the pseudo-alignment stage, a procedure is used to extract a subset of k-mers from the read and intersect them with the T-DBG to determine a "compatible" set of transcripts. In some sense, pseudo-alignment goes even further than mapping in that the pseudo-alignments themselves give no information about where (position) or how (orientation) a read maps to a transcript (though pseudo-alignments can be pre-processed, at some extra cost to approximate this extra information). The field of read-mapping is under constant development and we have developed a novel mapping technique, called quasi-mapping, using the suffix array, which is discussed in the following section.

[4]Since the problem is less well-studied than alignment and a reasonable determination of what constitutes "sufficiently well" is still a reasonable topic for discussion and debate

Chapter 2

RapMap

When reads are aligned to a collection of reference sequences that share a substantial amount of sub-sequence (near or exact repeats), a single read can have many potential alignments, and considering all such alignment can be crucial for downstream analysis (e.g. considering all alignment locations for a read within a transcriptome for the purpose of quantification Li and Dewey, 2011, or when attempting to cluster *de novo* assembled contigs by shared multi-mapping reads (Davidson and Oshlack, 2014)). However, reporting multiple potential alignments for each read is a difficult task, and tends to substantially slow down even very efficient alignment tools. Yet, in many cases, all of the information provided by the alignments is not necessary. For example, in the transcript analysis tasks mentioned above, simply the knowledge of the transcripts and positions to which a given read maps well is sufficient to answer the questions being posed. In support of such "analysis-oriented" computation, we propose a novel concept, called quasi-mapping, and an efficient algorithm implementing quasi-mapping (exposed in the software tool `RapMap`) to solve the problem of mapping sequenced reads to a target transcriptome. This algorithm is *considerably* faster than state-of-the-art aligners, and achieves its impressive speed by exploiting the structure of the transcriptome (without requiring an annotation), and eliding the computation of full-alignments (e.g. `CIGAR` strings). Further, our algorithm produces mappings that meet or exceed the accuracy of existing popular aligners under different metrics of accuracy. Finally, we demonstrate how the mappings produced by `RapMap` can be used in the downstream analysis task of transcript-level quantification from RNA-seq data, by modifying the Sailfish (Patro, Mount, and Kingsford, 2014) tool to take advantage of quasi-mappings, as opposed to raw k-mer counts, for transcript quantification.

2.1 Methods

The quasi-mapping concept, implemented in the tool `RapMap`, is a new mapping technique to allow the rapid and accurate mapping of sequenced fragments (single or paired-end reads) to a target transcriptome. `RapMap` exploits a combination of data structures — a hash table, suffix array (SA), and efficient rank data structure. It takes into account the special structure present in transcriptomic references, as exposed by the suffix array, to enable ultra-fast and accurate determination of the likely loci of origin of a sequencing read. Rather than a standard alignment, quasi-mapping produces what we refer to as fragment *mapping* information. In particular, it provides, for each query (fragment), the reference sequences (transcripts),

strand and position from which the query may have likely originated. In many cases, this mapping information is sufficient for downstream analysis. For example, tasks like transcript quantification, clustering of *de novo* assembled transcripts, and filtering of potential target transcripts can be accomplished with this mapping information. However, this method does not compute the base-to-base alignment between the query and reference. Thus, such mappings may not be appropriate in every situation in which alignments are currently used (e.g. variant detection).

We note here that the concept of quasi-mapping shares certain motivations with the notions of lightweight-alignment (Patro, Duggal, and Kingsford, 2015) and pseudo-alignment (Bray et al., 2015). Yet, all three concepts — and the algorithms and data structures used to implement them — are distinct and, in places, substantially different. Lightweight-alignment scores potential matches based on approximately consistent chains of super-maximal exact matches shared between the query and targets. Therefore, it typically requires some more computation than the other methods, but allows the reporting of a score with each returned mapping and a more flexible notion of matching. Pseudo-alignment, as implemented in `Kallisto`, refers only to the process of finding *compatible* targets for reads by determining approximately matching paths in a colored De Bruijn graph of a pre-specified order. Among compatible targets, extra information concerning the mapping (e.g. position and orientation) can be extracted *post-hoc*, but this requires extra processing, and the resulting mapping is no longer technically a pseudo-alignment. Quasi-mapping seeks to find the *best* mappings (targets and positions) for each read, and does so (approximately) by finding minimal collections of dynamically-sized, right-maximal matching contexts between target and query positions. The algorithm for quasi-mapping that we describe below achieves this using a combination of a k-mer lookup table and a generalized suffix array. While each of these approaches provide some insight into the problems of alignment and mapping, they represent distinct concepts and exhibit unique characteristics in terms of speed and accuracy, as demonstrated below[1].

An algorithm for Quasi-mapping The algorithm we use for quasi-mapping makes use of two main data structures, the generalized suffix array (Manber and Myers, 1993) $\texttt{SA}[T]$ of the transcriptome T, and a hash table h mapping each k-mer occurring in T to its suffix array interval (by default $k = 31$). Additionally, we must maintain the original text T upon which the suffix array was constructed, and the name and length of each of the original transcript sequences. T consists of a string in which all transcript sequences are joined together with a special separator character. Rather than designating a separate terminator $\$_i$ for each reference sequence in the transcriptome, we make use of a single separator $\$$, and maintain an auxiliary rank data structure which allows us to map from an arbitrary position in the concatenated text to the index of the reference transcript in which it appears. We use the rank9b algorithm and data structure of Vigna (2008) to perform the rank operation quickly.

[1] We do not compare against lightweight-alignment here, as no stand-alone implementation of this approach is currently available

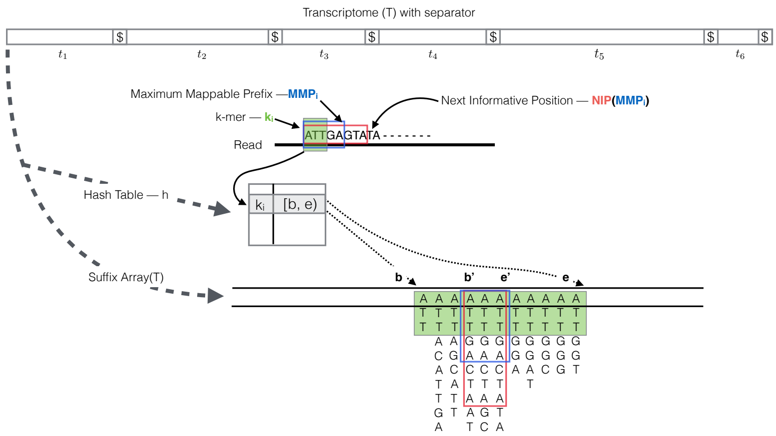

FIGURE 2.1: The transcriptome (consisting of transcripts $t_1, \ldots, t_6$) is converted into a \$-separated string, T, upon which a suffix array, $\texttt{SA}[T]$, and a hash table, h, are constructed. The mapping operation begins with a k-mer (here, $k = 3$) mapping to an interval $[b, e)$ in $\texttt{SA}[T]$. Given this interval and the read, MMP_i and $\texttt{NIP}(\text{MMP}_i)$ are calculated as described in section 2.1. The search for the next hashable k-mer begins k bases before $\texttt{NIP}(\text{MMP}_i)$.

