Abstract

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.

Results: We introduce a novel algorithm, quasi-mapping, 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 — the tool implementing this quasi-mapping algorithm — is capable of mapping sequencing reads to a target transcriptome substantially faster than existing alignment tools. The quasi-mapping algorithm itself 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.

Availability: RapMap is implemented in C++ and is available as open-source software, under GPL v3, at https://github.com/COMBINE-lab/RapMap

1 Introduction

The bioinformatics community has put tremendous effort into building a wide array of different tools to solve the read-alignment problem efficiently. These tools use many different strategies to quickly find potential alignment locations for reads; for example, Bowtie (Langmead et al. 2009), Bowtie2 (Langmead and Salzberg 2012), BWA (Li and Durbin 2009) and BWA-mem (Li 2013) use variants of the FM-index, while tools like the Subread aligner (Liao et al. 2013), Maq (Li et al. 2008) and MrsFast (Hach et al. 2010) use k-mer-based indices to help align reads efficiently. Because read alignment is such a ubiquitous task, the goal of such tools is often to provide accurate results as quickly as possible. Indeed, recent alignment tools like STAR (Dobin et al. 2013) demonstrate that rapid alignment of sequenced reads is possible, and tools like HISAT (Kim et al. 2015) demonstrate that this speed can be achieved with only moderate memory usage. 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 being mapped against a transcriptome, or when attempting to cluster de novo assembled contigs by shared multi-mapping reads (Davidson and Oshlack 2014)). However, reporting multiple potential alignments for a single 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 an-
swer the questions being posed. In support of such “analysis-oriented” computation, we propose a novel algorithm, called quasi-mapping, implemented in the 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 et al. 2014] tool to take advantage of quasi-mappings, as opposed to raw k-mer counts, for transcript quantification.

2 Methods

The quasi-mapping algorithm, implemented in the tool RapMap, is a new mapping technique to allow the rapid and accurate mapping of “short” fragments to a target transcriptome. The quasi-mapping algorithm 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 upon which the suffix array was constructed, and the original text 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.

Quasi-mapping determines the mapping locations for a query read $r$ through repeated application of (1) determining the next hashable 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 $I(k_i) = [b, e)$. 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 accelerator” 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 search to this region of the suffix array, which is typically very small.

After determining the length of MMP 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, the suffixes occurring at the lower and upper bounds of the suffix array interval corresponding to MMP may not differ from each other (See Figure 1). That is, if $I(MMP_i) = [b', e')$ is the suffix array interval corresponding to MMP, it is possible that $|\text{LCP}(T[SA[b']], T[SA[e'-1]])| > |\text{MMP}_i|$. In this case, it is most likely that the read and the reference sequence bases following MMP 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 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, with $I(MMP_i) = [b', e')$, we define $\text{NIP}(MMP_i) = |\text{LCP}(T[SA[b']], T[SA[e'-1]])| + 1$. Intuitively, the next informative position of prefix MMP 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 $I(MMP_i)$. To compute the longest common prefix between two suffixes when searching for the NIP, we use the “direct min” algorithm of Ilie et al. (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 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 $I(k_i)$ corresponding to the substring of the read consisting of
this k-mer. Then, the procedure described above is used to compute \( \text{MMP}_i \) and \( \ell = \text{NIP}(\text{MMP}_i) \). 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 set of transcripts that appear 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. In the next section, we analyze how the quasi-mapping algorithm described above compares to other aligners in terms of speed and mapping accuracy.

3 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 Bowtie2 (Langmead and Salzberg 2012) and STAR (Dobin et al. 2013) and the recently-introduced pseudo-alignment procedure used in the quantification tool Kallisto (Bray et al. 2015). All experiments were performed on a 64-bit linux server with 256GB of RAM and 4 x 6-core Intel Xeon E5-4607 v2 CPUs (with hyper-threading) 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. Though, for reasons stated below, we don’t benchmark the runtime of pseudo-alignment directly, we find that quasi-mapping and pseudo-alignment are similar in terms of speed. In fact, for both of these methods, simply writing the output to disk tends to dominate the time required for large input files with significant multi-mapping. 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.

