Rail-RNA: Scalable analysis of RNA-seq splicing and coverage

Abhinav Nellore[1, 2, 3], Leonardo Collado-Torres[2, 3, 4], Andrew E. Jaffe[2, 3, 4, 5], James Morton[6], Jacob Pritt[1, 3], José Alquicira-Hernández[2, 7], Jeffrey T. Leek[2, 3], and Ben Langmead[1, 2, 3]

[1]Department of Computer Science, Johns Hopkins University
[2]Department of Biostatistics, Johns Hopkins University
[3]Center for Computational Biology, Johns Hopkins University
[4]Lieber Institute for Brain Development, Johns Hopkins Medical Campus
[5]Department of Mental Health, Johns Hopkins University
[6]Department of Computer Science, University of Colorado Boulder
[7]Undergraduate Program on Genomic Sciences, National Autonomous University of Mexico

May 7, 2015

RNA sequencing (RNA-seq) experiments now span hundreds to thousands of samples. Current spliced alignment software is designed to analyze each sample separately. Consequently, no information is gained from analyzing multiple samples together, and it is difficult to reproduce the exact analysis without access to original computing resources. We describe Rail-RNA, a cloud-enabled spliced aligner that analyzes many samples at once. Rail-RNA eliminates redundant work across samples, making it more efficient as samples are added. For many samples, Rail-RNA is more accurate than annotation-assisted aligners. We use Rail-RNA to align 666 RNA-seq samples from the GEUVADIS project on Amazon Web Services in 12 hours for US$0.69 per sample. Rail-RNA produces alignments and base-resolution bigWig coverage files, ready for use with downstream packages for reproducible statistical analysis. We identify expressed regions in the GEUVADIS samples and show that both annotated and unannotated (novel) expressed regions exhibit consistent patterns of variation across populations and with respect to known confounders. Rail-RNA is open-source software available at http://rail.bio.

1 Introduction

Sequencing throughput has improved rapidly in the last several years [1] while cost has continued to drop [2]. RNA sequencing (RNA-seq) [3, 4], a common use of sequencing, involves isolating and sequencing mRNA from biological samples. RNA-seq has rapidly become a standard tool for studying gene expression due to its ability to detect novel transcriptional activity in a largely unbiased manner without relying on previously defined gene sequence. Indeed, the Sequence Read Archive contains data for over 140,000 RNA-seq experiments, including over 35,000 from human-derived samples [5]. RNA-seq continues to accumulate rapidly: large-scale projects like GTEx [6] and TCGA [7] are generating RNA-seq data on thousands of samples across normal and malignant tissues derived from hundreds to thousands of unique individuals.

The goal of an RNA-seq experiment is often to characterize expression across all samples and identify features associated with outcomes of interest. The first step, read alignment, determines where sequencing reads originated with respect to the reference genome or annotated transcriptome. Unlike read alignment for DNA sequencing reads, RNA-seq alignment should be splice-aware to accommodate intronic sequences that have been spliced out of the mature mRNA transcripts. While RNA-seq data can be generated on hundreds or thousands of samples, the alignment process has up to this point largely been performed on each sample separately [8–24].

We introduce Rail-RNA, a splice-aware, annotation-agnostic read aligner designed to analyze many RNA-seq samples at once. Rail-RNA makes maximal use of data from many samples by (a) borrowing strength across replicates to achieve accurate splice junction detection, even at low coverage, and (b) avoiding effort spent aligning sequences that are redundant either within or across samples. Furthermore, Rail-RNA can be run on a computer cluster at the user's home institution or on a cluster rented from a commercial cloud computing provider at a modest cost per sample. Cloud services rent out standardized units of hardware and software, enabling other Rail-RNA users to easily reproduce large-scale analyses performed in other labs. Rail-RNA's output is compatible with downstream tools for isoform assembly, quantitation, and count- and region-based differential expression.

We demonstrate Rail-RNA is more accurate than other tools, becoming even more accurate as more samples are added. We show Rail-RNA is less susceptible to biases affecting other tools; specifically, (a) Rail-RNA has substantially higher recall of alignments across low-coverage introns and (b) Rail-RNA is accurate without a gene annotation, avoiding annotation bias resulting from potentially incomplete [25] or incorrect transcript annotations. We run Rail-RNA on 666 paired-end lymphoblastoid cell line (LCL) RNA-seq samples from the GEUVADIS study [26], obtaining results in 12 hours at a of cost US\$0.69[1] per sample. This is inexpensive compared to per-sample sequencing costs, which are \$20 or more [27]. We illustrate the usability of Rail-RNA's output by performing a region-based differential expression analysis of the GEUVADIS dataset. Our analysis identifies 290,416 expressed regions, including 21,224 that map to intergenic sequence. We further show that intergenic and annotated regions show similar patterns of variation across populations and with respect to known technical confounders.

Altogether, Rail-RNA is a significant step in the direction of usable software that quickly, reproducibly, and accurately analyzes large numbers of RNA-seq samples at once.

2 Results

The GEUVADIS consortium performed mRNA sequencing of 465 lymphoblastoid cell line samples derived from CEPH (CEU), Finnish (FIN), British (GBR), Toscani (TSI) and Yoruba (YRI) populations of the 1000 Genomes Project [26], giving 666 paired-end samples. Per-sample sequencing depth is summarized in Supplementary material, Sec. 5.1. For information on reproducing all our results, including software versions, see Supplementary material, Sec. 5.2.

2.1 Design principles of Rail-RNA

Rail-RNA follows the MapReduce programming model, and is structured as an alternating sequence of aggregate steps and compute steps. Aggregate steps group and order data in preparation for

[1] Hereafter, we use the symbol \$ to denote USD.

future compute steps. Compute steps run concurrently on ordered groups of data. In this framework, it is natural to aggregate across samples—that is, to group data such that data from various input samples come together in a single compute step. Unlike previous tools, Rail-RNA aggregates across samples at multiple points in the pipeline to increase accuracy (borrowing strength for intron calling) and scalability (through elimination of redundant alignment work).

Rail-RNA can be run in elastic, parallel, or single-computer mode. In single-computer mode, Rail-RNA runs on multiple processors on a single computer. In parallel mode, Rail-RNA runs on any of the variety of cluster setups supported by IPython parallel [28]. These include Sun Grid Engine (SGE), Message Passing Interface (MPI), and StarCluster.

Elastic mode uses Hadoop [29], an implementation of MapReduce [30]. In elastic mode, Rail-RNA is run using the Amazon Web Services (AWS) Elastic MapReduce (EMR) service. EMR is specifically for running Hadoop applications on computer clusters rented on demand from Amazon's Elastic Compute Cloud (EC2). Amazon Simple Storage Service (S3) stores intermediate data and output. There are important advantages and disadvantages to commercial cloud computing services like AWS [31,32]. One advantage is that it facilitates reproducibility: one researcher can reproduce the same hardware and software setup used by another. Another advantage for Rail-RNA, which stores all intermediate and final results in S3, is that there is no risk of exhausting the cluster's disk space even for datasets with many samples. Without these facilities, making scalable software that runs easily on large numbers of RNA-seq samples in different laboratories is quite challenging.

Rail-RNA supports both paired-end and unpaired RNA-seq samples. It uses Bowtie 2 [33] to align reads, including reads spanning introns, so it is naturally both indel-aware (with affine gap penalty) and base quality-aware.

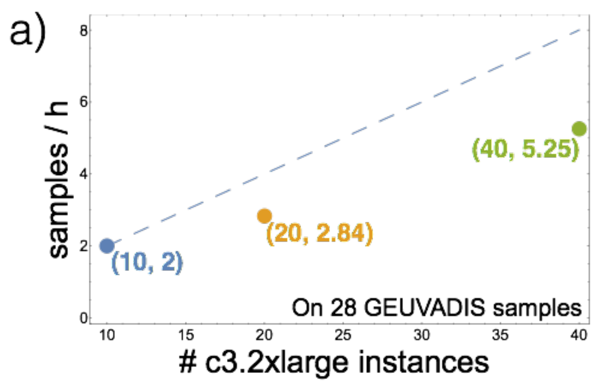
2.2 Scalability

We randomly selected 112 paired-end samples from the GEUVADIS study, with 16 coming from each of the 7 sequencing laboratories. We also randomly selected subsets of 28 and 56 samples from the 112 to illustrate Rail-RNA's scalability on EMR (Supplementary material, Sec. 5.3).

We use the term "instance" to refer to a computer (or virtualized fraction of one) that can be rented from EC2. An instance can have multiple processing cores. For experiments described here: (1) we used c3.2xlarge EC2 spot instances, each with eight processing cores. See Supplementary material, Sec. 5.4 for details on this instance type and Supplementary material, Sec. 5.5 for details on spot instances; (2) we used Amazon's Simple Storage Service (S3) to store inputs, intermediate results, and final results; (3) we performed all experiments and store all S3 data in the US Standard region (us-east-1). See Supplementary material, Sec. 5.6 for details on cost measurements. For every experiment in this paper, we measure cost by totaling the bid price for the EC2 spot instances ($0.11 per instance per hour using spot marketplace) and the Elastic MapReduce surcharges ($0.105 per instance per hour). On the one hand, this estimate can be low since it does not account for costs incurred by DynamoDB, S3 and other related services. On the other, the estimate can be high since the actual price paid depends on the spot market price which is lower than the bid price.