Quasi-mapping determines the mapping locations for a query read r through repeated application of (1) determining the next hash table k-mer that starts past the current query position, (2) computing the maximum mappable prefix (MMP) of the query beginning with this k-mer, and then (3) determining the next informative position (NIP) by performing a longest common prefix (LCP) query on two specifically chosen suffixes in the suffix array.

The algorithm begins by hashing the k-mers of r, from left-to-right (a symmetric procedure can be used for mapping the reverse-complement of a read), until some k-mer k_i — the k-mer starting at position i within the read — is present in h and maps to a valid suffix array interval. We denote this interval as $\texttt{I}(k_i) = [b, e)$. Because of the lexicographic order of the suffixes in the suffix array, we immediately know that this k-mer is a prefix of all of the suffixes appearing in the given interval. However, it may be possible to extend this match to some longer substring of the read beginning with k_i. In fact, the longest substring of the read that appears in the reference and is prefixed by k_i is exactly the maximum mappable prefix (MMP) (Dobin et al., 2013) of the suffix of the read beginning with k_i. We call this maximum mappable prefix MMP_i, and note that it can be found using a slight variant of the standard suffix array binary search (Manber and Myers, 1993) algorithm. For speed and simplicity, we implement the “simple accelerant” binary search variant of Gusfield (1997). Since we know that any substring that begins with k_i must reside in the interval $[b, e)$, we can restrict the MMP_i search to this region of the suffix array, which is typically very small.

After determining the length of MMP_i within the read, one could begin the search for the next mappable suffix array interval at the position following this MMP. However, though the current substring of the read will differ from all of the reference sequence suffixes at the base following MMP_i, the

suffixes occurring at the lower and upper bounds of the suffix array interval corresponding to MMP_i may not differ from each other (See Figure 2.1). That is, if $\texttt{I}(\text{MMP}_i) = [b', e')$ is the suffix array interval corresponding to MMP_i, it is possible that $|\texttt{LCP}(\texttt{T}[\texttt{SA}[b']], \texttt{T}[\texttt{SA}[e'-1]]))| > |\text{MMP}_i|$. In this case, it is most likely that the read and the reference sequence bases following MMP_i disagree as the result of a sequencing error, not because the (long) MMP discovered between the read and reference is a spurious match. Thus, beginning the search for the next MMP at the subsequent base in the read may not be productive, since the matches for this substring of the query may not be informative — that is, such a search will likely return the same (relative) positions and the set of transcripts. To avoid querying for such substrings, we define and make use of the notion of the next informative position (NIP). For a maximum mappable prefix MMP_i, with $\texttt{I}(\text{MMP}_i) = [b', e')$, we define $\texttt{NIP}(\text{MMP}_i) = |\texttt{LCP}(\texttt{T}[\texttt{SA}[b']], \texttt{T}[\texttt{SA}[e'-1]]))| + 1$. Intuitively, the next informative position of prefix MMP_i is designed to return the next position in the query string where a suffix array search is likely to yield a set of transcripts different from those contained in $\texttt{I}(\text{MMP}_i)$. To compute the longest common prefix between two suffixes when searching for the NIP, we use the "direct min" algorithm of Ilie, Navarro, and Tinta (2010). We found this to be the fastest approach. Additionally, it doesn't require the maintenance of an LCP array or other auxiliary tables aside from the standard suffix array.

Given the definitions we have explained above, we can summarize the quasi-mapping procedure as follows (an illustration of the mapping procedure is provided in Figure 2.1). First, a read is scanned from left to right (a symmetric procedure can be used for mapping the reverse-complement of a read) until a k-mer k_i is encountered that appears in h. A lookup in h returns the suffix array interval $\texttt{I}(k_i)$ corresponding to the substring of the read consisting of this k-mer. Then, the procedure described above is used to compute MMP_i and $\ell = \texttt{NIP}(\text{MMP}_i)$. The search procedure then advances to position $i + \ell - k$ in the read, and again begins hashing the k-mers it encounters. This process of determining the MMP and NIP of each processed k-mer and advancing to the next informative position in the read continues until the next informative position exceeds position $l_r - k$ where l_r is the length of the read r. The result of applying this procedure to a read is a set $S = \{(q_0, o_0, [b_0, e_0)), (q_1, o_1, [b_1, e_1)), \ldots\}$ of query positions, MMP orientations, and suffix array intervals, with one such triplet corresponding to each MMP.

The final set of mappings is determined by a consensus mechanism. Specifically, the algorithm reports the intersection of transcripts appearing in all hits — i.e. the set of transcripts that appear (in a consistent orientation) in every suffix array interval appearing in S. These transcripts, and the corresponding strand and location on each, are reported as *quasi-mappings* of this read. These mappings are reported in a `samtools`-compatible format in which the relevant information (e.g. target id, position, strand, pair status) is computed from the mapping. We note that alternative consensus mechanisms, both more and less stringent, are easy to enforce given the information contained in the hits (e.g. enforcing that the hits are co-linear with respect to both the query and reference). However, below, we consider this simple consensus mechanism.

Intuitively, `RapMap`'s combination of speed and accuracy result from the

manner in which it exploits the nature of exactly repeated sequence that is prevalent in transcriptomes (either as a result of alternative splicing or paralogous genes). In addition to an efficient search for MMPs and NIPs, the suffix array allows RapMap to encode exact matches between the query and many potential transcripts very efficiently (in the form of "hits"). This is because all reference locations for a given MMP appear in consecutive entries of the suffix array, and can be encoded efficiently by simply recording the suffix array interval corresponding to this MMP. By aggressively filtering the hits to determine the set of "best" matching transcripts and positions, RapMap is able to quickly discard small matches that are unlikely to correspond to a correct mapping. Similarly, the large collection of exact matches that appear in the reported mapping are very likely to appear in the alignment mapping (were the actual alignments to be computed). In some sense, the success of the strategy adopted by RapMap further validates the claim of Liao, Smyth, and Shi (2013) that the seed-and-vote paradigm can be considerably more efficient than the seed-and-extend paradigm, as RapMap adopts neither of these paradigms directly, but its approach is more similar to the former than the latter.

In the next section, we analyze how this algorithm for quasi-mapping, as described above, compares to other aligners in terms of speed and mapping accuracy.

2.2 Mapping speed and accuracy

To test the practical performance of quasi-mapping, we compared RapMap against a number of existing tools, and analyzed both the speed and accuracy of these tools on synthetic and experimental data. Benchmarking was performed against the popular aligners Bowtie 2 (Langmead and Salzberg, 2012) (v2.2.6) and STAR (Dobin et al., 2013) (v2.5.0c) and the recently-introduced pseudo-alignment procedure used in the quantification tool Kallisto (Bray et al., 2015) (v0.42.4). All experiments were scripted using Snakemake (Köster and Rahmann, 2012) and 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. Wall-clock time was recorded using the `time` command.

In our testing, we find that Bowtie 2 generally performs well in terms of reporting the true read origin among its set of multi-mapping locations. However, it takes considerably longer and tends to return a larger set of multi-mapping locations than the other methods. In comparison to Bowtie 2, STAR is *substantially* faster but somewhat less accurate. RapMap achieves accuracy comparable or superior to Bowtie 2, while simultaneously being much faster than even STAR. Kallisto is similar to (slightly slower than) RapMap in terms of single-threaded speed, and exhibits accuracy very similar to that of STAR. For both RapMap and Kallisto, simply writing the output to disk tends to dominate the time required for large input files with significant multi-mapping (though we eliminate this overhead when benchmarking). This is due, in part, to the verbosity of the standard SAM format in which results are reported, and suggests that it may be worth developing a more efficient and succinct output format for mapping information.