3.1 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 database consisting of 86,090 transcripts corresponding to protein-coding genes. The dataset consists of \( \sim 48 \) 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 \textit{de novo} transcriptomics). The parameters of STAR (see Appendix A.1) were adjusted appropriately for this purpose (e.g. to disallow introns etc.). Similarly, \texttt{Bowtie 2} was also used to align reads directly to the target transcriptome; the parameters for \texttt{Bowtie 2} are given in Appendix A.1.

### 3.1.1 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. Finally, we choose not to include \texttt{Kallisto} in the timing benchmarks for three reasons. First, unlike all other methods tested here, it is not multi-threaded. Second, it does not provide a stand alone pseudo-aligner, and so the recorded time would also include the time required for transcript-level abundance estimation. Finally, we cannot account for “outlier” reads since it does not provide an option to limit the number of multi-mapping locations.

As Figure 2 illustrates, \texttt{RapMap} out-performs both \texttt{Bowtie 2} and \texttt{STAR} in terms of speed by a substantial margin, and finishes mapping the reads with a single thread faster than \texttt{STAR} and \texttt{Bowtie 2} with 10 threads. We consider varying the number of threads used by \texttt{RapMap} and \texttt{STAR} to demonstrate how performance scales with the number of threads provided. On this data set, \texttt{RapMap} quickly approaches peak performance after using only a few threads. We believe that this is not due to limits on the scalability of \texttt{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 \texttt{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 \cite{marc2011library} library) can provide input to \texttt{RapMap} quickly enough to make use of a larger number of threads. Since running \texttt{Bowtie 2} with each potential number of threads on this dataset is very time-consuming, we only consider \texttt{Bowtie 2}’s runtime using 10 threads.

### 3.1.2 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 (since each simulated read actually comes from some reference transcript). Given these definitions, we report precision, recall, F1-Score and false discovery rate (FDR) in Table I 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, \texttt{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 \texttt{RapMap} yields very similar (in fact, slightly better) precision and recall. \texttt{STAR} and \texttt{Kallisto} obtain similar precision to \texttt{Bowtie 2} and \texttt{RapMap}, but have lower recall. \texttt{STAR} and \texttt{Kallisto} perform similarly in general, though \texttt{Kallisto} achieves a lower (better) FDR than \texttt{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 \texttt{RapMap} yields very accurate mapping results that match or exceed those of \texttt{Bowtie 2}.
3.2 Speed and concordance on experimental data

We also explore the concordance of quasi-mapping with different mapping and alignment approaches using experimental data from the study of Cho et al. (2014). These sequencing reads are derived from human lymphoblastoid cell line GM12878 (NCBI GEO accession SRR1293902). The sample consists of ~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 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, \( U \), to be sets 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 \( e_{Ai} = \{r_i, t_1, t_2, t_3\} \in U \). Similarly, if tool \( B \) maps read \( r_i \) to transcripts \( \{t_2, t_3, t_4\} \) then \( e_{Bi} = \{r_i, t_2, t_3, t_4\} \in U \). Given a universe \( U \), 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. Using these definitions, we have computed the sizes of the intersections of the results of all tools Figure 3.

We have defined the intersections as is commonly done when creating Venn plots (though we have chosen not to use a proportional Venn plot here since it is, in general, not possible with > 3 sets). For example, the set \( B \) corresponds only to those elements returned by Bowtie 2 that do not appear in some other intersection (i.e. that are not in the set returned by any other method). Thus, the bars corresponding to each individual method represent the isolated elements where a method returns mapping results that do not agree with any other method.