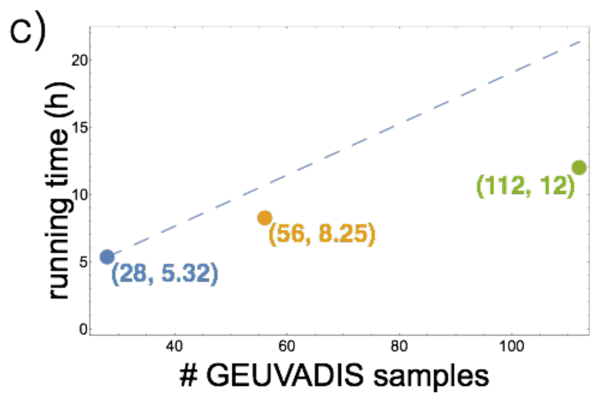
We copied 2.4 terabytes of compressed FASTQ data from the GEUVADIS study from a European Nucleotide Archive FTP server [34,35] to a bucket Amazon's Simple Storage Service (S3). We reserved 20 c3.2xlarge instances and ran Rail-RNA's preprocessing procedure (see Section 3.1) to prepare data for the rest of the analysis. Preprocessing took 1 hour and 8 minutes and cost approximately $8.60.

We ran Rail-RNA several times to assess scalability, which we measured with respect to both



b)

# instances	$/sample
10	1.19
20	1.46
40	1.58



d)

# samples	$/sample
28	1.58
56	1.26
112	0.95

Figure 1: a) depicts scaling with respect to cluster size. Horizontal axis is the number of instances in the cluster. Vertical axis is throughput measured as number of samples analyzed per hour. The dashed line illustrates ideal linear scaling, extrapolated from the 10-instance result. b) is a table of per-sample costs for each experiment in a). c) plots Rail-RNA's running time on 28, 56, and 112 paired-end GEUVADIS samples. The dashed line represents linear scaling, extrapolating that it takes twice as long to analyze approximately twice as much data. Rail-RNA achieves better-than-linear scaling with respect to number of samples through reduced-redundancy analysis, reflected in the table of costs in d): cost per sample decreases as number of samples analyzed increases.

the number of instances in the cluster and the number of input samples. To measure scalability with respect to cluster size, Rail-RNA was run on a random subset of 28 of the 112 samples using EMR clusters of 10, 20, and 40 instances (Figure 1a). The dashed line shows an ideal linear extrapolation from the 10-instance data point. This shows how throughput would increase if doubling the number of instances also doubled throughput. Rail-RNA's scaling falls short of ideal linear scaling, but this is expected due to various sources of overhead including communication and load imbalance. Importantly, Rail-RNA's throughput continues to grow as the number of instances increases to 40, indicating Rail-RNA can make effective use of hundreds of processors at once.

To measure scalability with respect to number of input samples, Rail-RNA was run on 28, 56, and 112 samples using a cluster of 40 instances (Figure 1b). The dashed line extrapolates linear scaling from the 28-sample data point, assuming doubling the number of samples doubles running time. The 56- and 112-sample points fall well below the line, indicating Rail-RNA achieves better-than-linear scaling of running time with respect to number of samples analyzed. This is because Rail-RNA identifies and eliminates redundant alignment work within and across samples. Elimination across samples is particularly beneficial, with cost per sample dropping from \$1.58 for 28 samples to \$0.95 for 112 samples (Figure 1d). In an experiment described in Section 2.4, per-sample cost was reduced to \$0.69 per sample.

2.3 Accuracy

We simulated 112 RNA-seq samples with 40 million 76-bp paired-end reads each using Flux Simulator [36]. Typically, Flux assigns expression levels to transcripts according to a modified Zipf's law. Instead, we used RPKM expression levels from the set of 112 randomly selected paired-end samples studied in Section 2.2; they are taken from the GEUVADIS study [26] and are available at [37]. See Supplementary material, Sec. 5.7 for simulation details.

We compared Rail-RNA's accuracy to that of TopHat 2 v2.0.12 [10], STAR v2.4.0j [11] and HISAT v0.1.5-beta [38]. We ran TopHat 2 with ("Tophat 2 ann") and without ("Tophat 2 no ann") the Gencode v12 annotation [39] provided. We ran STAR in three ways: in one pass ("STAR 1 pass"); in one pass with introns from Gencode v12 provided ("STAR 1 pass ann"); and in two passes ("STAR 2 pass"). We similarly ran HISAT in three ways: in one pass ("HISAT 1 pass"); in one pass with introns from Gencode v12 provided ("HISAT 1 pass ann"); and in two passes ("HISAT 2 pass"). One-pass methods align reads and call introns in one step, whereas two-pass methods additionally perform a second step that realigns reads in light of intron calls from the first. Supplementary material, Sec. 5.8 describes the protocols in detail.

When an alignment program is run with annotation ("Tophat 2 ann," "STAR 1 pass ann," and "HISAT 1 pass ann"), we provide the same annotation from which Flux Simulator simulated the reads. That is, the provided annotation consists of a superset of the actual transcripts present. Consequently, protocols where the annotation is provided have an artificial advantage.

Rail-RNA was run in three ways:

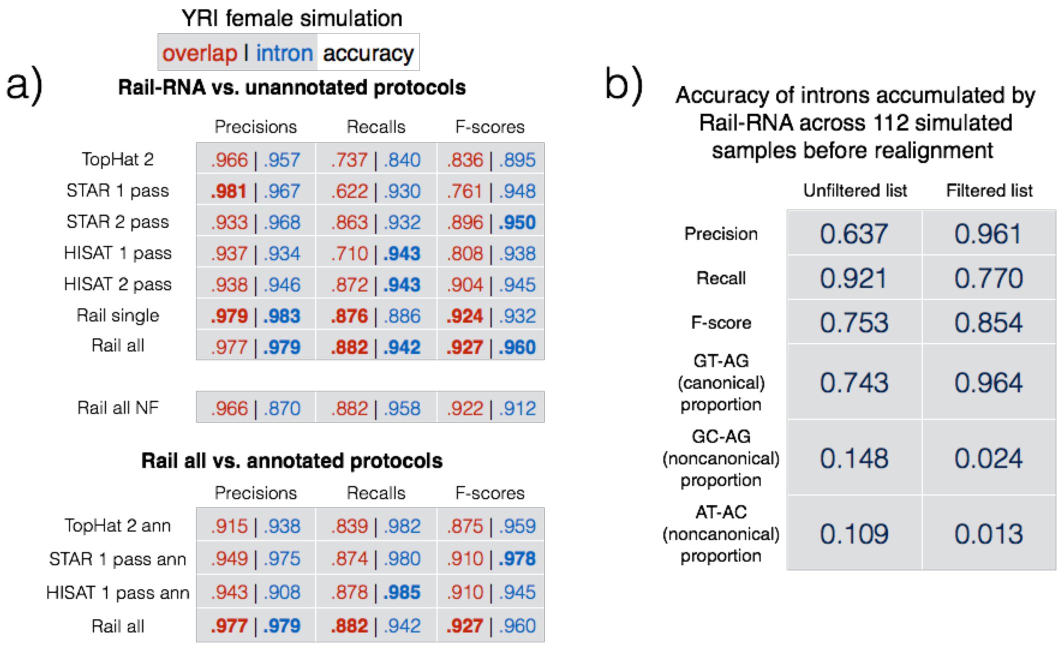
1. **On a single sample ("Rail single").** Rail-RNA uses reads from the given sample to identify introns. Like in two-pass protocols, reads are then realigned to transcript fragments to handle the case where a read overlaps an intron near its 5' or 3' end.

2. **On 112 samples with intron filter ("Rail all").** After initial alignment, Rail-RNA compiles a list of introns found in any sample. The list is then filtered; an intron passes the filter if (a) it appears in at least 5% of the input samples, or (b) it is overlapped by at least 5 reads in any sample. The filtered intron list is used to build transcript fragments for the realignment step.

3. **On 112 samples without intron filter ("Rail all NF").** Identical to "Rail all" but with no intron filter. This is not a recommended protocol; we include it only to show the filter's effect.

In none of the three modes does Rail-RNA use a gene annotation: Rail-RNA is consistently annotation-agnostic.

We report two kinds of accuracy: (1) overlap accuracy, measuring precision and recall of overlap events. Here, each event is an instance where the primary alignment of a read overlaps an intron; (2) intron accuracy, measuring precision of unique introns called by a given aligner and recall of the set of unique introns within a sample or across several samples. For each precision-recall pair, we also compute F-score, the harmonic mean of precision and recall. For detailed definitions, see Supplementary material, Sec. 5.9.



a) YRI female simulation

overlap | intron accuracy

Rail-RNA vs. unannotated protocols

	Precisions	Recalls	F-scores
TopHat 2	.966 | .957	.737 | .840	.836 | .895
STAR 1 pass	**.981** | .967	.622 | .930	.761 | .948
STAR 2 pass	.933 | .968	.863 | .932	.896 | **.950**
HISAT 1 pass	.937 | .934	.710 | **.943**	.808 | .938
HISAT 2 pass	.938 | .946	.872 | **.943**	.904 | .945
Rail single	**.979** | **.983**	**.876** | .886	**.924** | .932
Rail all	.977 | **.979**	**.882** | **.942**	**.927** | **.960**
Rail all NF	.966 | .870	.882 | .958	.922 | .912

Rail all vs. annotated protocols

	Precisions	Recalls	F-scores
TopHat 2 ann	.915 | .938	.839 | .982	.875 | .959
STAR 1 pass ann	.949 | .975	.874 | .980	.910 | **.978**
HISAT 1 pass ann	.943 | .908	.878 | **.985**	.910 | .945
Rail all	**.977** | **.979**	**.882** | .942	**.927** | .960

b) Accuracy of introns accumulated by Rail-RNA across 112 simulated samples before realignment

	Unfiltered list	Filtered list
Precision	0.637	0.961
Recall	0.921	0.770
F-score	0.753	0.854
GT-AG (canonical) proportion	0.743	0.964
GC-AG (noncanonical) proportion	0.148	0.024
AT-AC (noncanonical) proportion	0.109	0.013

Figure 2: (a) Accuracy results of various tools for a single simulated sample whose genome coverage mimics that of a YRI female GEUVADIS sample. "Rail all" protocols involve running Rail-RNA on all 112 samples we simulated, including the YRI female. Overlap accuracy metrics are shown in red, intron accuracy metrics in blue. Some tools are run with (top) and without (bottom) a provided gene annotation. Note that Rail-RNA never uses an annotation. For Rail-RNA vs. unannotated protocols, the best two results in each column are in boldface, and for Rail-RNA vs. annotated protocols, the best result in each column is in boldface. (b) Rail-RNA's accuracy on two sets of introns found across all 112 simulated samples: one before and one after application of the intron filter.