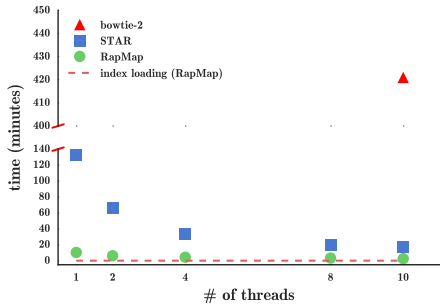

FIGURE 2.2: The time taken by `Bowtie 2`, `STAR` and `RapMap` to process the synthetic data using varying numbers of threads. `RapMap` processes the data substantially faster than the other tools, while providing results of comparable or better accuracy.

2.2.1 Speed and accuracy on synthetic data

To test the accuracy of different mapping and alignment tools in a scenario where we know the true origin of each read, we generated data using the Flux Simulator (Griebel et al., 2012). This synthetic dataset was generated for the human transcriptome from an annotation taken from the ENSEMBL (Cunningham et al., 2015) database consisting of $86,090$ transcripts corresponding to protein-coding genes. The dataset consists of ~ 48 million 76 base pair, paired-end reads. The detailed parameters used for the Flux Simulator can be found in Appendix A.2.

When benchmarking these methods, reads were aligned directly to the transcriptome, rather than to the genome. This was done because we wish to benchmark the tools in a manner that is applicable when the reference genome may not even be known (e.g. in *de novo* transcriptomics). The parameters of `STAR` (see Appendix A.1) were adjusted appropriately for this purpose (e.g. to dis-allow introns etc.). Similarly, `Bowtie 2` was also used to align reads directly to the target transcriptome; the parameters for `Bowtie 2` are given in Appendix A.1.

Mapping speed

We wish to measure, as directly as possible, just the time required by the mapping algorithms of the different tools. Thus, when benchmarking the runtime of different methods, we do not save the resulting alignments to disk. Further, to mitigate the effect of "outliers" (a small number of reads which map to a very large number of low-complexity reference positions), we bound the number of different transcripts to which a read can map to be 200.

Additionally, we have also benchmarked `Kallisto`, but have not included the results in Figure 2.2, as the software, unlike the other methods, does not allow multi-threaded execution if mappings are being reported. Thus, we ran `Kallisto` with a single thread, using the `-pseudobam` flag and redirecting output to `/dev/null` to avoid disk overhead. `Kallisto` requires 17.87m to map the 48M simulated reads, which included <1m of quantification time. By comparison, `RapMap` required 11.65m to complete with a single thread.

TABLE 2.1: Accuracy of aligners/mappers under different metrics

	Bowtie 2	Kallisto	RapMap	STAR
reads aligned	47579567	44804857	47613536	44711604
recall	97.41	91.60	97.49	91.35
precision	98.31	97.72	98.48	97.02
F1-score	97.86	94.56	97.98	94.10
FDR	1.69	2.28	1.52	2.98
hits per read	5.98	5.30	4.30	3.80

Finally, we note Kallisto, STAR and RapMap require 2-3× the memory of Bowtie 2, but all of the methods tested here exhibit reasonable memory usage. The synthetic set of 48 million reads can be mapped to an index of the entire human transcriptome on a typical laptop with 8 GB of RAM.

As Figure 2.2 illustrates, RapMap out-performs both Bowtie 2 and STAR in terms of speed by a substantial margin, and finishes mapping the reads with a single thread faster than STAR and Bowtie 2 with 10 threads. We consider varying the number of threads used by RapMap and STAR to demonstrate how performance scales with the number of threads provided. On this dataset, RapMap quickly approaches peak performance after using only a few threads. We believe that this is not due to limits on the scalability of RapMap, but rather because the process is so quick that, for a dataset of this size, simply reading the index constitutes a large (and growing) fraction of the total runtime (dotted line) as the number of threads is increased. Thus, we believe that the difference in runtime between RapMap and the other methods may be even larger for datasets consisting of a very large number of reads, where the disk can reach peak efficiency and the multi-threaded input parser (we use the parser from the Jellyfish (Marçais and Kingsford, 2011) library) can provide input to RapMap quickly enough to make use of a larger number of threads. Since running Bowtie 2 with each potential number of threads on this dataset is very time-consuming, we only consider Bowtie 2's runtime using 10 threads.

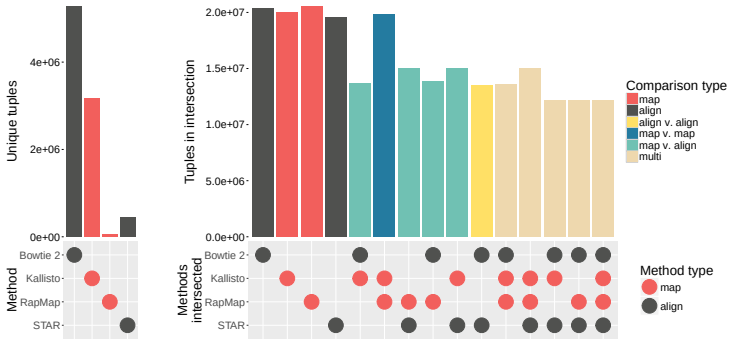

FIGURE 2.3: Mapping agreement between subsets of Bowtie 2, STAR, Kallisto and RapMap.

Mapping accuracy

Since the Flux Simulator records the true origin of each read, we make use of this information as ground truth data to assess the accuracy of different methods. However, since a single read may have multiple, equally-good alignments with respect to the transcriptome, care must be taken in defining accuracy-related terms appropriately. A read is said to be correctly mapped by a method (a true positive) if the set of transcripts reported by the mapper for this read contains the true transcript. A read is said to be incorrectly mapped by a method (a false positive) if it is mapped to some set of 1 or more transcripts, none of which are the true transcript of origin. Finally, a read is considered to be incorrectly un-mapped by a method (a false negative) if the method reports no mappings, but the transcript of origin is in the reference. Given these definitions, we report precision, recall, F1-Score and false discovery rate (FDR) in Table 2.1 using the standard definitions of these metrics. Additionally, we report the average number of "hits-per-read" (hpr) returned by each of the methods. Ideally, we want a method to return the smallest set of mappings that contains the true read origin. However, under the chosen definition of a true positive mapping, the number of reported mappings is not taken into account, and a result is considered a true positive so long as it contains the actual transcript of origin. The hpr metric allows one to assess how many *extra* mappings, on average, are reported by a particular method.

As expected, `Bowtie 2`— perhaps the most common method of directly mapping reads to transcriptomes — performs very well in terms of precision and recall. However, we find that `RapMap` yields very similar (in fact, slightly better) precision and recall. `STAR` and `Kallisto` obtain similar precision to `Bowtie 2` and `RapMap`, but have a lower recall. `STAR` and `Kallisto` perform similarly in general, though `Kallisto` achieves a lower (better) FDR than `STAR`. Taking the F1-score as a summary statistic, we observe that all methods perform reasonably well, and that, in general, alignment-based methods do not seem to be more accurate than mapping-based methods. We also observe that `RapMap` yields very accurate mapping results that match or exceed those of `Bowtie 2`.