Under this measure of agreement, quasi-mapping and Kallisto appear to agree on the exact same transcript assignments for the largest number of reads. Further, quasi-mapping 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 the main reason that 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 actually only attempt to return multi-mapping locations that are equivalently “best” (STAR seems to do this fairly often as well). 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

<table>
<thead>
<tr>
<th>Aligner</th>
<th>Bowtie 2</th>
<th>Kallisto</th>
<th>RapMap</th>
<th>STAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>align</td>
<td>47579567</td>
<td>44774502</td>
<td>47677356</td>
<td>44711604</td>
</tr>
<tr>
<td>recall</td>
<td>97.41</td>
<td>91.53</td>
<td>97.62</td>
<td>91.35</td>
</tr>
<tr>
<td>precision</td>
<td>98.31</td>
<td>97.67</td>
<td>98.35</td>
<td>97.02</td>
</tr>
<tr>
<td>F1-score</td>
<td>97.86</td>
<td>94.50</td>
<td>97.98</td>
<td>94.10</td>
</tr>
<tr>
<td>FDR</td>
<td>1.69</td>
<td>2.33</td>
<td>1.65</td>
<td>2.98</td>
</tr>
<tr>
<td>hits per read</td>
<td>5.98</td>
<td>5.30</td>
<td>4.30</td>
<td>3.80</td>
</tr>
</tbody>
</table>

Table 1: Accuracy of aligners/mappers under different metrics

Figure 3: Read agreement between Bowtie 2, STAR, Kallisto and RapMap.
of hits per read is higher than for the other methods (see Table 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 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.

4 Application of quasi mapping to RNA-seq 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 et al., 2014; Zhang and Wang, 2014; Bray et al., 2015; Patro et al., 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 et al., 2014) software, which we refer to as quasi-Sailfish. 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.

4.1 Transcript quantification

2 In an RNA-seq experiment, the underlying transcriptome consists of $M$ transcripts and their respective counts. The transcriptome can be represented as a set $X = \{(t_1, \ldots, t_M), (c_1, \ldots, c_M)\}$, 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 any position of transcript $t_i$ is proportional to its nucleotide fraction $\eta_i = \frac{c_i l_i}{\sum_{j=1}^{M} c_j l_j}$. 

If we normalize the $\eta_i$ 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

$$\tau_i = \frac{\eta_i}{\sum_{j=1}^{M} \eta_j}.$$ 

(2)

When performing transcript-level quantification, $\eta$ and $\tau$ 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 $\tau$ and the immediately related metric of transcripts per million (TPM).

4.2 Quasi-mapping-based Sailfish

Using the quasi-mapping procedure provided by RapMap as a library, we have updated the Sailfish (Patro et al., 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. 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 et al. (2015), equivalence classes are denoted as $C = \{C^1, C^2, \ldots\}$, and the count of fragments associated with equivalence class $C^j$ is given by $d^j$. Associated with each equivalence class $C^j$ is an ordered collection of transcript identifiers $t^j = (t_{j1}, t_{j2}, \ldots)$ which is simply the collection of transcripts to which all equivalent fragments in this class map. We call $t^j$ the label of class $C^j$. 

\[\text{denoted by} \quad \eta_i = \frac{c_i l_i}{\sum_{j=1}^{M} c_j l_j}.\] 

(1)
4.2.1 Inferring transcript abundances

The equivalence classes \( \mathcal{C} \) and their associated counts are used to estimate the number of fragments originating from each transcript. The estimated count vector is denoted by \( \alpha \), and \( \alpha_i \) 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)].

