RapMap: A Rapid, Sensitive and Accurate Tool for Mapping RNA-seq Reads to Transcriptomes

Avi Srivastava[1], Hirak Sarkar[1], Rob Patro[1,*]

[1] Department of Computer Science, Stony Brook University Stony Brook, NY 11794-2424
* To whom correspondence should be addressed, rob.patro@cs.stonybrook.edu

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. We demonstrate how quasi-mapping can be successfully applied to the problems of transcript-level quantification from RNA-seq reads and the clustering of contigs from *de novo* assembled transcriptomes into biologically-meaningful groups.

Availability: `RapMap` is implemented in `C++11` 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 [15], `Bowtie 2` [14], BWA [20] and BWA-mem [19] use variants of the FM-index, while tools like the Subread aligner [22], Maq [21] and MrsFast [10] 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` [5] demonstrate that rapid alignment of sequenced reads is possible, and tools like HISAT [13] 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 subsequence (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 [17], or when attempting to cluster *de novo* assembled contigs by shared multi-mapping reads [4]). 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 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 [30] tool to take advantage of quasi-mappings, as opposed to raw k-mer counts, for transcript quantification. We also demonstrate how quasi-mappings can be used to effectively cluster contigs from *de novo* assemblies. We show that the resulting clusterings are of comparable or superior accuracy to those produced by recent methods such as CORSET [4], but that they can be computed *much* more quickly using quasi-mapping.

2 Methods

The quasi-mapping algorithm, 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. 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 by the suffix array, to enable ultrafast 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 the mapping information provided by the quasi-mapping procedure. 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).

Quasi-mapping Quasi-mapping makes use of two main data structures, the generalized suffix array [24] $\mathtt{SA}[T]$ 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 [34] 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 $\mathtt{I}\left(k_i\right) = [b, e)$. 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) [5] 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 [24] algorithm. For speed and simplicity, we implement the "simple accelerant" binary search variant of Gusfield

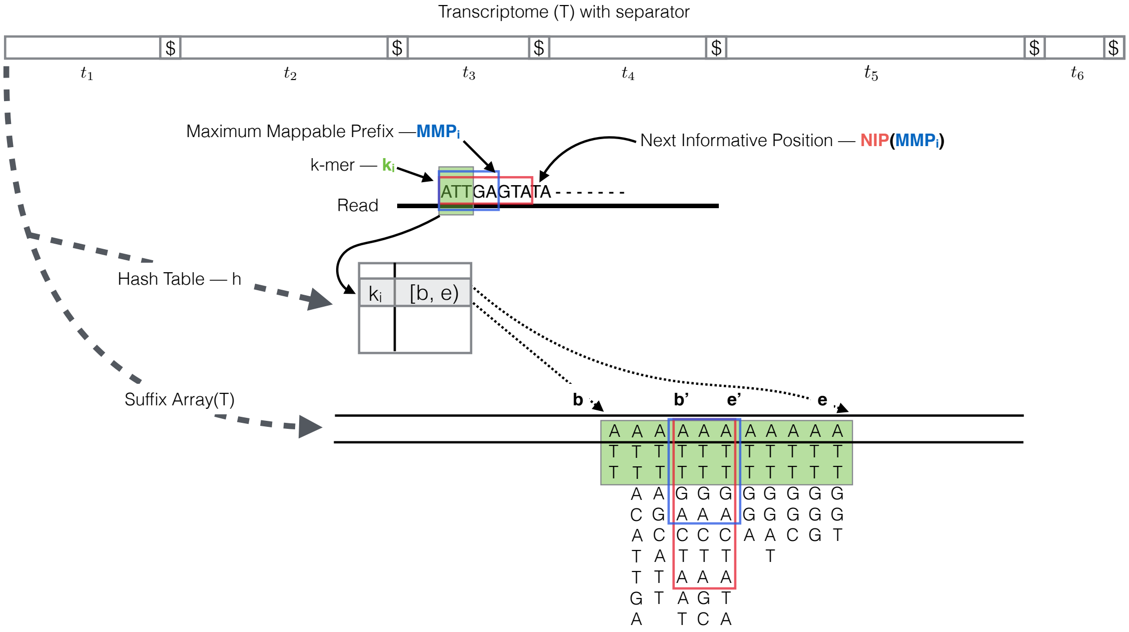
Transcriptome (T) with separator
$
t_1
t_2
t_3
t_4
t_5
t_6
Maximum Mappable Prefix — MMPi
k-mer — ki
Next Informative Position — NIP(MMPi)
Read
ATTGAGTATA
Hash Table — h
ki
[b, e)
Suffix Array(T)
b
b'
e'
e