Additionally, we tested the impact of noisy reads (i.e. reads not generated from the indexed reference) on the accuracy of the different mappers and aligners. To create these background reads, we use a model inspired by (Gilbert et al., 2004), in which reads are sampled from nascent, un-spliced transcripts. The details of this experiment are included in Appendix A.3.

2.2.2 Speed and concordance on experimental data

We also explore the concordance of `RapMap` with different mapping and alignment approaches using experimental data from the study of Cho et al. (2014) (NCBI GEO accession `SRR1293902`). The sample consists of $\sim$ 26 million 75 base-pair, paired-end reads sequenced on an Illumina HiSeq.

Since we do not know the true origin of each read, we have instead examined the agreement between the different tools (see Figure 2.3). Intuitively, two tools agree on the mapping locations of a read if they align / map this read to the same subset of the reference transcriptome (i.e. the

same set of transcripts). More formally, we define the elements of our universe, $\mathcal{U}$, to be tuples consisting of a read identifier and the set of transcripts returned by a particular tool. For example, if, for read r_i, tool A returns alignments to transcripts $\{t_1, t_2, t_3\}$ then $e_{Ai} = (r_i, \{t_1, t_2, t_3\}) \in \mathcal{U}$. Similarly, if tool B maps read r_i to transcripts $\{t_2, t_3, t_4\}$ then $e_{Bi} = (r_i, \{t_2, t_3, t_4\}) \in \mathcal{U}$. Here, tools A and B do not agree on the mapping of read r_i. Given a universe $\mathcal{U}$ thusly-defined, we can employ the normal notions of set intersection and difference to explore how different subsets of methods agree on the mapping locations of the sequenced reads. These concordance results are presented in Figure 2.3, which uses a bar plot to show the size of each set of potential intersections between the results of the tools we consider. In Figure 2.3 the dot matrix below the bar plot identifies the tools whose results are intersected to produce the corresponding bar. Tools producing mappings and alignments are denoted with black and red dots and bars respectively. The left bar plot shows the size of the unique tuples produced by each tool (alignments / mappings that do not match with any other tool). The right bar plot shows the total number of tuples produced by each tool, and well as the concordance among all different subsets of tools.

Under this measure of agreement, `RapMap` and `Kallisto` appear to agree on the exact same transcript assignments for the largest number of reads. Further, `RapMap` and `Kallisto` have the largest pairwise agreements with the aligners (`STAR` and `Bowtie 2`) — that is, the traditional aligners exactly agree more often with these tools than with each other. It is important to note that one possible reason we see (seemingly) low agreement between `Bowtie 2` and other methods is because the transcript alignment sets reported by `Bowtie 2` are generally larger (i.e. contain more transcripts) than those returned by other methods, and thus fail to qualify under our notion of agreement. This occurs, partially, because `RapMap` and `Kallisto` (and to some extent `STAR`) do not tend to return sub-optimal multi-mapping locations. However, unlike `Bowtie 1`, which provided an option to return only the best "stratum" of alignments, there is no way to require that `Bowtie 2` return only the best multi-mapping locations for a read. We observe similar behavior for `Bowtie 2` (i.e. that it returns a larger set of mapping locations) in the synthetic tests as well, where the average number of hits per read is higher than for the other methods (see Table 2.1). In terms of runtime, `RapMap`, `STAR` and `Bowtie 2` take 3, 26, and 1020 minutes respectively to align the reads from this experiment using 4 threads. We also observed a similar trend in terms of the average number of hits per read here as we did in the synthetic dataset. The average number of hits per read on this data were 4.56, 4.68, 4.21, 7.97 for `RapMap`, `Kallisto`, `STAR` and `Bowtie 2` respectively.

2.3 Application of quasi-mapping for transcript quantification

While mapping cannot act as a stand-in for full alignments in all contexts, one problem where similar approaches have already proven very useful is transcript abundance estimation. Recent work (Patro, Mount, and Kingsford, 2014; Zhang and Wang, 2014; Bray et al., 2015; Patro, Duggal, and Kingsford, 2015) has demonstrated that full alignments are not necessary

to obtain accurate quantification results. Rather, simply knowing the transcripts and positions where reads may have reasonably originated is sufficient to produce accurate estimates of transcript abundance. Thus, we have chosen to apply quasi-mapping to transcript-level quantification as an example application, and have implemented our modifications as an update to the Sailfish (Patro, Mount, and Kingsford, 2014) software, which we refer to as quasi-Sailfish. These changes are present in the Sailfish software from version `0.7` forward. Here, we compare this updated method to the transcript-level quantification tools RSEM (Li et al., 2010), Tigar2 (Nariai et al., 2014) and `Kallisto` (Bray et al., 2015), the last of which is based on the pseudo-alignment concept mentioned above.

2.3.1 Transcript quantification

In an RNA-seq experiment, the underlying transcriptome consists of M transcripts and their respective counts. The transcriptome can be represented as a set $\mathcal{X} = \{(t_1, \ldots, t_M), (c_1, \ldots, c_M)\}$, where t_i denotes the nucleotide sequence of transcript i and c_i denotes the number of copies of t_i in the sample. The length of transcript t_i is denoted by l_i. Under ideal, uniform, sampling conditions (i.e. without considering various types of experimental bias), the probability of drawing a fragment from a transcript t_i is proportional to its nucleotide fraction (Li et al., 2010) denoted by $\eta_i = \frac{c_i l_i}{\sum_{j=1}^{M} c_j l_j}$.

If we normalize the η_i for each transcript by its length l_i, we obtain a measure of the relative abundance of each transcript called the transcript fraction (Li et al., 2010), which is given by $\tau_i = \frac{\frac{\eta_i}{l_i}}{\sum_{j=1} \frac{\eta_i}{l_i}}$.

When performing transcript-level quantification, $\boldsymbol{\eta}$ and $\boldsymbol{\tau}$ are generally the quantities we are interested in inferring. Since they are directly related, knowing one allows us to directly compute the other. Below, we describe our approach to approximating the estimated number of reads originating from each transcript, from which we estimate $\boldsymbol{\tau}$ and, subsequently transcripts per million (TPM).

2.3.2 Quasi-mapping-based Sailfish

Using the quasi-mapping procedure provided by `RapMap` as a library, we have updated the Sailfish (Patro, Mount, and Kingsford, 2014) software to make use of quasi-mapping, as opposed to individual k-mer counting, for transcript-level quantification. In the updated version of Sailfish, the `index` command builds the quasi-index over the reference transcriptome as described in Section 2.1. Given the index and a set of sequenced reads, the `quant` command quasi-maps the reads and uses the resulting mapping information to estimate transcript abundances.

To reduce the memory usage and computational requirements of the inference procedure, quasi-Sailfish reduces the mapping information to a set of equivalence classes over sequenced fragments. These equivalence classes are similar to those used in Nicolae et al. (2011), except that the position of each fragment within a transcript is not considered when defining the equivalence relation. Specifically, any fragments that map to exactly the

same set of transcripts are placed into the same equivalence class. Following the notation of Patro, Duggal, and Kingsford (2015), the equivalence classes are denoted as $\boldsymbol{\mathcal{C}} = \{\mathcal{C}^1, \mathcal{C}^2, \dots\}$, and the count of fragments associated with equivalence class $\mathcal{C}^j$ is given by d^j. Associated with each equivalence class $\mathcal{C}^j$ is an ordered collection of transcript identifiers $\mathbf{t}^j = (t_{j1}, t_{j2}, \dots)$ which is simply the collection of transcripts to which all equivalent fragments in this class map. We call $\mathbf{t}^j$ the *label* of class $\mathcal{C}^j$.