The VBEM update rule can be written as a simple iterative update in terms of the equivalence classes, their counts, and the prior \( (\alpha_0) \). The iterative update rule for the VBEM is:

\[
\alpha_{i}^{u+1} = \alpha_0 + \sum_{i' \in \mathcal{C}} d_j \left( \frac{e^{\gamma_{i}^u \frac{1}{l_i}}}{\sum_{k \in \mathcal{U}} e^{\gamma_{k}^u \frac{1}{l_k}}} \right),
\]

where

\[
\gamma_{i}^u = \Psi(\alpha_0 + \alpha_{i}^u) - \Psi(\sum_k \alpha_0 + \alpha_k^u)
\]

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 — \( \hat{\alpha} \) — Equation (3) 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 \( \hat{\alpha} \), we compute the TPM for transcript \( i \) as

\[
TPM_i = 10^6 \frac{\alpha_i}{\sum_j \frac{\alpha_j}{\hat{l}_j}}.
\]

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

4.3 Performance comparison with other quantification tools

We compared the accuracy of quasi-Sailfish (q-Sailfish in Table 2) to the transcript-level quantification tools RSEM ([Li et al. 2010]), Tigar 2 ([Nariai et al. 2014]), and Kallisto ([Bray et al. 2015]) using 6 different accuracy metrics and data from two different simulation pipelines. One of the simulated datasets was generated with the Flux Simulator ([Griibel et al. 2012]), and is the same dataset used in Section 3 to assess mapping accuracy and performance on synthetic data. The other dataset was generated using the RSEM-sim 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 we 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). We define the latter four metrics below, letting \( x_i \) denote the true number of reads originating from transcript \( i \) and \( y_i \) denote the estimated number of reads.

The relative error for transcript \( i \) (RE\(_i\)) is given by

\[
RE_i = \frac{x_i - y_i}{x_i}.
\]

The error indicator for transcript \( i \) (EI\(_i\)) is given by

\[
EI_i = \begin{cases} 
1 & \text{if } |RE_i| > 0.1 \\
0 & \text{otherwise} 
\end{cases}
\]

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 Equations (6) and (7), the aggregate true positive error fraction (TPEF) is defined as

\[
TPEF = \frac{1}{|X^+|} \sum_{i \in X^+} EI_i.
\]

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

\[
TPME = \text{median} \left( \{RE_i\}_{i \in X^+} \right).
\]

Finally, the absolute relative difference for transcript \( i \) (ARD\(_i\)) is defined as

\[
ARD_i = \begin{cases} 
0 & \text{if } x_i + y_i = 0 \\
\frac{|x_i - y_i|}{0.5(x_i + y_i)} & \text{otherwise} 
\end{cases}
\]
Table 2: Performance evaluation of different tools along with quasi enabled sailfish (q-Sailfish) with other tools on synthetic data generated by Flux simulator and RSEM simulator

<table>
<thead>
<tr>
<th></th>
<th>Flux simulation</th>
<th>RSEM-sim simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Kallisto</td>
<td>RSEM</td>
</tr>
<tr>
<td>Proportionality corr.</td>
<td>0.74</td>
<td>0.78</td>
</tr>
<tr>
<td>Spearman corr.</td>
<td>0.69</td>
<td>0.73</td>
</tr>
<tr>
<td>TPEF</td>
<td>0.77</td>
<td>0.96</td>
</tr>
<tr>
<td>TPME</td>
<td>-0.24</td>
<td>-0.37</td>
</tr>
<tr>
<td>MARD</td>
<td>0.36</td>
<td>0.29</td>
</tr>
<tr>
<td>wMARD</td>
<td>4.68</td>
<td>5.23</td>
</tr>
</tbody>
</table>

Consequently, the mean absolute relative difference (MARD) is defined as

$$MARD = \frac{1}{M} \sum_i |ARD_i|$$ (11)

and the weighted mean absolute relative difference (wMARD) is defined as

$$wMARD = \sum_{i \in ARD^+} \frac{\log (\max (x_i, y_i))}{M} |ARD_i|$$ (12)

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

Each of these metrics captures a different notion of accuracy, and so we consider many different metrics 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 true positive error fraction (TPEF) assesses the fraction of transcripts where the estimate is different from the true count by more than some nominal fraction (in this case 10%). Unlike TPEF, the TPME metric takes into account the direction of the mis-estimate (i.e. is the estimate an over or under-estimate of the true value?). However, both of these metrics are assessed only on truly-expressed transcripts, and so provide no insight into the tendency of a quantifier to produce false positive estimates.