The "Rail all" and "Rail all NF" protocols were run on all 112 GEUVADIS-based simulations. Other protocols were run on the "YRI female" simulated sample, based on the expression profile of the GEUVADIS sample labeled "NA19129.6.M_120217_1." Figure 2 displays overlap and intron accuracy measurements for "YRI female."

As illustrated in Figure 2a, Rail-RNA has the highest overlap F-score of the protocols tested, including those using a gene annotation. Rail-RNA's overlap precision is comparable to the highest of any protocol ("STAR 1 pass" in this case) and its recall is comparable to the highest of any protocol ("STAR 1 pass ann"). Further, analyzing many samples at once ("Rail all") achieves greater F-score compared to analyzing one ("Rail single"). This is more pronounced for intron accuracy than for overlap accuracy since borrowing strength across replicates is particularly effective at detecting low-coverage introns. See Supplementary material, Sec. 5.10 for details.

Figure 2b demonstrates efficacy of the intron filter in the "Rail all" protocol. Precision/recall are defined similarly as for a single sample, but pooled across all samples. That is, recall is the fraction of introns with at least one simulated spanning read in at least one sample that Rail-RNA detects in at least one sample. The improvement in precision when moving from the unfiltered (0.637) to the filtered (0.961) intron list shows that the filter removes a large fraction of incorrect intron calls because they are supported in only a few samples. Further, the distribution of filtered introns with certain donor/acceptor motifs matches the expected distribution (GT-AG: 95%, GC-AG: 1.7%, AT-AC: 0.16%) much more closely than the same distribution for the unfiltered introns.

2.4 Analysis of GEUVADIS RNA-seq samples

We demonstrate the utility of Rail-RNA's outputs by performing a novel analysis of 666 paired-end GEUVADIS RNA-seq samples from lymphoblastoid cell lines (LCLs) [26]. Starting from FASTQ inputs, Rail-RNA produced bigWig files [40] encoding genomic coverage; per-sample BED files recording locations of introns, insertions and deletions; and sorted, indexed BAM [41] files recording the alignments. Rail-RNA ran on 61 c3.8xlarge Amazon EC2 instances (see 5.4) spanning 1,952 processors. The run completed in 12 hours and cost $453.84 overall ($0.69 per sample).

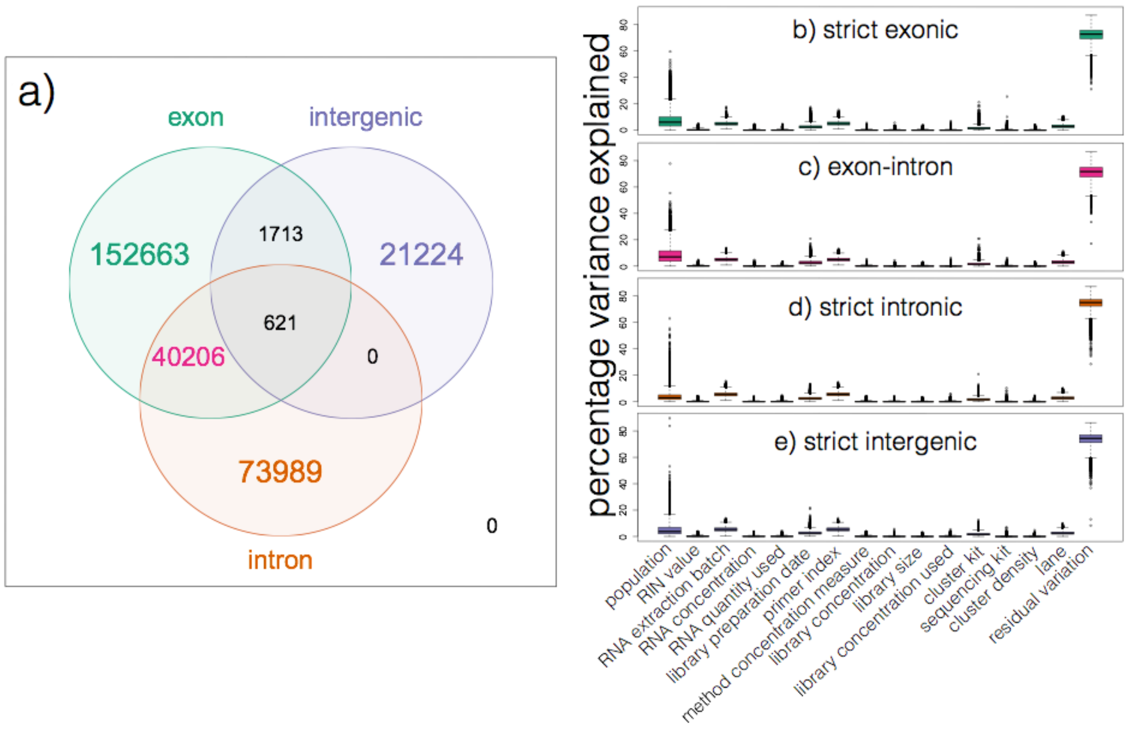


Figure 3: (a) Number of expressed regions (ERs) from the 666 GEUVADIS samples overlapping known exons, introns, and intergenic sequence as determined from Ensembl v75. (b)-(e) Boxplots of percentage variance in expression explained by assorted variables (population and 14 technical variables) as well as residual "unexplained" variation restricted to (b) strictly exonic ERs, (c) ERs overlapping known exons and introns, (d) strictly intronic ERs, and (e) strictly intergenic ERs. Each expressed region corresponds to one point in each of the 15 × 4 boxplots.

We analyzed bigWig outputs using the `derfinder` Bioconductor package [42]. `derfinder` identified contiguous genomic stretches where average coverage across all samples was above a genome-wide threshold (see Supplementary material, Sec. 5.11). Adjacent bases above the threshold were grouped into "expressed regions" (ERs). We identified 290,416 ERs in this way; median ER length was 72 bp (interquartile range, IQR: 7-144). While gene annotation/structure is not used to identify ERs, the regions can be overlapped with a gene annotation to assess novel transcriptional activity, as shown in Figure 3a. Relative to Ensembl v75 [39], we found that 152,663 ERs (52.6%) were within exons (median length: 93 bp, IQR: 21-155) and another 40,206 (13.8%) overlapped both exonic and intronic sequence (median length: 131 bp, IQR: 76-264) perhaps representing alternative ends to known exons. We also found 73,989 regions (25.5%) contained within intronic sequence (median length: 9 bp, IQR: 3-42) and another 21,224 regions (7.3%) within intergenic sequence (median length: 20 bp, IQR: 2-76). These intronic and intergenic ERs could represent novel (polyadenlyated, since the data was generated using polyA+ prep) non-coding RNA.

We also reproduced the variance component modeling illustrated in Figure 3 of Hoen et al 2013 [43]. At each of the 290,416 ERs, we modeled $\log_2$-adjusted expression levels as a function of many largely technical variables related to the RNA (RIN value, RNA concentration, and RNA quantity), the sequencing libraries (library preparation date, library primer index for each sample, method then target and actual quantities of library concentrations, and library size) and the sequencing itself (cluster kit, sequencing kit, cluster density and sequencing lane). We further included the ethnicity of each sample as a variable because it appeared to explain moderate variability in expression levels at many ERs. Lastly, we calculated the amount of residual variability not explained by any of the variable we considered. Interestingly, we found little difference in the amounts of variability explained by each of the variables studied when we stratified the ERs by the annotation categories described above (Figure 3b-e), suggesting that the technical factors affect expression of previously unannotated (i.e. intronic and intergenic) features in a similar manner as they do annotated gene structures.

3 Methods

Rail-RNA seeks the best alignment for each input read as follows. Let $[n]$ denote $\{1, \ldots, n\}$ for some positive integer n. Say Rail-RNA finds a number N of possible alignments of a given read $\mathbf{r}$. Let $i \in [N]$ index these alignments. Each has an alignment score $s(i)$ and a number $n(i)$ of introns overlapped by the alignment. $s(i)$ is determined by Bowtie 2's local alignment parameters [33], which assign penalties to gaps and mismatches in a user-configurable fashion. Consider the set of reads such that each has exactly one highest-scoring alignment in the same sample as $\mathbf{r}$. Call these the uniquely aligned reads. Let $c(i, j)$ be the number of uniquely aligned reads covering the jth intron overlapped by alignment i, and suppose

$$c(i) := \min_{j \in [n(i)]} c(i, j)\,. \tag{1}$$

If $n(i) = 0$, set $c(i) = 1$. Let

$$P := \operatorname*{argmin}_{k \in \operatorname{argmax}_{i \in [N]} s(i)} n(k)\,. \tag{2}$$

P will often have one element. When there is more than one, Rail-RNA draws the primary alignment of $\mathbf{r}$ at random weighted by $c(i)$:

$$p(i) = \frac{c(i)}{\sum_{k \in P} c(k)}\,, \; i \in P\,. \tag{3}$$

In short, we are selecting a parsimonious set of introns that "explain" the reads. In case of ties, we seek to preserve the coverage distributions given by uniquely aligned reads.

The steps in Rail-RNA's analysis pipeline are described below. Figure 4 is an illustration of how Rail-RNA eliminates redundant alignment work. Methods used in the statistical analysis of the GEUVADIS dataset are described in Supplementary material, Sec. 5.11.

3.1 Preprocess reads

Input reads are downloaded as necessary, preprocessed into a format appropriate for Rail-RNA (e.g. combining pairs so both ends are in a single record), and subdivided to enable parallelism. For detail on how load balance is maintained, see Supplementary material, Sec. 5.12.