Inferring transcript abundances

The equivalence classes $\boldsymbol{\mathcal{C}}$ and their associated counts and labels are used to estimate the number of fragments originating from each transcript. The estimated count vector is denoted by $\boldsymbol{\alpha}$, and α_i is the estimated number of reads originating from transcript t_i. In quasi-Sailfish, we use the variational Bayesian expectation maximization (VBEM) algorithm to infer the parameters (the estimated number of reads originating from each transcript) that maximize a variational objective. Specifically, we maximize a simplified version of the variational objective of Nariai et al. (2013).

TABLE 2.2: Performance evaluation of different tools along with quasi enabled sailfish (q-Sailfish) with other tools on synthetic data generated by Flux simulator.

	Kallisto	RSEM	q-Sailfish	Tigar 2
Proportionality corr.	0.74	0.78	0.75	0.77
Spearman corr.	0.69	0.73	0.70	0.72
TPEF	0.77	0.96	0.60	0.59
TPME	-0.24	-0.37	-0.10	-0.09
MARD	0.36	0.29	0.31	0.26
wMARD	4.68	5.23	4.45	4.35

TABLE 2.3: Performance evaluation of different tools along with quasi enabled sailfish (q-Sailfish) with other tools on synthetic data generated by RSEM simulator.

	Kallisto	RSEM	q-Sailfish	Tigar 2
Proportionality corr.	0.91	0.93	0.91	0.93
Spearman corr.	0.91	0.93	0.91	0.93
TPEF	0.53	0.49	0.53	0.50
TPME	0.00	-0.01	0.00	0.00
MARD	0.29	0.25	0.29	0.23
wMARD	1.00	0.88	1.01	0.94

The VBEM update rule can be written as a simple iterative update in terms of the equivalence classes, their counts, and the prior (α_0). The iterative update rule for the VBEM is:

$$\alpha_i^{u+1} = \alpha_0 + \sum_{\mathcal{C}^j \in \mathcal{C}} d_j \left(\frac{e^{\gamma_i^u} \frac{1}{\tilde{l}_i}}{\sum_{t_k \in \mathbf{t}^j} e^{\gamma_k^u} \frac{1}{\tilde{l}_k}} \right), \tag{2.1}$$

where

$$\gamma_i^u = \Psi(\alpha_0 + \alpha_i^u) - \Psi(\sum_k \alpha_0 + \alpha_k^u) \tag{2.2}$$

and $\Psi(\cdot)$ is the digamma function. Here, $\hat{l}_i$ is the *effective* length of transcript t_i, computed as in Li et al. (2010). To determine the final estimated counts — $\boldsymbol{\alpha}$ — Equation (2.1) is iterated until convergence. The estimated counts are considered to have converged when no transcript has estimated counts differing by more than one percent between successive iterations.

Given $\boldsymbol{\alpha}$, we compute the TPM for transcript i as

$$TPM_i = 10^6 \frac{\frac{\alpha_i}{\hat{l}_i}}{\sum_j \frac{\alpha_j}{\hat{l}_j}}. \tag{2.3}$$

Sailfish outputs, for each transcript, its name, length, effective length, TPM and the estimated number of reads originating from it.

2.3.3 Quantification performance comparison

We compared the accuracy of quasi-Sailfish (Sailfish v0.9.0; q-Sailfish in Tables 2.2 and 2.3) to the transcript-level quantification tools RSEM (Li et al., 2010) (v1.2.22), Tigar 2 (Nariai et al., 2014) (v2.1), and `Kallisto` (Bray et al., 2015) (v0.42.4) using 6 different accuracy metrics and data from two different simulation pipelines. One of the simulated datasets was generated with the Flux Simulator (Griebel et al., 2012), and is the same dataset used in Section 2.2 to assess mapping accuracy and performance on synthetic data. The other dataset was generated using the RSEM simulator via the same methodology adopted by Bray et al. (2015). That is, RSEM was run on sample `NA12716_7` of the Geuvadis RNA-seq data (Lappalainen et al., 2013) to learn model parameters and estimate true expression. The learned model was then used to generate the simulated dataset, which consists of 30 million 75 bp paired-end reads.

We measure the accuracy of each method based on the estimated versus true number of reads originating from each transcript, and consider 6 different metrics of accuracy; proportionality correlation (Lovell et al., 2015), Spearman correlation, the true positive error fraction (TPEF), the true positive median error (TPME), the mean absolute relative difference (MARD) and the weighted mean absolute relative difference (wMARD). Detailed definitions for the last four metrics are provided in Appendix A.5.

Each of these metrics captures a different notion of accuracy, and all are reported to provide a more comprehensive perspective on quantifier accuracy. The first two metrics — proportionality and Spearman correlation — provide a global notion of how well the estimated and true counts agree, but are fairly coarse measures. The TPEF assesses the fraction of transcripts where the estimate is different from the true count by more than some nominal fraction (here 10%). Unlike TPEF, the TPME metric takes into account the direction of the mis-estimate (i.e. is it an over or under-estimate of the true value?). However, both metrics are assessed only on truly-expressed transcripts, and so provide no insight into the tendency of a quantifier to produce false positives.

The absolute relative difference (ARD) metric has the benefit of being defined on all transcripts as opposed to only those which are truly expressed and ranges from 0 (lowest) to 2 (highest). Since the values of this metric are tightly bounded, the aggregate metric, MARD, is not dominated by high expression transcripts. Unfortunately, it therefore has limited ability to capture the magnitude of mis-estimation. The wMARD metric attempts to account for the magnitude of mis-estimation, while still trying to ensure that the measure is not completely dominated by high expression transcripts. This is done by scaling each ARD_i value by the logarithm of the expression.

Tables 2.2 and 2.3 show the performance of all 4 quantifiers, under all 6 metrics, on both datasets. While all methods seem to perform reasonably well, some patterns emerge. RSEM seems to perform very well in terms of the correlation metrics, but less well in terms of the TPEF, TPME, and wMARD metrics (specifically in the Flux Simulator-generated dataset). This is likely a result of the lower mapping rate obtained on this data by RSEM's very strict `Bowtie 2` parameters. Tigar 2 generally performs very well under a broad range of metrics, and produces highly-accurate results. However, it is *by far* the slowest method considered here, and requires over a day to complete on the Flux simulator data and almost 7 hours to complete on the RSEM-sim data given 16 threads (and not including the time required for `Bowtie 2` alignment of the reads). Finally, both quasi-Sailfish and `Kallisto` perform well in general under multiple different metrics, with quasi-Sailfish tending to produce somewhat more accurate estimates. Both of these methods also completed in a matter of minutes on both datasets.

One additional pattern that emerges is that the RSEM-sim data appears to present a much simpler inference problem compared to the Flux Simulator data. One reason for this may be that the RSEM-sim data is very "clean" — yielding concordant mapping rates well over 99%, even under RSEM's strict `Bowtie 2` mapping parameters. As such, all methods tend to perform well on this data, and there is comparatively little deviation between the methods under most metrics.

For completeness, we also provide (in Appendix A.4) the results, under all of these metrics, where the true and predicted abundances are considered in terms of TPM rather than number of reads. We find that the results are generally similar, with the exception that TIGAR 2 performs considerably worse under the TPM measure.