Figure 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. The search for the next hashable k-mer begins k bases before $\texttt{NIP}(\text{MMP}_i)$.

[9]. 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 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 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 *et al.* [11]. We found this to be the fastest approach. Additionally, it doesn't require the maintainence 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 $\mathtt{I}(k_i)$ corresponding to the substring of the read consisting of this k-mer. Then, the procedure described above is used to compute MMP_i and $\ell = \mathtt{NIP}(\mathrm{MMP}_i)$. 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 `Bowtie 2` [14] and `STAR` [5] and the recently-introduced pseudo-alignment procedure used in the quantification tool `Kallisto` [1]. 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.

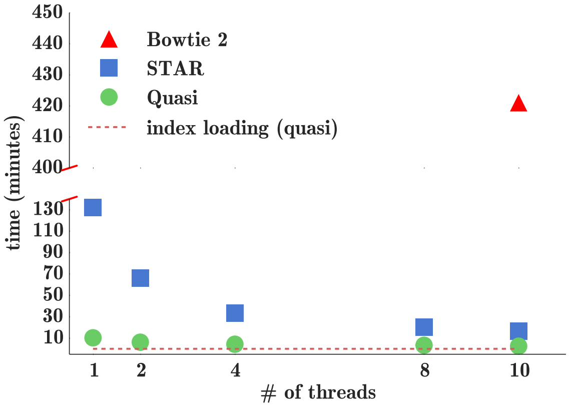

Figure 2: The time taken by `Bowtie 2`, `STAR` and `RapMap` to process the synthetic data using varying numbers of threads. `RapMap` processes the data substantially faster than the other tools, while providing results of comparable or better accuracy.

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 [8]. This synthetic dataset was generated for the human transcriptome from an annotation taken from the ENSEMBL [3] 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 paramters for `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 `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, `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 [25] 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.

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 positve) 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 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 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`.

Table 1: Accuracy of aligners/mappers under different metrics

	`Bowtie 2`	`Kallisto`	`RapMap`	`STAR`
reads aligned	47579567	44774502	47677356	44711604
recall	97.41	91.53	97.62	91.35
precision	98.31	97.67	98.35	97.02
F1-score	97.86	94.50	97.98	94.10
FDR	1.69	2.33	1.65	2.98
hits per read	5.98	5.30	4.30	3.80

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.* [2]. These sequencing reads are derived from human lymphoblastoid cell line GM12878 (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 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 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 $e_{Ai} = \{r_i, t_1, t_2, t_3\} \in \mathcal{U}$. 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, tool A and tool 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 3, which illustrates the size of each set of potential intersections between the results of the tools we consider.

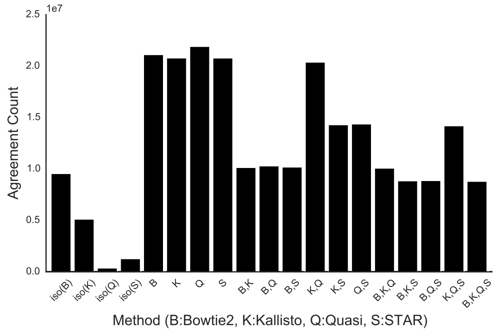

Figure 3: Read agreement between `Bowtie 2`, `STAR`, `Kallisto` and `RapMap`. iso (M) is the set containing only elements unique to method M.