3.2 Align reads to genome

Reads are partitioned by nucleotide sequence so that workers operate on sets of reads with identical nucleotide sequences. Each nucleotide sequence Q is aligned to the reference genome using Bowtie 2. Depending on the alignment, one of two actions is taken:

1. If the alignment is perfect (an exact end-to-end match), all reads with sequence Q are assigned the same primary alignment placed in set F. This eliminates redundant alignment work. Alignments in F are final and appear in Rail-RNA's terminal output.

2. If the alignment is not perfect, Q is placed in set S_{search} or set $S_{\text{no search}}$. Subsequently, sequences in S_{search} are aligned across introns. Reads whose sequences are in $S_{\text{no search}}$ are never aligned across introns but are later realigned to transcript fragments.

Sequences in S_{search} are divided into short overlapping substrings called readlets. See Supplementary material, Sec. 5.13 for details on this step. In the next step, readlets are aligned to the reference genome for intron detection.

3.3 Align readlets to genome

Readlets from the previous step are partitioned by readlet sequence. This allows workers to operate on all readlets with identical nucleotide sequences at once, even if they originated from various reads and various samples, avoiding redundant alignment effort. Each unique readlet sequence is aligned to the genome using Bowtie [44]. Rail-RNA searches for several possible alignments—by default up to 30 using the command-line parameters `-a -m 30`—of each unique readlet sequence. This step outputs alignments of the unique readlet sequences.

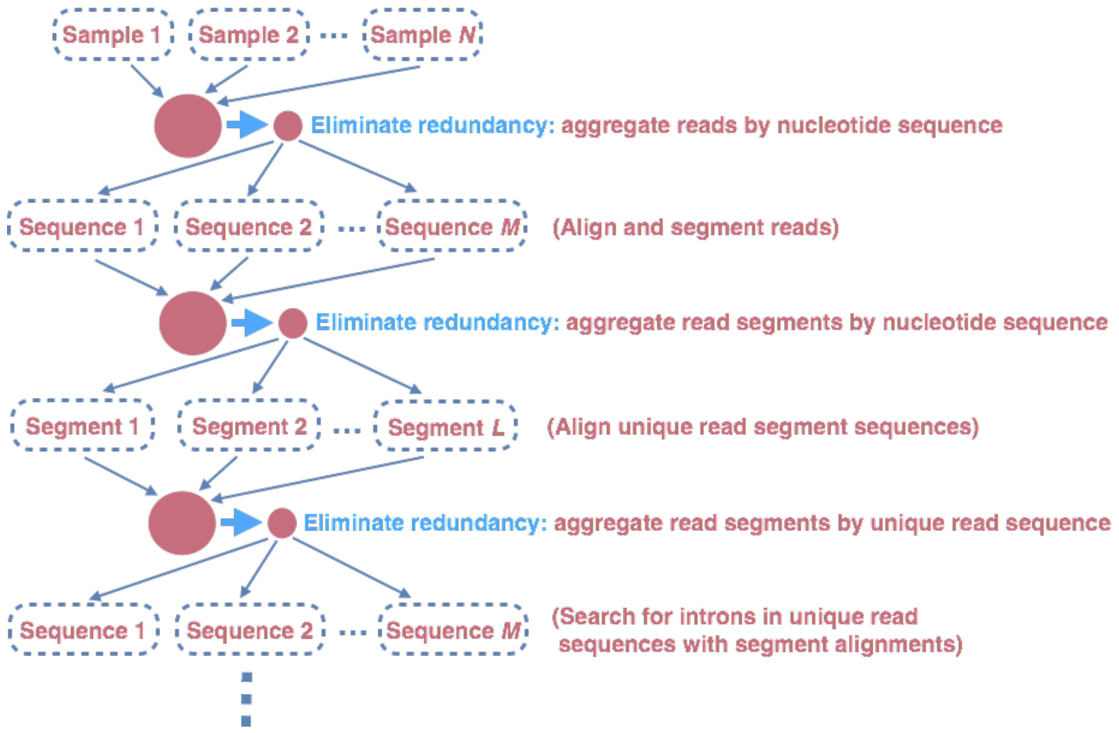

Sample 1
Sample 2
Sample N
Eliminate redundancy: aggregate reads by nucleotide sequence
Sequence 1
Sequence 2
Sequence M
(Align and segment reads)
Eliminate redundancy: aggregate read segments by nucleotide sequence
Segment 1
Segment 2
Segment L
(Align unique read segment sequences)
Eliminate redundancy: aggregate read segments by unique read sequence
Sequence 1
Sequence 2
Sequence M
(Search for introns in unique read sequences with segment alignments)

Figure 4: Rail-RNA eliminates redundant alignment work by first aggregating reads by nucleotide sequence and, where appropriate, aligning only unique read sequences to the genome. Read sequences that do not align perfectly to the genome are divided into overlapping segments, and after these segments are aggregated, each unique segment is aligned to the genome exactly once. Rail-RNA then gathers all segment alignments belonging to a read sequence and searches for introns within each unique read sequence.

3.4 Detect introns using readlet alignments

Output from the previous step is partitioned by read sequence so that all readlet alignments associated with a unique read sequence are handled at once by a given worker. Correlation clustering and maximal clique enumeration are used to detect introns spanned by each read sequence uses, as detailed in Supplementary material, Sec. 5.14. The step outputs introns detected along with information about the intron's coverage in each sample.

3.5 Filter introns violating confidence criteria

We observed in simulations that when many samples are analyzed by Rail-RNA, true positive intron calls from the previous step tend to be present in a substantial fraction of samples, whereas false positives tend to be unique to one sample or a few samples (Figure 2b). Without a filter, the global list of introns tends to grow quickly with the number of samples, with most of the growth attributable to false positives. To keep alignment and junction precision high while borrowing strength from across samples, this step filters out introns violating certain confidence criteria.

The previous step's output is sorted by intron so a given worker can study the samples and number of reads per sample in which a given intron occurs. An intron not meeting one of the following criteria is eliminated from consideration here:

1. The intron appears in at least $K\%$ of samples.
2. The intron is covered by at least J reads in at least one sample.

$K = J = 5$ by default, but both are configurable.

3.6 Enumerate intron configurations along read segments

The output of the previous step is a set of introns with associated information about which samples they appear in. To capture alternative splicings, this step enumerates the ways multiple introns detected in the same sample (and on the same strand) could be overlapped by a read segment $\mathbf{s}(\textit{readlet\_config\_size})$ spanning *readlet_config_size* bases. One possible way that a read or readlet can overlap multiple introns is called an "intron configuration." Intron configurations for readlets are obtained and output as described in Supplementary material, Sec. 5.15.

3.7 Retrieve and index transcript fragments that overlap introns

The previous step's output is partitioned by intron configuration, regardless of the sample from which the configuration originated. In one step, each worker operates on an intron configuration at a time, concatenating the exonic bases surrounding its introns to form a transcript fragment of size *readlet_config_size*. In practice, extra care is taken to avoid including intronic bases in transcript fragments; see Supplementary material, Sec. 5.16 for details. In a second step, exactly one worker operates on all transcript fragments. The command `bowtie2-build` is invoked to create an index of transcript fragments Bowtie 2 can use to realign reads in the next step. In elastic mode, this index is distributed across nodes and is available to all workers in the next step. In local and parallel modes, the index is placed on a filesystem accessible to all workers.

3.8 Finalize intron cooccurrences on read sequences

In this step, unique read sequences from group **B** are aligned to the transcript fragment reference using Bowtie 2 in `--local` mode with a minimum score threshold of 48 by default. Local alignment is needed since indexed sequences are of length *readlet_config_size*, making end-to-end alignments impossible. Rail-RNA runs Bowtie 2 with the parameter `-k 30` by default so that many alignment are reported per read sequence.

Once all introns overlapped by segments of a given read sequence spanning L bases are found, an undirected graph is compiled. A vertex corresponds to a distinct intron (t, s, e), where t denotes strand, s the start coordinate, and e the end coordinate. An edge is placed between two vertices (t_i, s_i, e_i) and (t_j, s_j, e_j) if and only if

1. $t_i = t_j$; the introns are on the same strand.
2. There are no more than $L - 1$ exonic bases between introns i and j.
3. $\min(e_i, e_j) - \max(s_i, s_j) \geq 0$; the introns do not overlap.

The Bron-Kerbosch algorithm is executed on this graph to determine all its maximal cliques. These maximal cliques are considered candidate intron configurations, each of which corresponds to a possible alignment of the read sequence. The output of this step includes intron configurations and corresponding unique read sequences from which they are derived.

3.9 Realign reads

Reads whose associated sequences were placed in S_{search} or $S_{\text{no search}}$ in the "Align reads to genome" step (3.2) are included in the input together with transcript fragments from the previous step. Each worker operates on a group of reads and a corresponding group of transcript fragments. Reads are then aligned to transcript fragments as follows.

1. Transcript fragments are recorded and indexed with `bowtie2-build`.
2. Reads are realigned to the new index using Bowtie 2 in `--local` mode.

Though there is an opportunity to eliminate redundant work in this step, Rail-RNA avoids this to preserve base quality-awareness.

3.10 Collect and compare read alignments

Bowtie 2 alignments of reads accumulated in previous steps, except for those that aligned perfectly in the "Align reads to genome" step (Section 3.2), are collected here and partitioned by read name. A worker operates on all alignments of a given read at once. For each read, if there is exactly one highest-scoring alignment for that read, it is chosen as the primary alignment. Otherwise, Rail-RNA attempts to break the tie by selecting the alignment spanning the fewest introns. If there is still a tie, it is broken by a random draw weighted by the number of uniquely aligned reads supporting each intron, as described by (3).