Chapter 3

Conclusion and Future Work

3.1 Discussion & Conclusion

In this study, we have argued for the usefulness of our novel approach, quasi-mapping, for mapping RNA-seq reads. More generally, we suspect that read *mapping*, wherein sequencing reads are assigned to reference locations, but base-to-base alignments are not computed, is a broadly useful tool. The speed of traditional aligners like `Bowtie 2` and `STAR` is limited by the fact that they must produce optimal alignments for each location to which a read is reported to align.

In addition to showing the speed and accuracy of quasi-mapping directly, we apply it to a problem in transcriptome analysis. we have updated the Sailfish software to make use of the quasi-mapping information produced by `RapMap`, rather than direct k-mer counts, for purposes of transcript-level abundance estimation. This update improves both the speed and accuracy of Sailfish, and also reduces the complexity of its code base. We demonstrate, on synthetic data generated via two different simulators, that the resulting quantification estimates have accuracy comparable to state-of-the-art tools.

However, `RapMap` is a stand-alone mapping program and need not be used only for the applications we describe here. We expect that quasi-mapping will prove a useful and rapid alternative to alignment for tasks ranging from filtering large read sets (e.g. to check for contaminants or the presence or absence specific targets) to more mundane tasks like quality control and, perhaps, even to related tasks like metagenomic and metatranscriptomic classification and abundance estimation. We hope that the quasi-mapping concept, and the availability of `RapMap` and the efficient and accurate mapping algorithms it exposes, will encourage the community to explore replacing alignment with mapping in the numerous scenarios where traditional alignment information is unnecessary for downstream analysis.

3.2 Future Work

The concept of quasi-mapping is fast, accurate and solves an important problem, read-mapping. Since the mapping approach is still new and unexplored, we are reaching out to find other biological applications where `RapMap` can be useful. In the following section, we discuss some of the applications on which we are currently working. However, this is in no way an exhaustive list and we believe `RapMap` has the capability to simplify many more biological analysis.

3.2.1 RapClust (Srivastava et al., 2016)

Estimating gene expression from RNA-seq reads is an especially challenging task when no reference genome is present. Typically, this problem is solved by performing *de novo* assembly of the RNA-seq reads, and subsequently mapping these reads to the resulting contigs to estimate expression. Due to sequencing errors and artifacts, and genetic variation and repeats, *de novo* assemblers often fragment individual isoforms into separately assembled contigs. Davidson and Oshlack (2014) argue that better differential expression results can be obtained in *de novo* assemblies if contigs are first clustered into groups. They present a tool, CORSET, to perform this clustering, and compare their approach to existing tools such as CD-HIT (Fu et al., 2012). CD-HIT compares the sequences (contigs) directly and clusters them by sequence similarity. CORSET, alternatively, aligns reads to contigs (allowing multi-mapping) and defines a distance between each pair of contigs based on the number of multi-mapping reads shared between them, and the changes in estimated expression inferred for these contigs under different conditions. Hierarchical agglomerative clustering is then performed on these distances to obtain a clustering of contigs.

`RapMap` can be used for the same task, by taking an approach similar to that of CORSET. By mapping the RNA-seq reads to the target contigs and simultaneously constructing collapsed classes over fragments we can construct a weighted undirected graph. Given this undirected graph that represents the pair-wise similarity between contigs, we can use the *Markov Cluster Algorithm* (Van Dongen, 2001) to cluster the graph. In fact, `RapMap`-enabled clustering, as discussed in our recent paper (Srivastava et al., 2016), appears to provide comparable or better clusterings than existing methods, and produces these clusterings much more quickly. In RapClust, we presented a fast and accurate methodology for the data-driven clustering of *de novo* transcriptome assemblies. But, there are many interesting directions for future work on this problem, we believe that the quality of the resulting clusters could be improved through a data-driven selection of the appropriate cutoff parameters. Another potential improvement on the current methodology would be to adopt a more robust log fold-change test, that may be more accurate in separating contigs that do not originate from the same gene. Sailfish is capable of producing not only transcript-level abundances but also estimates of the variance of each predicted abundance via posterior Gibbs sampling or bootstraps. This variance information can be incorporated into the estimates of log fold-change differences to allow for increased precision in separating potential paralogs. While the existing method works well in the completely *de novo* context (i.e. even when the genomes or transcriptomes of closely related organisms may not be available), integrating homology information, when available, has the potential to improve the clustering results (and provide meaningful biological annotations for the clusters). The best way to integrate this information is an exciting direction for future work.

3.2.2 RapAlign

As discussed in Section 1.5 by relaxing the problem of read-alignment to read-mapping we were able to devise fast methods like quasi-mapping.

But, `RapMap` has the capability to be developed as full read aligner. We are working on designing a method to retrieve base-to-base alignments from the mappings obtained by `RapMap`. Specifically, we believe that multi-mapping alignments can be computed from the mappings at only a marginal extra cost, given RapMap's knowledge about similarities among the reference sequence being mapped to. Additionally, we are working on a compressed index to be used with `RapMap` to work around the problem of the big index so that `RapMap` can be used with very large reference sequences (e.g. thousands of genomes in a metagenomic/metatranscriptomic context).

Appendix A

Appendix

A.1 Parameters for mapping and alignment tools

When `Bowtie 2` was run to produce alignment results, it was run with default parameters with the exception of `-k 200` and `-no-discordant`. When timing `Bowtie 2` the the number of threads (`-p`) was set in accordance with what is mentioned in the relevant text, and the output was piped to `/dev/null`. When `Bowtie 2` was used to produce alignment results for quantification with `RSEM`, `RSEM`'s `Bowtie 2` wrapper (with its default parameters) was used to generate alignments.

When producing alignment results, `STAR` was run with the following parameters: `-outFilterMultimapNmax 200 -outFilterMismatchNmax 99999 -outFilterMismatchNoverLmax 0.2 -alignIntronMin 1000 -alignIntronMax 0 -limitOutSAMoneReadBytes 1000000 -outSAMmode SAMUnosrted`. Additionally, when timing `STAR`, it was run with the number of threads (`-runThreadN`) specified in the relevant text and with the `-outSAMMode None` flag.

To obtain the "pseudo-alignments" produces by `Kallisto`, it was run with the `-pseudobam` flag.

When producing mapping results, `RapMap` was run with the option `-m 200` to limit multi-mapping reads to 200 locations. Additionally, when timing `RapMap`, it was run with the number of threads (`-t`) specified in the relevant text and with the `-n` flag to suppress output.