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 that 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 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 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.

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 [30, 35, 1, 31] 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 [30] 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 [18], Tigar2 [27] and `Kallisto` [1], the last of which is based on the pseudo-alignment concept mentioned above.

4.1 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 [18] denoted by

$$\eta_i = \frac{c_i l_i}{\sum_{j=1}^{M} c_j l_j}. \tag{1}$$

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 [18], which is given by

$$\tau_i = \frac{\frac{\eta_i}{l_i}}{\sum_{j=1} \frac{\eta_i}{l_i}}. \tag{2}$$

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 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 [30] 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.* [28], 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.* [31], the equivalence classes are denoted as $\boldsymbol{\mathcal{C}} = \{\mathcal{C}^1, \mathcal{C}^2, \ldots\}$, 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 $\mathbf{t}^j = (t_{j1}, t_{j2}, \ldots)$ 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 $\mathcal{C}^j$.

4.2.1 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

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

	Flux simulation				RSEM-sim simulation			
	Kallisto	RSEM	q-Sailfish	Tigar 2	Kallisto	RSEM	q-Sailfish	Tigar 2
Proportionality corr.	0.74	0.78	0.75	0.77	0.91	0.93	0.92	0.93
Spearman corr.	0.69	0.73	0.71	0.72	0.91	0.93	0.92	0.93
TPEF	0.77	0.96	0.61	0.59	0.53	0.49	0.52	0.50
TPME	-0.24	-0.37	-0.10	-0.09	-0.00	-0.01	0.00	0.00
MARD	0.36	0.29	0.31	0.26	0.29	0.25	0.26	0.23
wMARD	4.68	5.23	4.47	4.35	1.00	0.88	1.02	0.94

et al. [26].

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:

$$\alpha_i^{u+1} = \alpha_0 + \sum_{\mathcal{C}^j \in \mathcal{C}} d_j \left(\frac{e^{\gamma_i^u} \frac{1}{\hat{l}_i}}{\sum_{t_k \in \mathbf{t}^j} e^{\gamma_k^u} \frac{1}{\hat{l}_k}} \right), \quad (3)$$

where

$$\gamma_i^u = \Psi(\alpha_0 + \alpha_i^u) - \Psi(\sum_k \alpha_0 + \alpha_k^u) \quad (4)$$

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.* [18]. To determine the final estimated counts — $\boldsymbol{\alpha}$ — 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 $\boldsymbol{\alpha}$, we compute the TPM for transcript i as

$$TPM_i = 10^6 \frac{\frac{\alpha_i}{\hat{l}_i}}{\sum_j \frac{\alpha_j}{\hat{l}_j}}. \quad (5)$$

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 [18], Tigar 2 [27], and `Kallisto` [1] using 6 different accuracy metrics and data from two different simulation pipelines. One of the simulated datasets was generated with the Flux Simulator [8], 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.* [1]. That is, RSEM was run on sample `NA12716_7` of the Geuvadis RNA-seq data [16] 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 [23], 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

$$\mathrm{RE_i} = \frac{x_i - y_i}{x_i}. \quad (6)$$

The error indicator for transcript i (EI_i) is given by

$$\mathrm{EI_i} = \begin{cases} 1 & \text{if } |RE_i| > 0.1 \\ 0 & \text{otherwise} \end{cases}, \quad (7)$$

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

$$\text{TPEF} = \frac{1}{|X^+|} \sum_{i \in X^+} EI_i. \tag{8}$$

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

$$\text{TPME} = \text{median}\left(\{\text{RE}_i\}_{i \in X^+}\right). \tag{9}$$

Finally, the absolute relative difference for transcript i (ARD_i) is defined as

$$\text{ARD}_i = \begin{cases} 0 & \text{if } x_i + y_i = 0 \\ \frac{|x_i - y_i|}{0.5(x_i + y_i)} & \text{otherwise} \end{cases}. \tag{10}$$

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

$$\text{MARD} = \frac{1}{M} \sum_i \text{ARD}_i \tag{11}$$

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

$$\text{wMARD} = \sum_{i \in \text{ARD}^+} \frac{\log\left(\max\left(x_i, y_i\right)\right) \text{ARD}_i}{M}, \tag{12}$$

where, $\text{ARD}^+ = \{i | ARD_i > 0\}$, 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 ARD_i 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 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.