3.11 Compile coverage vectors and write bigWigs

Rail-RNA records vectors encoding depth of coverage at each position in the reference genome. One bigWig file is produced per sample. Rail-RNA also records a TSV file containing normalization factors for each sample. These facilitate across-sample comparisons, such as differential expression tests. In elastic mode, BED and TSV files are uploaded to S3. For example, the analysis in Section 2.4 was performed on bigWig files on S3. Further, tools such as the UCSC genome browser [45] allow users to visualize portions of bigWig files without having to download them first. These methods are detailed in Supplementary material, Sec. 5.17.

3.12 Write intron and indel BEDs

Indels and introns associated with the primary alignments of reads selected in previous steps are collected and partitioned by distinct intron or indel per sample. In a first step, a given sample's coverage of each intron or indel is computed by a worker. In a second step, three BED files are written per sample: one contains introns, one contains insertions, and one contains deletions. These files are in the same format as TopHat's analogous output [46]. In particular, each line of the intron BED corresponds to a different intron; each intron is specified by two connected BED blocks, where a block spans as many bases as the maximal overhang of all reads in the sample spanning that intron. In elastic mode, all BED files are uploaded to S3, where they can be analyzed as soon as Rail-RNA completes.

3.13 Write BAMs recording alignments

By default, all primary alignments, including the perfect alignments from the "Align reads to genome" step, are output here. The user may specify the `--bowtie2-args="-k X"` parameter to ask Rail-RNA to output up to X highest-scoring alignments per read. Alignments are written as sorted, indexed BAM files. By default, one BAM file is output per sample per chromosome. In elastic mode, all BAM files are uploaded to S3. Tools such as the UCSC genome browser [45] allow users to visualize portions of BAM files without having to download them first.

4 Discussion

Designing software that is both usable and able to run on many samples at a time is technically challenging. Rail-RNA demonstrates that this comes with unique and substantial advantages. By identifying and eliminating redundant alignment work within and across samples, Rail-RNA achieves better-than-linear growth in throughput (and consequently reduction in cost) per sample as the number of samples grows. By using information from all samples when analyzing data from a given sample, Rail-RNA achieves superior accuracy, with its accuracy advantage growing as samples are added. Rail-RNA also substantially resolves two biases affecting other tools: bias against low-coverage introns and annotation bias. Rail-RNA results obtained by one investigator can be reliably reproduced by another since Rail-RNA computer clusters rented from commercial cloud services have standardized hardware and software.

We demonstrated Rail-RNA by re-analyzing a 666-sample dataset in 12 hours at a per-sample cost of $0.69, far lower than sequencing cost [27]. We analyzed region-level differential expression by simply passing Rail-RNA's bigWig outputs to standard downstream tools (e.g. the derfinder Bioconductor packages) for further analysis. Users can reproduce this analysis using resources rented form a commercial cloud provider, without having to downloading any large datasets. Altogether, this as an important step in the direction of usable software that quickly, reproducibly, and accurately analyzes large numbers of RNA-seq samples at once.

Acknowledgments AN, LCT, JTL, and BL were supported by NIH/NIGMS grant 1R01GM105705 to JTL. AN was supported by a seed grant from the Institute for Data Intensive Engineering and Science (IDIES) at Johns Hopkins University to BL and JTL. LCT was supported by Consejo Nacional de Ciencia y Tecnología México 351535. BL and JP were supported by a Sloan Research Fellowship to BL. JAH was supported by Fundación UNAM. BL was supported by National Science Foundation grant IIS-1349906. Experiments using Amazon Web Services were supported by two AWS in Education grants to BL.

References

[1] T. C. Glenn, "Field guide to next-generation dna sequencers," *Molecular Ecology Resources*, vol. 11, no. 5, pp. 759–769, 2011.

[2] E. C. Hayden, "Is the $1,000 genome for real?," *Nature News*, 2014.

[3] Z. Wang, M. Gerstein, and M. Snyder, "Rna-seq: a revolutionary tool for transcriptomics," *Nature Reviews Genetics*, vol. 10, no. 1, pp. 57–63, 2009.

[4] F. Ozsolak and P. M. Milos, "Rna sequencing: advances, challenges and opportunities," *Nature Reviews Genetics*, vol. 12, no. 2, pp. 87–98, 2010.

[5] R. Leinonen, H. Sugawara, and M. Shumway, "The sequence read archive," *Nucleic acids research*, p. gkq1019, 2010.

[6] J. Lonsdale, J. Thomas, M. Salvatore, R. Phillips, E. Lo, S. Shad, R. Hasz, G. Walters, F. Garcia, N. Young, *et al.*, "The genotype-tissue expression (gtex) project," *Nature genetics*, vol. 45, no. 6, pp. 580–585, 2013.

[7] "The Cancer Genome Atlas." `http://cancergenome.nih.gov`, Aug. 2013.

[8] K. Wang, D. Singh, Z. Zeng, S. J. Coleman, Y. Huang, G. L. Savich, X. He, P. Mieczkowski, S. A. Grimm, C. M. Perou, *et al.*, "Mapsplice: accurate mapping of rna-seq reads for splice junction discovery," *Nucleic acids research*, p. gkq622, 2010.

[9] C. Trapnell, L. Pachter, and S. L. Salzberg, "Tophat: discovering splice junctions with rna-seq," *Bioinformatics*, vol. 25, no. 9, pp. 1105–1111, 2009.

[10] D. Kim, G. Pertea, C. Trapnell, H. Pimentel, R. Kelley, and S. L. Salzberg, "Tophat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions," *Genome Biol*, vol. 14, no. 4, p. R36, 2013.

[11] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, and T. R. Gingeras, "Star: ultrafast universal rna-seq aligner," *Bioinformatics*, vol. 29, no. 1, pp. 15–21, 2013.

[12] T. Bonfert, G. Csaba, R. Zimmer, and C. C. Friedel, "A context-based approach to identify the most likely mapping for rna-seq experiments," *BMC bioinformatics*, vol. 13, no. Suppl 6, p. S9, 2012.

[13] N. Philippe, M. Salson, T. Commes, and E. Rivals, "Crac: an integrated approach to the analysis of rna-seq reads," *Genome biology*, vol. 14, no. 3, p. R30, 2013.

[14] J. Hu, H. Ge, M. Newman, and K. Liu, "Osa: a fast and accurate alignment tool for rna-seq," *Bioinformatics*, vol. 28, no. 14, pp. 1933–1934, 2012.

[15] Y. Zhang, E.-W. Lameijer, P. AC't Hoen, Z. Ning, P. E. Slagboom, and K. Ye, "Passion: a pattern growth algorithm-based pipeline for splice junction detection in paired-end rna-seq data," *Bioinformatics*, vol. 28, no. 4, pp. 479–486, 2012.

[16] F. De Bona, S. Ossowski, K. Schneeberger, and G. Rätsch, "Optimal spliced alignments of short sequence reads," *BMC Bioinformatics*, vol. 9, no. Suppl 10, p. O7, 2008.

[17] N. Cloonan, Q. Xu, G. J. Faulkner, D. F. Taylor, D. T. Tang, G. Kolle, and S. M. Grimmond, "Rna-mate: a recursive mapping strategy for high-throughput rna-sequencing data," *Bioinformatics*, vol. 25, no. 19, pp. 2615–2616, 2009.

[18] G. R. Grant, M. H. Farkas, A. D. Pizarro, N. F. Lahens, J. Schug, B. P. Brunk, C. J. Stoeckert, J. B. Hogenesch, and E. A. Pierce, "Comparative analysis of rna-seq alignment algorithms and the rna-seq unified mapper (rum)," *Bioinformatics*, vol. 27, no. 18, pp. 2518–2528, 2011.

[19] S. Huang, J. Zhang, R. Li, W. Zhang, Z. He, T.-W. Lam, Z. Peng, and S.-M. Yiu, "Soapsplice: genome-wide ab initio detection of splice junctions from rna-seq data," *Frontiers in genetics*, vol. 2, 2011.

[20] K. F. Au, H. Jiang, L. Lin, Y. Xing, and W. H. Wong, "Detection of splice junctions from paired-end rna-seq data by splicemap," *Nucleic acids research*, vol. 38, no. 14, pp. 4570–4578, 2010.

[21] D. W. Bryant, R. Shen, H. D. Priest, W.-K. Wong, and T. C. Mockler, "Supersplatspliced rna-seq alignment," *Bioinformatics*, vol. 26, no. 12, pp. 1500–1505, 2010.

[22] T. D. Wu and S. Nacu, "Fast and snp-tolerant detection of complex variants and splicing in short reads," *Bioinformatics*, vol. 26, no. 7, pp. 873–881, 2010.

[23] S. Marco-Sola, M. Sammeth, R. Guigó, and P. Ribeca, "The gem mapper: fast, accurate and versatile alignment by filtration," *Nature methods*, vol. 9, no. 12, pp. 1185–1188, 2012.

[24] G. Jean, A. Kahles, V. T. Sreedharan, F. D. Bona, and G. Rätsch, "Rna-seq read alignments with palmapper," *Current protocols in bioinformatics*, pp. 11–6, 2010.

[25] A. E. Jaffe, J. Shin, L. Collado-Torres, J. T. Leek, R. Tao, C. Li, Y. Gao, Y. Jia, B. J. Maher, T. M. Hyde, *et al.*, "Developmental regulation of human cortex transcription and its clinical relevance at single base resolution," *Nature neuroscience*, 2014.

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

[27] P. A. Combs and M. B. Eisen, "Low-cost, low-input rna-seq protocols perform nearly as well as high-input protocols," *PeerJ PrePrints*, vol. 3.

[28] F. Perez and B. E. Granger, "Ipython: a system for interactive scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 21–29, 2007.

[29] "Welcome to Apache Hadoop." http://hadoop.apache.org, Aug. 2013.

[30] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[31] M. C. Schatz, B. Langmead, and S. L. Salzberg, "Cloud computing and the dna data race," *Nature biotechnology*, vol. 28, no. 7, pp. 691–693, 2010.