The absolute relative difference (ARD) metric has the benefit of being defined on all transcripts as opposed to only those which are truly expressed. The possible value of the ARD ranges from 0 to 2, where 0 represents perfect agreement between the true and predicted values and 2 is the maximum possible difference. Since the values of this metric are tightly bounded, the aggregate metric, MARD, is not dominated by high expression transcripts. Unfortunately, for this reason, it has limited ability to capture the magnitude of mis-estimation. Finally, 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 ARDi value by the logarithm of the expression.

Table 2 shows the performance of all 4 quantifiers, under all 6 metrics, on both datasets we consider. 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 by RSEM’s very strict Bowtie 2 parameters on this data. 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 Bowtie 2 alignment time). 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 meth-
ods tend to perform well on this data, and there is comparatively little deviation between the methods under most metrics.

5 Discussion & Conclusion

In this paper we have argued for the usefulness of our novel algorithm, 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.

One area in which recent work has shown that traditional alignments are not necessary to produce accurate results is transcript-level abundance estimation. To apply quasi-mapping in this context, 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 codebase. 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 transcript quantification. We expect that mapping will prove a useful and rapid alternative to alignment for tasks ranging from clustering de novo assembled transcripts (as in Davidson and Oshlack (2014)) to filtering large read sets (e.g. to check for contaminants or the presence or absence specific targets) to more mundane tasks like quality control (e.g. ensuring that a sufficient fraction of reads map to the reference from which they are derived) and, potentially, even to related tasks like metagenomic and metatranscriptomic classification and abundance estimation.

In addition to the quasi-mapping procedure described in this paper, RapMap also exposes an independent, multi-threaded re-implementation of the concept of pseudo-alignment, as originally introduced by Bray et al. (2015) in the Kallisto software. We hope that 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.

Acknowledgement

The authors would like to thank Geet Duggal and Richard Smith-Unna for useful discussions regarding various aspects of this work.

References


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 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 999999 --outFilterMismatchNoverLmax 0.2 --alignIntronMin 1000 --alignIntronMax 0 --limitOutSAMoneReadBytes 1000000
```

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:

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REF_FILE_NAME</td>
<td>Human_Genome</td>
</tr>
<tr>
<td>GEN_DIR</td>
<td>protein_coding.gtf</td>
</tr>
<tr>
<td>NB_MOLECULES</td>
<td>5000000</td>
</tr>
<tr>
<td>TSS_MEAN</td>
<td>50</td>
</tr>
<tr>
<td>POLYA_SCALE</td>
<td>NaN</td>
</tr>
<tr>
<td>POLYA SHAPE</td>
<td>NaN</td>
</tr>
<tr>
<td>FRAG_SUBSTRATE</td>
<td>RNA</td>
</tr>
<tr>
<td>FRAG_METHOD</td>
<td>UR</td>
</tr>
<tr>
<td>FRAG_UR_ETA</td>
<td>350</td>
</tr>
<tr>
<td>RTRANSCRIPTION</td>
<td>YES</td>
</tr>
<tr>
<td>RT_MOTIF</td>
<td>default</td>
</tr>
<tr>
<td>GC_MEAN</td>
<td>NaN</td>
</tr>
<tr>
<td>PCR_PROBABILITY</td>
<td>0.05</td>
</tr>
<tr>
<td>PCR_DISTRIBUTION</td>
<td>default</td>
</tr>
</tbody>
</table>
```

FILTERING YES

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_NUMBER</td>
<td>1500000000</td>
</tr>
<tr>
<td>READ_LENGTH</td>
<td>76</td>
</tr>
<tr>
<td>PAIRED_END</td>
<td>YES</td>
</tr>
<tr>
<td>ERR_FILE</td>
<td>76</td>
</tr>
<tr>
<td>FASTA</td>
<td>YES</td>
</tr>
</tbody>
</table>
```