Table 3: Performance of CORSET, CD-HIT and `RapMap` enabled clustering (R-CL) on yeast and human data

	Human			Yeast		
	CORSET	CD-HIT	R-CL	CORSET	CD-HIT	R-CL
precision	0.96	0.96	0.95	0.36	0.41	0.36
recall	0.56	0.37	0.60	0.63	0.36	0.71
time (min)	957	268	8	23	5	2

5 Application of quasi-mapping for clustering *de novo* assemblies

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 separate assembled contigs. Davidson and Oshlack [4] 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 [6]. CD-HIT compares the sequences (contigs) directly, and clusters them by sequence similarity. CORSET, alternatively, aligns reads to contigs (allowing multimapping) and defines a distance between each pair of contigs based on the number of multimapping 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.

Here, we show how `RapMap` can be used for the same task, by taking an approach similar to that of CORSET. First, we map the RNA-seq reads to the target contigs and simultaneously construct equivalence classes over the mapped fragments as in Section 4. We construct a weighted, undirected graph from these equivalence classes as follows. Given a set of contigs $\mathbf{c}$ and the equivalence classes $\mathcal{C}$, we construct $G = (V, E)$ such that $V = \mathbf{c}$, and $E = \{\{u, v\} \mid \exists j : u, v \in \mathbf{t}^j\}$. We define the weight of edge $\{u, v\}$ as $w(u, v) = \frac{R_{u,v}}{\min(R_u, R_v)}$. Here R_u is the total number of reads belonging to all equivalences classes in which contig u appears in the label. R_v is defined analogously. $R_{u,v}$ is the total sum of reads in all equivalence classes for which contigs u and v appear in the label. Given the undirected graph G, we use the *Markov Cluster Algorithm*, as implemented in MCL [33], to cluster the graph.

To benchmark the time and accuracy of our clustering scheme compared to CD-HIT and CORSET, we used two datasets from the CORSET paper [4]. The first dataset is from human and consists of 231 million reads in total, across two conditions, each with three replicates (as originally described by Trapnell *et al.* [32]). The second dataset, from yeast, was originally published by Nookaew *et al.* [29] and consists 36 million reads, grown in two different conditions with three replicates each. For both of these datasets, we consider clustering the contigs of the corresponding *de novo* assemblies, which were generated using Trinity [7].

To measure accuracy, we consider the precision and recall induced by a clustering with respect to the true genes from which each contig originates. Assembled contigs were mapped to annotated transcripts using BLAT [12], and labeled with their gene of origin. A pair of contigs from the same cluster is regarded as true positive (tp) if they are from the same gene in the ground truth set. Similarly, a pair is a false positive (fp) if they are not from same gene but are clustered together. A pair is a false negative (fn) if they are from same gene but not predicted to be in the same cluster and all the remaining pairs are true negatives (tn). With these definitions of tp, fp, tn and fn we can define precision and recall in standard manner. As shown in Table 3, when considering both precision and recall, `RapMap` (quasi-mapping) enabled clustering performs substantially better than CD-HIT and similar to CORSET. `RapMap` enabled clustering takes 8 minutes and 2 minutes to cluster the human and yeast datasets respectively — which is substantially faster than the other tools. To generate the timing results above, CD-HIT was run with 25 threads. The time recorded for CORSET consists of both the time required to align the reads using `Bowtie 2` (using 25 threads) and the time required to perform the actual clustering, which is single threaded. The time recorded for `RapMap` enabled clustering consists of the time required to quasi-map the reads, build the equivalence classes and construct the graph (using 25 threads), plus the time required to cluster the graph with MCL (using a single thread). Overall, on these datasets, `RapMap`-enabled clustering appears to provide comparable or better clusterings than existing methods, and produces these clusterings much more quickly.