[32] L. D. Stein *et al.*, "The case for cloud computing in genome informatics," *Genome Biol*, vol. 11, no. 5, p. 207, 2010.

[33] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nature methods*, vol. 9, no. 4, pp. 357–359, 2012.

[34] "GEUVADIS data access portal from the european nucleotide archive." http://www.ebi.ac.uk/ena/data/view/ERP001942. Accessed: 2014-10-02.

[35] R. Leinonen, R. Akhtar, E. Birney, L. Bower, A. Cerdeno-Tárraga, Y. Cheng, I. Cleland, N. Faruque, N. Goodgame, R. Gibson, *et al.*, “The european nucleotide archive,” *Nucleic acids research*, p. gkq967, 2010.

[36] T. Griebel, B. Zacher, P. Ribeca, E. Raineri, V. Lacroix, R. Guigó, and M. Sammeth, “Modelling and simulating generic rna-seq experiments with the flux simulator,” *Nucleic acids research*, vol. 40, no. 20, pp. 10073–10083, 2012.

[37] “Rpkms for mrna sequencing data computed by GEUVADIS consortium.” http://www.ebi.ac.uk/arrayexpress/files/E-GEUV-3/GD660.TrQuantRPKM.txt.gz. Accessed: 2014-10-02.

[38] D. Kim, B. Langmead, and S. L. Salzberg, “Hisat: a fast spliced aligner with low memory requirements,” *Nature methods*, 2015.

[39] F. Cunningham, M. R. Amode, D. Barrell, K. Beal, K. Billis, S. Brent, D. Carvalho-Silva, P. Clapham, G. Coates, S. Fitzgerald, *et al.*, “Ensembl 2015,” *Nucleic acids research*, vol. 43, no. D1, pp. D662–D669, 2015.

[40] W. J. Kent, A. S. Zweig, G. Barber, A. S. Hinrichs, and D. Karolchik, “Bigwig and bigbed: enabling browsing of large distributed datasets,” *Bioinformatics*, vol. 26, no. 17, pp. 2204–2207, 2010.

[41] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, *et al.*, “The sequence alignment/map format and samtools,” *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[42] L. Collado-Torres, A. C. Frazee, M. I. Love, R. A. Irizarry, A. E. Jaffe, and J. T. Leek, “derfinder: Software for annotation-agnostic rna-seq differential expression analysis,” *bioRxiv*, p. 015370, 2015.

[43] P. AC’t Hoen, M. R. Friedländer, J. Almlöf, M. Sammeth, I. Pulyakhina, S. Y. Anvar, J. F. Laros, H. P. Buermans, O. Karlberg, M. Brännvall, *et al.*, “Reproducibility of high-throughput mrna and small rna sequencing across laboratories,” *Nature biotechnology*, 2013.

[44] B. Langmead, C. Trapnell, M. Pop, S. L. Salzberg, *et al.*, “Ultrafast and memory-efficient alignment of short dna sequences to the human genome,” *Genome Biol*, vol. 10, no. 3, p. R25, 2009.

[45] W. J. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and D. Haussler, “The human genome browser at ucsc,” *Genome research*, vol. 12, no. 6, pp. 996–1006, 2002.

[46] “TopHat 2 manual.” http://ccb.jhu.edu/software/tophat/manual.shtml. Accessed: 2014-10-14.

[47] “Amazon elastic compute cloud instance type information.” http://aws.amazon.com/ec2/instance-types/. Accessed: 2014-10-16.

[48] S. B. Montgomery, M. Sammeth, M. Gutierrez-Arcelus, R. P. Lach, C. Ingle, J. Nisbett, R. Guigo, and E. T. Dermitzakis, “Transcriptome genetics using second generation sequencing in a caucasian population,” *Nature*, vol. 464, no. 7289, pp. 773–777, 2010.

[49] J. Harrow, A. Frankish, J. M. Gonzalez, E. Tapanari, M. Diekhans, F. Kokocinski, B. L. Aken, D. Barrell, A. Zadissa, S. Searle, *et al.*, "Gencode: the reference human genome annotation for the encode project," *Genome research*, vol. 22, no. 9, pp. 1760–1774, 2012.

[50] "STAR manual." https://github.com/alexdobin/STAR/blob/master/doc/STARmanual.pdf. Accessed: 2015-03-26.

[51] P. G. Engström, T. Steijger, B. Sipos, G. R. Grant, A. Kahles, G. Rätsch, N. Goldman, T. J. Hubbard, J. Harrow, R. Guigó, *et al.*, "Systematic evaluation of spliced alignment programs for rna-seq data," *Nature methods*, 2013.

[52] L. Collado-Torres, A. Nellore, A. Jaffe, J. Leek, and B. Langmead, "GEUVADIS expressed regions coverage matrix." http://dx.doi.org/10.6084/m9.figshare.1405638.

[53] "Elephant Bird github repository." https://github.com/kevinweil/elephant-bird/. Accessed: 2014-10-14.

[54] M. Burset, I. Seledtsov, and V. Solovyev, "Analysis of canonical and non-canonical splice sites in mammalian genomes," *Nucleic Acids Research*, vol. 28, no. 21, pp. 4364–4375, 2000.

[55] N. Bansal, A. Blum, and S. Chawla, "Correlation clustering," *Machine Learning*, vol. 56, no. 1-3, pp. 89–113, 2004.

[56] N. Ailon, M. Charikar, and A. Newman, "Aggregating inconsistent information: ranking and clustering," *Journal of the ACM (JACM)*, vol. 55, no. 5, p. 23, 2008.

[57] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[58] "Bedgraph format specification." http://genome.ucsc.edu/goldenpath/help/bedgraph.html. Accessed: 2014-10-26.

[59] J. H. Bullard, E. Purdom, K. D. Hansen, and S. Dudoit, "Evaluation of statistical methods for normalization and differential expression in mrna-seq experiments," *BMC bioinformatics*, vol. 11, no. 1, p. 94, 2010.

5 Supplementary material

5.1 GEUVADIS samples

Figure 5 depicts the distribution of read counts across the 666 paired-end GEUVADIS RNA-seq samples. The median number of reads in a sample is 52, 643, 207 with median absolute deviation 12, 526, 395. Details on experimental and sequencing protocols are provided in the supplementary note of [43].

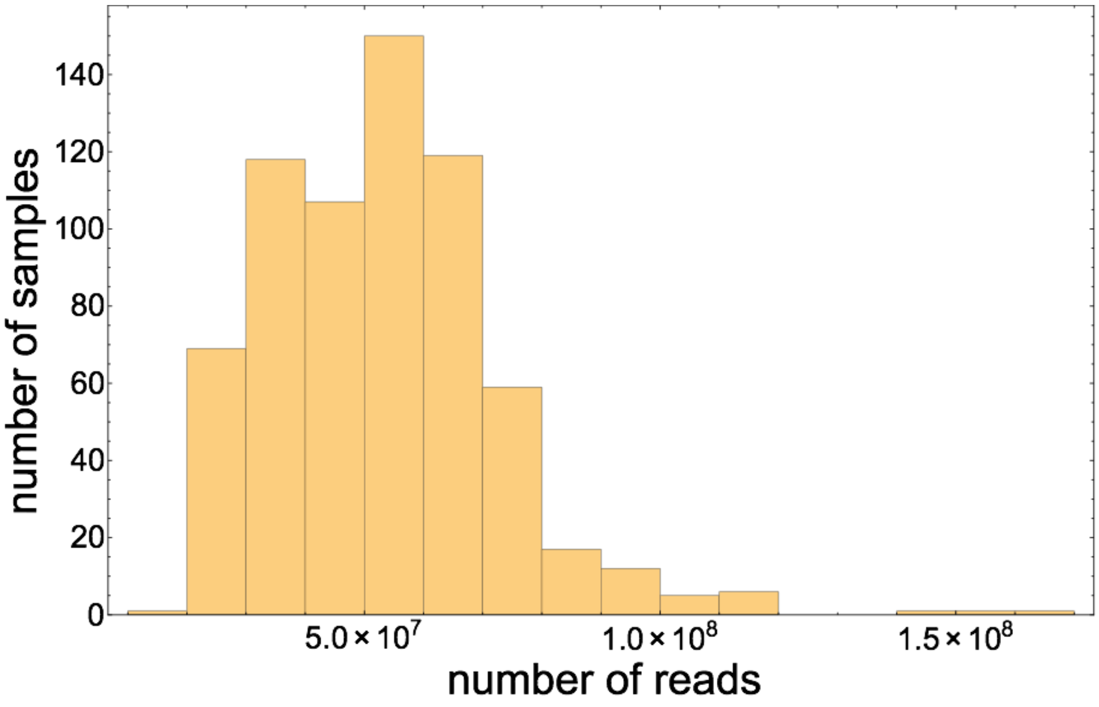


Figure 5: Histogram of read counts in all 666 paired-end GEUVADIS samples.

5.2 Reproducing results

All scripts used to obtain the results in this paper are available at `https://github.com/nellore/rail/tree/master/eval`. Instructions on how to reproduce our results may be found at `https://github.com/nellore/rail/blob/master/eval/README.md`.

Rail-RNA 0.1.0a reproduces results in this paper and has several dependencies. In the experiments we conducted, Rail-RNA wrapped Bowtie 1 v1.1.1, Bowtie 2 v2.2.4, and SAMTools v0.1.19. When run in its elastic mode, Rail-RNA used PyPy v2.2.1. In all other cases, Rail-RNA used PyPy v2.4.0. Version 2.0.12 of TopHat 2 was used and, like Rail-RNA, it wrapped Bowtie 2 v2.2.4 and SAMTools v0.1.19. Version 2.4.0j of STAR was used, and version 0.1.5-beta of HISAT was used. Flux Simulator 1.2.1 was used to obtain simulated samples.