A.2 Flux Simulator parameters

The Flux simulator dataset was generated using the following parameters:

```
REF_FILE_NAME   Human_Genome
GEN_DIR     protein_coding.gtf

NB_MOLECULES    5000000
TSS_MEAN    50
POLYA_SCALE NaN
POLYA_SHAPE NaN

FRAG_SUBSTRATE  RNA
FRAG_METHOD UR
FRAG_UR_ETA     350

RTRANSCRIPTION  YES
```

```
RT_MOTIF default

GC_MEAN NaN
PCR_PROBABILITY 0.05
PCR_DISTRIBUTION default

FILTERING YES

READ_NUMBER 150000000
READ_LENGTH 76
PAIRED_END  YES
ERR_FILE    76
FASTA       YES
```

The following parameters were used to produce noise reads:

```
PAIRED_END YES
REF_FILE_NAME noisy.gtf

READ_LENGTH 76
PRO_FILE_NAME flux_simulator_noise_expression.pro
ERR_FILE 76
GEN_DIR Human_Genome/
SEQ_FILE_NAME noise_reads.bed
PCR_DISTRIBUTION none
POLYA_SCALE NaN
FASTA YES

NB_MOLECULES 2000000
READ_NUMBER 34382441
UNIQUE_IDS YES
POLYA_SHAPE NaN
```

A.3 Mapping accuracy in the presence of noisy reads

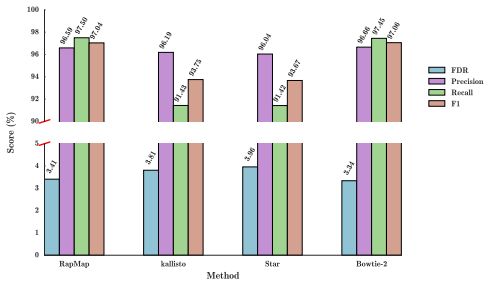

FIGURE A.1: Precision, recall and F1-score (top) and FDR (bottom) on the simulated dataset with noise, for the 4 different tools we consider.

We tested the effect of including background (i.e. noise) reads on the accuracy of the different mapping and alignment tools. In this experiment, we sampled 9 million reads from the 48 million read simulated data set used

in Section 2.2.1. We then incorporated an additional 1 million "noise" reads from a simulated dataset generated with the Flux Simulator using a custom annotation. This noise annotation was created by constructing a single interval for each transcript, which contained the entire genomic range from the initial until the terminal exons (i.e. it contained all intervening intronic regions). Thus, for each annotated transcript, the noise annotation contains a nascent, un-spliced version of this transcript. This model of noise was motivated from the observation of (Gilbert et al., 2004), that some RNA-seq data (e.g. human brain tissue) contains reads potentially derived from nascent, un-spliced variants of expressed transcripts.

As shown in Supplementary Figure 1 we observe that, in the presence of noise, the precision for all the tools decreases slightly compared to the "clean", 48 million read dataset described in Section 2.2.1. This is because some small fraction of noisy reads are assigned as false positives, as they map to the mature version of their corresponding transcript of origin that appears in the reference. Overall, however, the results follow a very similar trend both with and without noisy reads. Specifically, `RapMap` (quasi-mapping) performs almost identically to `Bowtie 2`, while `Kallisto` and `STAR` yield very similar results — somewhat under-performing `RapMap` and `Bowtie 2`. This clearly demonstrates that, in the presence of noisy reads, all of the tools degrade gracefully and still perform reasonably well, with no discernible difference between mapping and alignment-based tools.

A.4 Quantification results using TPM

In addition to computing the error metrics based on the estimated versus true number of reads originating from each transcript (as provided in **??**), we also evaluated the same metrics based instead on the TPM of each transcript. That is, all of the metrics defined in Section 2.3.3 and appendix A.5 remain the same, except that x_i now denotes the true TPM value for transcript i and y_i denotes the estimated TPM of transcript i. We note that the Flux Simulator provides neither effective lengths nor TPM estimates directly. To obtain the ground truth TPM values for the Flux Simulator dataset, we first computed the effective length of each transcript (by convolving the characteristic function over the transcript with the *true* fragment length distribution), and then computed the TPM value for each transcript using Equation (2.3). The results are generally similar to what was observed at the read level, except that TIGAR 2 seems to perform considerably worse under a number of metrics on the RSEM-sim dataset when considering the TPM measure of abundance.

A.5 Error Metrics

We define the error metrics reported in Section 2.3.3 below, letting x_i denote the true number of reads originating from transcript i and y_i denote the estimated number of reads.

TABLE A.1: Performance evaluation of different tools along with quasi enabled sailfish (q-Sailfish) with other tools on synthetic data generated by Flux simulator.

	Kallisto	RSEM	q-Sailfish	Tigar 2
Proportionality corr.	0.79	0.80	0.80	0.80
Spearman corr.	0.69	0.73	0.71	0.60
TPEF	0.87	0.88	0.84	0.94
TPME	0.07	0.13	0.12	-0.40
MARD	0.35	0.27	0.31	0.35
wMARD	0.67	1.22	0.69	1.76

TABLE A.2: Performance evaluation of different tools along with quasi enabled sailfish (q-Sailfish) with other tools on synthetic data generated by RSEM simulator.

	Kallisto	RSEM	q-Sailfish	Tigar 2
Proportionality corr.	0.94	0.96	0.94	0.93
Spearman corr.	0.91	0.93	0.91	0.89
TPEF	0.51	0.47	0.50	0.95
TPME	0.00	0.00	0.00	0.21
MARD	0.28	0.25	0.28	0.48
wMARD	-0.74	-0.73	-0.74	0.12

The relative error for transcript i (RE_i) is given by $\mathrm{RE_i} = \frac{x_i - y_i}{x_i}$ and the error indicator for transcript i (EI_i) is given by

$$\mathrm{EI_i} = \begin{cases} 1 & \text{if } |RE_i| > 0.1 \\ 0 & \text{otherwise} \end{cases}, \tag{A.1}$$

and it is equal to 1 if the estimated count for this truly expressed transcript (it is undefined, as is RE_i, when $x_i = 0$) differs from the true count by more than 10%. Given RE_i and EI_i, the aggregate true positive error fraction (TPEF) is defined as $\mathrm{TPEF} = \frac{1}{|X^+|}\sum_{i \in X^+} EI_i$. Here, X^+ is the set of "truly expressed" transcripts (those having at least 1 read originating from them in the ground truth). Similarly, the true positive median error is define as $\mathrm{TPME} = \mathrm{median}\left(\{\mathrm{RE}_i\}_{i \in X^+}\right)$. Finally, the absolute relative difference for transcript i (ARD_i) is defined as

$$\mathrm{ARD}_i = \begin{cases} 0 & \text{if } x_i + y_i = 0 \\ \frac{|x_i - y_i|}{0.5(x_i + y_i)} & \text{otherwise} \end{cases}. \tag{A.2}$$

Consequently, the mean absolute relative difference (MARD) is defined as $\mathrm{MARD} = \frac{1}{M}\sum_i \mathrm{ARD}_i$, and the weighted mean absolute relative difference (wMARD) is defined as

$$\mathrm{wMARD} = \sum_{i \in \mathrm{ARD}^+} \frac{\log\left(\max\left(x_i, y_i\right)\right) \mathrm{ARD}_i}{M}, \tag{A.3}$$

where, $\mathrm{ARD}^+ = \{i | ARD_i > 0\}$, and M is the total number of transcripts.

Bibliography

Alignment vs. Mapping. http://robpatro.com/blog/?p=260. Accessed: 2015-08-15.

Alternative Splicing. https://en.wikipedia.org/wiki/Alternative_splicing.

Altschul, Stephen F et al. (1990). "Basic local alignment search tool". In: *Journal of molecular biology* 215.3, pp. 403–410.

Altschul, Stephen F et al. (1997). "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs". In: *Nucleic acids research* 25.17, pp. 3389–3402.

Bray, Nicolas et al. (2015). "Near-optimal RNA-Seq quantification". In: *arXiv preprint arXiv:1505.02710*.

Burrows, Michael and David Wheeler (1994). "A block-sorting lossless data compression algorithm". In: *DIGITAL SRC RESEARCH REPORT*. Citeseer.

Cho, H. et al. (2014). "High-resolution transcriptome analysis with long-read RNA sequencing". In: *PLoS ONE* 9.9, e108095.