6 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.

In addition to showing the speed and accuracy of quasi-mapping directly, we apply it to two problems in transcriptome analysis. First, 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. We also demonstrate the application of `RapMap` to the problem of clustering *de novo* assembled contigs, a task that has been shown to improve expression quantification and downstream differential expression analysis [4]. `RapMap` can produce clusterings of comparable or superior accuracy to those of existing tools, and can do so much more quickly.

However, `RapMap` is a stand-alone mapping program, and need not be used only for the applications we describe here. We expect that 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 among others.

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.* [1] 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 un-necessary 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

[1] Bray, N., Pimentel, H., Melsted, P., and Pachter, L. (2015). Near-optimal RNA-Seq quantification. *arXiv preprint arXiv:1505.02710*.

[2] Cho, H., Davis, J., Li, X., Smith, K. S., Battle, A., and Montgomery, S. B. (2014). High-resolution transcriptome analysis with long-read RNA sequencing. *PLoS ONE*, **9**(9), e108095.

[3] Cunningham, F., Amode, M. R., Barrell, D., Beal, K., Billis, K., Brent, S., Carvalho-Silva, D., Clapham, P., Coates, G., Fitzgerald, S., *et al.* (2015). Ensembl 2015. *Nucleic acids research*, **43**(D1), D662–D669.

[4] Davidson, N. M. and Oshlack, A. (2014). Corset: enabling differential gene expression analysis for de novo assembled transcriptomes. *Genome biology*, **15**(7), 410.

[5] Dobin, A., Davis, C. A., Schlesinger, F., Drenkow, J., Zaleski, C., Jha, S., Batut, P., Chaisson, M., and Gingeras, T. R. (2013). Star: ultrafast universal rna-seq aligner. *Bioinformatics*, **29**(1), 15–21.

[6] Fu, L., Niu, B., Zhu, Z., Wu, S., and Li, W. (2012). Cd-hit: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, **28**(23), 3150–3152.

[7] Grabherr, M. G., Haas, B. J., Yassour, M., Levin, J. Z., Thompson, D. A., Amit, I., Adiconis, X., Fan, L., Raychowdhury, R., Zeng, Q., *et al.* (2011). Full-length transcriptome assembly from rna-seq data without a reference genome. *Nature biotechnology*, **29**(7), 644–652.

[8] Griebel, T., Zacher, B., Ribeca, P., Raineri, E., Lacroix, V., Guigó, R., and Sammeth, M. (2012). Modelling and simulating generic rna-seq experiments with the flux simulator. *Nucleic acids research*, **40**(20), 10073–10083.

[9] Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA.

[10] Hach, F., Hormozdiari, F., Alkan, C., Hormozdiari, F., Birol, I., Eichler, E. E., and Sahinalp, S. C. (2010). mrsfast: a cache-oblivious algorithm for short-read mapping. *Nature methods*, **7**(8), 576–577.

[11] Ilie, L., Navarro, G., and Tinta, L. (2010). The longest common extension problem revisited and applications to approximate string searching. *Journal of Discrete Algorithms*, **8**(4), 418–428.

[12] Kent, W. J. (2002). Blat?the blast-like alignment tool. *Genome research*, **12**(4), 656–664.

[13] Kim, D., Langmead, B., and Salzberg, S. L. (2015). Hisat: a fast spliced aligner with low memory requirements. *Nature methods*, **12**(4), 357–360.

[14] Langmead, B. and Salzberg, S. L. (2012). Fast gapped-read alignment with Bowtie 2. *Nature methods*, **9**(4), 357–359.