5.3 GEUVADIS subsets

The 28 samples spanned 1,638,479,586 reads, the 56 samples spanned 3,127,479,220, and the 112 samples spanned 6,402,672,808 reads. When we discuss Rail-RNA's scaling with respect to number of samples, we implicitly assume that doubling the number of samples roughly doubles the number of reads. This is approximately true for the GEUVADIS subsets we consider.

5.4 Amazon Elastic Compute Cloud instance specifications

All experiments in this paper were conducted on either c3.2xlarge instances or c3.8xlarge instances. A c3.2xlarge instance is a virtualized computer powered by Intel Xeon E5-2680 v2 (Ivy Bridge) 64-bit processors that provides 15 GB of RAM, 320 GB of solid-state storage, and 8 2.8-Ghz processing cores. A c3.8xlarge instance is a virtualized computer powered by Intel Xeon E5-2680 v2 (Ivy Bridge) processors that provides 60 GB of RAM, 640 GB of solid-state storage, and 32 2.8-Ghz processing cores [47].

5.5 Spot instances

Spot instances permit bidding for EC2 computing capacity. If a bid exceeds the market price, which varies continuously according to supply and demand, requested computing capacity is made available to the user. A risk of using spot instances is that the computing capacity may be lost if market price rises to exceed the original bid price. Market prices vary from region (with an Amazon Web Services data center) to region. In March 2014, when our experiments were run, the market price in the US Standard region was under \$0.07 per hour per c3.2xlarge instance. By contrast, reserving a c3.2xlarge instance in March 2014 in the US Standard region on demand, which ensures no job failure due to fluctuations in market price, cost \$0.42 per hour in March 2014.

5.6 Measuring Amazon Web Services costs

For every experiment in this paper, we measure cost by totaling the bid price for the EC2 spot instances (\$0.11 per instance per hour using spot marketplace) and the Elastic MapReduce surcharges (\$0.105 per instance per hour). On the one hand, this estimate can be low since it does not account for costs incurred by DynamoDB, S3 and other related services. On the other, the estimate can be high since the actual price paid depends on the spot market price which is lower than the bid price.

5.7 Simulated data

The 112 GEUVADIS samples on whose coverage distributions our simulated samples are based are the same as those studied in Section 2.2: all seven labs that pursued the study are represented, with sixteen samples from each lab. RPKMs were computed by Flux Capacitor [48] for transcripts from Gencode v12 [49] after an initial mapping to the GRCh37 (hg19) reference genome using GEM v1.349 [23]. To derive expression profiles for our simulated samples, we multiplied RPKMs by transcript lengths. For each sample, we interrupted the Flux Simulator pipeline after it had generated an expression profile and overwrote the appropriate intermediate file with our computed expression profile. After we resumed the pipeline, Flux Simulator simulated fragmentation, reverse transcription, and sequencing; for each sample, this yielded a FASTQ file with raw reads and a BED file specifying each read's location in the genome. Flux Simulator's built-in 76-base error model was used in the sequencing step. This model is biased towards generating more base substitutions near the low-quality ends of reads: the error rate ranges from 0.01% at the high-quality end to 0.03% at the low-quality end.

The simulated datasets may be regenerated using the script `https://github.com/buci/rail/tree/master/eval/run_all_small_data_sims_locally.sh`.

5.8 Aligner protocols

Commands executed for different alignment protocols are contained in `https://github.com/buci/rail/tree/master/eval/run_all_small_data_sims_locally.sh`. Default command-line parameters were used for all protocols unless otherwise specified. In the "STAR 1 pass" protocol, STAR searches for introns in each individual read and does not realign reads. The "STAR 2 pass" protocol does not build a new index; rather, the protocol involves a single invocation of the STAR command with the parameters

```
--twopass1readsN 50000000 --sjdbOverhang 75
```

, conforming to the protocol from the STAR manual [50]. We found empirically that an older two-pass protocol where

1. an index with the exonic bases surrounding introns compiled from the STAR 1 pass run is built. This index includes not just such transcript fragments, but also independently the full genome;
2. STAR is rerun to realign all reads to this index;

gives almost exactly the same results. The older protocol is published in the supplement of the RGASP paper [51], which compares the accuracies of several spliced alignment protocols. Otherwise, the STAR protocols used here mirror those used for the RGASP paper.

In the "HISAT 1 pass" protocol, HISAT searches for introns in each read while also accumulating a list of introns on the fly for which to additionally search in reads. In "HISAT 2 pass" protocol,

1. HISAT was run in a mode where novel splice sites were output to a file using its `--novel-splicesite-outfile` parameter.
2. HISAT was rerun on all reads with as extra input the novel splice sites found in 1. using the `--novel-splicesite-infile` parameter.

5.9 Accuracy metrics

Overlap accuracy. Define a *true overlap* as the event that a read's true alignment from simulation overlaps an intron. Define a *positive overlap* as the event that a read's primary alignment as reported by a given spliced alignment program overlaps an intron. A *true positive* is a case where an intron overlapped by the true alignment of a read is also overlapped by the reported primary alignment of the read. Precision is

$$\text{precision} = \frac{\#\text{ true positives}}{\#\text{ positive overlaps}} .$$

Recall is

$$\text{recall} = \frac{\#\text{ true positives}}{\#\text{ true overlaps}} .$$

Intron accuracy. Let R be the set of introns overlapped by at least one primary alignment reported by the aligner. Let T be the set of introns overlapped by at least one simulated read. Precision is

$$\text{precision} = \frac{|R \cap T|}{|R|}$$

and recall is

$$\text{recall} = \frac{|R \cap T|}{|T|}\,.$$

The F-score is the harmonic mean of precision and recall:

$$\text{F-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}\,.$$

5.10 Intron coverage bias

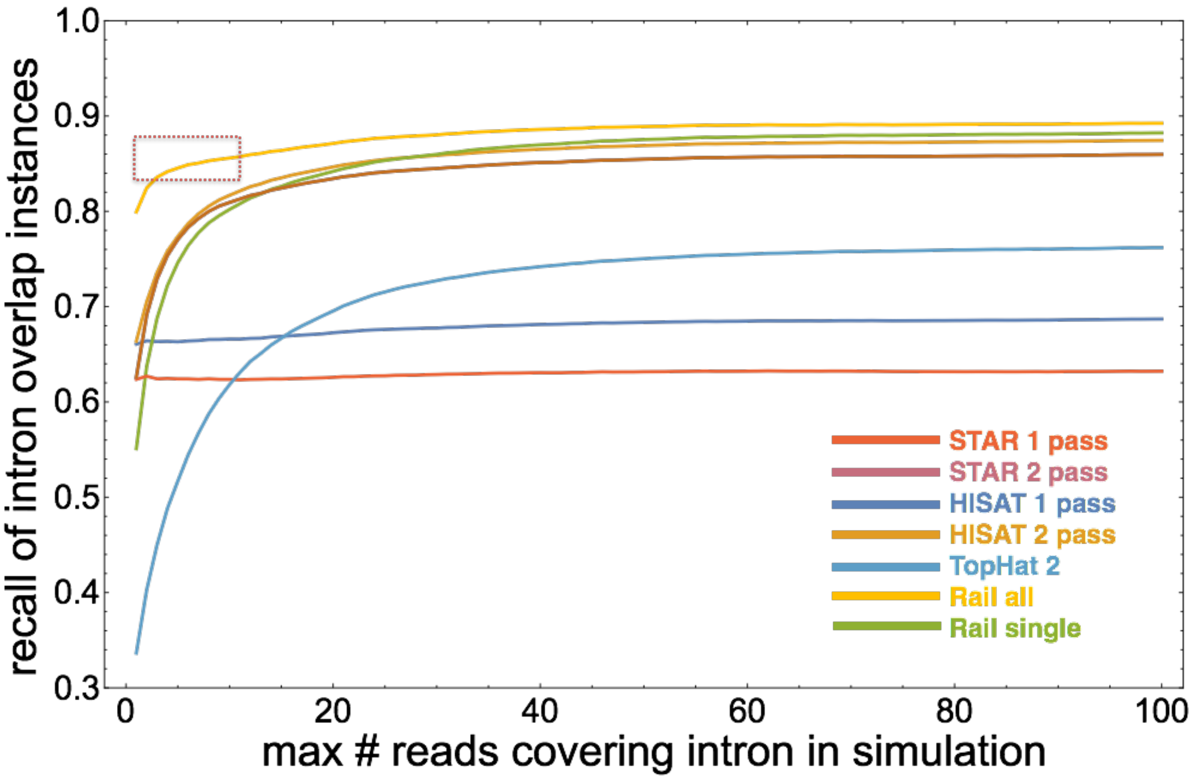


Figure 6: The "Rail all" protocol (yellow line at the top) achieves high recall without succumbing significantly to coverage bias where introns are covered by few reads (red dotted box), unlike 2-pass protocols single-sample protocols "STAR 2 pass," "HISAT 2 pass," TopHat 2, and "Rail single." 1-pass protocols are not subject to this intron coverage bias but do not achieve high recall.