Cunningham, Fiona et al. (2015). "Ensembl 2015". In: *Nucleic acids research* 43.D1, pp. D662–D669.

Davidson, Nadia M and Alicia Oshlack (2014). "Corset: enabling differential gene expression analysis for de novo assembled transcriptomes". In: *Genome biology* 15.7, p. 410.

Dobin, Alexander et al. (2013). "STAR: ultrafast universal RNA-seq aligner". In: *Bioinformatics* 29.1, pp. 15–21.

Ewing, Brent and Phil Green (1998). "Base-calling of automated sequencer traces using phred. II. Error probabilities". In: *Genome research* 8.3, pp. 186–194.

Ferragina, Paolo and Giovanni Manzini (2000). "Opportunistic data structures with applications". In: *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*. IEEE, pp. 390–398.

— (2001). "An experimental study of an opportunistic index". In: *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, pp. 269–278.

Fu, Limin et al. (2012). "CD-HIT: accelerated for clustering the next-generation sequencing data". In: *Bioinformatics* 28.23, pp. 3150–3152.

Gilbert, Christopher et al. (2004). "Elongator interactions with nascent mRNA revealed by RNA immunoprecipitation". In: *Molecular cell* 14.4, pp. 457–464.

Griebel, Thasso et al. (2012). "Modelling and simulating generic RNA-Seq experiments with the flux simulator". In: *Nucleic acids research* 40.20, pp. 10073–10083.

Gusfield, Dan (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. New York, NY, USA: Cambridge University Press. ISBN: 0-521-58519-8.

Hach, Faraz et al. (2010). "mrsFAST: a cache-oblivious algorithm for short-read mapping". In: *Nature methods* 7.8, pp. 576–577.

Ilie, Lucian, Gonzalo Navarro, and Liviu Tinta (2010). "The longest common extension problem revisited and applications to approximate string searching". In: *Journal of Discrete Algorithms* 8.4, pp. 418–428.

Kim, Daehwan, Ben Langmead, and Steven L Salzberg (2015). "HISAT: a fast spliced aligner with low memory requirements". In: *Nature methods* 12.4, pp. 357–360.

Köster, Johannes and Sven Rahmann (2012). "Building and Documenting Workflows with Python-Based Snakemake." In: *GCB*, pp. 49–56.

Langmead, Ben and Steven L Salzberg (2012). "Fast gapped-read alignment with Bowtie 2". In: *Nature methods* 9.4, pp. 357–359.

Langmead, Ben et al. (2009). "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome". In: *Genome Biology* 10.3, R25.

Lappalainen, Tuuli et al. (2013). "Transcriptome and genome sequencing uncovers functional variation in humans". In: *Nature*.

Li, Bo and Colin N Dewey (2011). "RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome". In: *BMC bioinformatics* 12.1, p. 323.

Li, Bo et al. (2010). "RNA-Seq gene expression estimation with read mapping uncertainty". In: *Bioinformatics* 26.4, pp. 493–500.

Li, Heng (2013). *Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM*.

Li, Heng and Richard Durbin (2009). "Fast and accurate short read alignment with Burrows–Wheeler transform". In: *Bioinformatics* 25.14, pp. 1754–1760.

Li, Heng and Nils Homer (2010). "A survey of sequence alignment algorithms for next-generation sequencing". In: *Briefings in bioinformatics* 11.5, pp. 473–483.

Li, Heng, Jue Ruan, and Richard Durbin (2008). "Mapping short DNA sequencing reads and calling variants using mapping quality scores". In: *Genome research* 18.11, pp. 1851–1858.

Li, Ruiqiang et al. (2008). "SOAP: short oligonucleotide alignment program". In: *Bioinformatics* 24.5, pp. 713–714.

Li, Ruiqiang et al. (2009). "SOAP2: an improved ultrafast tool for short read alignment". In: *Bioinformatics* 25.15, pp. 1966–1967.

Liao, Yang, Gordon K Smyth, and Wei Shi (2013). "The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote". In: *Nucleic acids research* 41.10, e108–e108.

Lovell, David et al. (2015). "Proportionality: a valid alternative to correlation for relative data". In: *PLoS computational biology* 11.3, e1004075.

Manber, Udi and Gene Myers (1993). "Suffix arrays: a new method for on-line string searches". In: *siam Journal on Computing* 22.5, pp. 935–948.

Marçais, Guillaume and Carl Kingsford (2011). "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers". In: *Bioinformatics* 27.6, pp. 764–770.

Nariai, Naoki et al. (2013). "TIGAR: transcript isoform abundance estimation method with gapped alignment of RNA-Seq data by variational Bayesian inference". In: *Bioinformatics* 29, btt381.

Nariai, Naoki et al. (2014). "TIGAR2: sensitive and accurate estimation of transcript isoform expression with longer RNA-Seq reads". In: *BMC genomics* 15.Suppl 10, S5.

Nicolae, M. et al. (2011). "Estimation of alternative splicing isoform frequencies from RNA-Seq data". In: *Algorithms for Molecular Biology* 6:9.

Patro, Rob, Geet Duggal, and Carl Kingsford (2015). "Salmon: Accurate, Versatile and Ultrafast Quantification from RNA-seq Data using Lightweight-Alignment". In: *bioRxiv* 9, p. 021592.

Patro, Rob, Stephen M Mount, and Carl Kingsford (2014). "Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms". In: *Nature biotechnology* 32.5, pp. 462–464.

Reis-Filho, Jorge S et al. (2009). "Next-generation sequencing". In: *Breast Cancer Res* 11.Suppl 3, S12.

Sanger, Fred and Alan R Coulson (1975). "A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase". In: *Journal of molecular biology* 94.3, pp. 441–448.

Schadt, Eric E, Steve Turner, and Andrew Kasarskis (2010). "A window into third-generation sequencing". In: *Human molecular genetics*, ddq416.

Schatz, Michael C, Arthur L Delcher, and Steven L Salzberg (2010). "Assembly of large genomes using second-generation sequencing". In: *Genome research* 20.9, pp. 1165–1173.

Smith, Temple F and Michael S Waterman (1981). "Identification of common molecular subsequences". In: *Journal of molecular biology* 147.1, pp. 195–197.

Srivastava, Avi et al. (2016). "Accurate, Fast and Lightweight Clustering of de novo Transcriptomes using Fragment Equivalence Classes". In: *arXiv preprint arXiv:1604.03250*.

Trapnell, Cole, Lior Pachter, and Steven L Salzberg (2009). "TopHat: discovering splice junctions with RNA-Seq". In: *Bioinformatics* 25.9, pp. 1105–1111.

Van Dongen, Stijn Marinus (2001). "Graph clustering by flow simulation". In:

Venter, J Craig et al. (2001). "The sequence of the human genome". In: *science* 291.5507, pp. 1304–1351.

Vigna, Sebastiano (2008). "Broadword implementation of rank/select queries". In: *Experimental Algorithms*. Springer, pp. 154–168.

Wang, Zhong, Mark Gerstein, and Michael Snyder (2009). "RNA-Seq: a revolutionary tool for transcriptomics". In: *Nature Reviews Genetics* 10.1, pp. 57–63.

Zhang, Zhaojun and Wei Wang (2014). "RNA-Skim: a rapid method for RNA-Seq quantification at transcript level". In: *Bioinformatics* 30.12, pp. i283–i292.