[15] Langmead, B., Trapnell, C., Pop, M., Salzberg, S. L., *et al.* (2009). Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, **10**(3), R25.

[16] Lappalainen, T., Sammeth, M., Friedländer, M. R., AC‘t Hoen, P., Monlong, J., Rivas, M. A., Gonzàlez-Porta, M., Kurbatova, N., Griebel, T., Ferreira, P. G., *et al.* (2013). Transcriptome and genome sequencing uncovers functional variation in humans. *Nature*.

[17] Li, B. and Dewey, C. N. (2011). Rsem: accurate transcript quantification from rna-seq data with or without a reference genome. *BMC bioinformatics*, **12**(1), 323.

[18] Li, B., Ruotti, V., Stewart, R. M., Thomson, J. A., and Dewey, C. N. (2010). Rna-seq gene expression estimation with read mapping uncertainty. *Bioinformatics*, **26**(4), 493–500.

[19] Li, H. (2013). Aligning sequence reads, clone sequences and assembly contigs with bwa-mem.

[20] Li, H. and Durbin, R. (2009). Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, **25**(14), 1754–1760.

[21] Li, H., Ruan, J., and Durbin, R. (2008). Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome research*, **18**(11), 1851–1858.

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

[23] Lovell, D., Pawlowsky-Glahn, V., Egozcue, J. J., Marguerat, S., and Bähler, J. (2015). Proportionality: a valid alternative to correlation for relative data. *PLoS computational biology*, **11**(3), e1004075.

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

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

[26] Nariai, N., Hirose, O., Kojima, K., and Nagasaki, M. (2013). Tigar: transcript isoform abundance estimation method with gapped alignment of rna-seq data by variational bayesian inference. *Bioinformatics*, **29**, btt381.

[27] Nariai, N., Kojima, K., Mimori, T., Sato, Y., Kawai, Y., Yamaguchi-Kabata, Y., and Nagasaki, M. (2014). Tigar2: sensitive and accurate estimation of transcript isoform expression with longer rna-seq reads. *BMC genomics*, **15**(Suppl 10), S5.

[28] Nicolae, M., Mangul, S., Mandoiu, I., and Zelikovsky, A. (2011). Estimation of alternative splicing isoform frequencies from rna-seq data. *Algorithms for Molecular Biology*, **6:9**.

[29] Nookaew, I., Papini, M., Pornputtpong, N., Scalcinati, G., Fagerberg, L., Uhlén, M., and Nielsen, J. (2012). A comprehensive comparison of rna-seq-based transcriptome analysis from reads to differential gene expression and cross-comparison with microarrays: a case study in saccharomyces cerevisiae. *Nucleic acids research*, page gks804.

[30] Patro, R., Mount, S. M., and Kingsford, C. (2014). Sailfish enables alignment-free isoform quantification from rna-seq reads using lightweight algorithms. *Nature biotechnology*, **32**(5), 462–464.

[31] Patro, R., Duggal, G., and Kingsford, C. (2015). Salmon: Accurate, versatile and ultrafast quantification from rna-seq data using lightweight-alignment. *bioRxiv*, **9**, 021592.

[32] Trapnell, C., Hendrickson, D. G., Sauvageau, M., Goff, L., Rinn, J. L., and Pachter, L. (2013). Differential analysis of gene regulation at transcript resolution with rna-seq. *Nature biotechnology*, **31**(1), 46–53.

[33] Van Dongen, S. (2000). A cluster algorithm for graphs. *Report-Information systems*, (10), 1–40.

[34] Vigna, S. (2008). Broadword implementation of rank/select queries. In *Experimental Algorithms*, pages 154–168. Springer.

[35] Zhang, Z. and Wang, W. (2014). Rna-skim: a rapid method for rna-seq quantification at transcript level. *Bioinformatics*, **30**(12), i283–i292.

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 alignemnts.

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 folloing 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
```