Let $c(r)$ denote the true coverage of the minimally covered intron overlapped by a read $\mathbf{r}$ in a sample s. Here, "true coverage" refers to the number of reads truly derived from loci overlapping a given intron. For example, if $\mathbf{r}$ overlaps two introns i_1 and i_2, and i_1 is truly covered by 10 reads

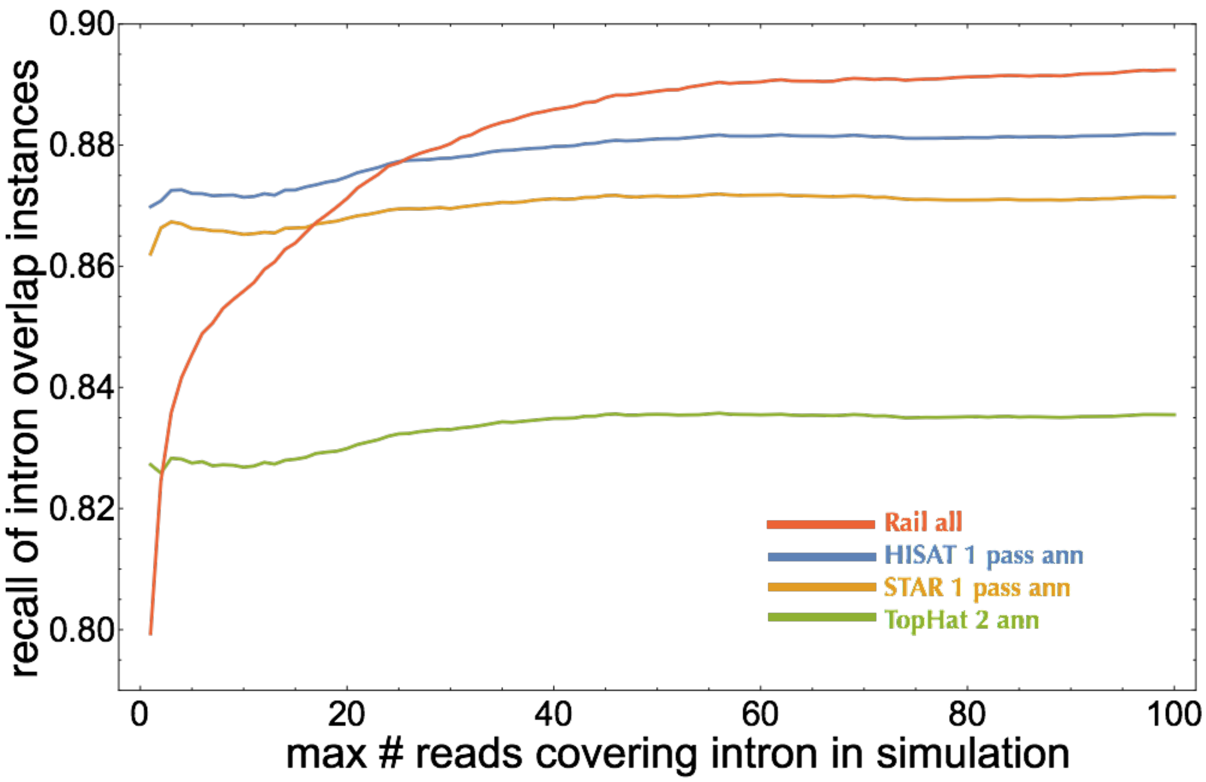


Figure 7: The "Rail all" protocol (red curved line) does not appear to correct for intron coverage bias as well as annotated protocols, but annotated protocols have an artificial advantage: they are given as additional input every true intron sampled in simulation.

in S from s while i_2 is truly covered by 25 reads in S, $c(r) = 10$. Figure 6 plots recall for various protocols that do not use annotation at various thresholds t of $c(r)$; a given point on any curve accounts for every read $\mathbf{r}$ for which $c(r) \leq t$. This plot reveals a bias present in certain spliced alignment protocols that involve realignment (i.e., the 2 pass protocols): the correct alignments of reads that overlap introns whose true coverage is low are recalled less often. This intron coverage bias has a simple explanation: if there are more reads covering a given intron, it is likelier that intron is detected initially (on the first pass) by an alignment protocol. So during realignment, it is likelier that splice junctions are detected successfully for reads overlapping highly covered introns than it is for poorly covered introns. By contrast, if no realignment is performed, recall should be independent of $c(r)$: the probability a splice junction is detected for a given read is independent of the probabilities splice junctions are detected in other reads. Neither "STAR 1 pass" nor "HISAT 1 pass" perform realignment, so their recalls are more or less horizontal lines across the plot. However, recall is diminished compared to protocols that do perform realignment. (HISAT achieves higher recall than STAR on one pass because it accumulates an intron list as it aligns a sample, searching for a larger number of introns as more reads are aligned. But the aligner does not have a large list of introns at its disposal when it is working through the first reads, so one pass still cannot match the recall of two passes.) Figure 7 plots recall for various protocols that use annotation as well as the "Rail all" protocol at various thresholds t of $c(r)$. "STAR 1-pass ann," "HISAT 1-pass ann," and "TopHat 2 ann" do indeed provide high recall while minimizing intron coverage bias; an annotation can reveal the locations of poorly covered introns. *However, annotation-based protocols are themselves subject to a bias*: it is likelier that introns listed in the annotation are found than it is that novel introns are. This bias is not discernible in Figure 7. Moreover, because the annotated protocols tested here are given the artificial advantage of foreknowledge of the entire space of introns sampled by the simulated samples, Figure 7 exaggerates the positive effect of annotation. "Rail all" achieves significant intron coverage bias mitigation, however, faring nearly as well as even the annotated protocols: note that recall on the y-axis never drops below 0.79.

The "Rail all" protocol is immune to annotation bias while minimizing intron coverage bias and simultaneously providing high recall through realignment. The protocol benefits from how there is a higher probability any given intron–even one that is poorly covered across samples–is detected. When many samples are analyzed together, the intron coverage bias is greatly diminished.

5.11 Expressed Regions analysis

For each sample, base level coverage is imported from the bigWig files and then adjusted by the total number of mapped reads to represent the coverage from a library size of 80 million reads. Expressed regions (ERs) are then determined by the `regionMatrix()` function from `derfinder` (version 1.0.10) [42] applied to the adjusted coverage with a mean cutoff of 5 reads. ERs are identified for each chromosome separately and then merged with a custom script. Overlap with Ensembl v75 [39] known exons, introns and intergenic regions is determined with the corresponding genomic state object via `makeGenomicState()` and `annotateRegions()` from `derfinder`.

`regionMatrix()` also generates the corresponding adjusted coverage matrix for library size of 80 million reads assuming a read length of 75 base pairs. This matrix is $\log_2$ transformed using an offset of 1 and is available via Figshare [52]. A linear regression is then fitted with the 15 technical variables available: population, RIN value, RNA extraction batch, RNA concentration, RNA quantity used, library preparation date, primer index, method concentration measure, library concentration, library size, library concentration used, cluster kit, sequencing kit, cluster density, and sequencing lane. The percent of variance explained is determined by the residual sum of squares for the given variable divided by the total sum of squares from all variables as well as the residual variation.

The GEUVADIS expressed regions are available in `supplementaryExpressedRegions.csv`. This CSV file has the following columns:

1. seqnames: chromosome where the ER is located.
2. start: extreme left position of the ER.
3. end: extreme right position of the ER.
4. width: width of the ER in base pairs.
5. strand: strand of the ER.
6. value: mean adjusted coverage for the ER across all samples.
7. area: sum of adjusted coverage for the ER across all samples.

8. indexStart: internal index start position.

9. indexEnd: internal index end position.

10. cluster: cluster ID for the ER where ERs belong to the same cluster if they are at most 3000 base pairs apart. IDs are unique by chromosome only.

11. clusterL: cluster length in base pairs.

5.12 Detail: Preprocess reads

In local and parallel modes, workers divide the reads for each sample (typically stored in one FASTQ or a pair of FASTQs) among several gzip-compressed files. Splitting a sample's reads up in this manner enables their distribution among workers in the next step of the pipeline. In elastic mode, the reads for each sample are written to a single LZO-compressed file that may subsequently be indexed for the same purpose as file-splitting in local and parallel modes: this also permits fast distribution of file splits among workers in the next step. The distribution is facilitated by Twitter's Elephant Bird library [53] for Hadoop.

In local and parallel modes, if all input reads are on a locally accessible filesystem, the preprocess map step described in the main text is preceded by two load-balancing steps. In the first—itself a map step—each worker operates on a sample at a time, counting the number of reads. Only one worker is active in a step that follows, collecting read counts across samples and deciding how many and which reads should be processed by a given worker at a time in the preprocess map step. This ensures that load is distributed evenly across workers during preprocessing. In elastic mode or if some samples must be downloaded from the Internet, a worker operates on a sample at a time in the preprocess map step. This permits deletion of a given downloaded set of reads by a worker as soon as it is preprocessed, saving storage space.

5.13 Detail: Align reads to genome

Reads are partitioned by nucleotide sequence so that each worker can operate on a set of reads that share the same sequence at once. A task is composed of several such sets; to reduce redundancy in analysis, a worker nominates from each set a read with the highest mean quality score in that set for alignment using Bowtie 2 [33] in its "local" mode. (To avoid redundancy between sequences and reversed complements, a read's nucleotide sequence is taken to be either the original or its reversed complement, whichever is first in lexicographic order.) Call the set of these nominated reads A_1. If a read in A_1 has exactly one perfect alignment, we assume it does not overlap any introns and place it *and all other reads with the same nucleotide sequence* in set F; that is, the alignment found is assigned to all reads that share the same nucleotide sequence. If a read has more than one perfect alignment, we also assume the read does not overlap any introns and place all it in set F; however, we place other reads with the same nucleotide sequence in set A_2. Reads in A_2 are aligned on a second run of Bowtie 2 performed in-step by the same worker. This lets Bowtie 2 break ties in their alignment scores to select primary alignments. Alignments of reads in F are final and appear in Rail-RNA's terminal output; they attain the highest possible alignment score and overlap no introns.

Bowtie 2's local mode soft-clips the ends of alignments if doing so improves local alignment score. We exploit this feature to determine which read sequences should be probed for overlapped introns. If the primary alignment of any read in A_1 has at least one edit or is soft-clipped, we add the read to a set C, place its associated nucleotide sequence in set S, and add all other reads with the same nucleotide sequence to A_2. Note that S is composed of unique read sequences, while A_1, A_2, and F may contain reads with the same sequence. Any read whose nucleotide sequence is in S is realigned in a later step. Set S is further divided into subset S_{search} and its complement $S_{\text{no search}}$. If the primary alignment of a read in A_1 lacks a soft-clipped end spanning at least some user-specified number of bases (by default 1), its nucleotide sequence is added to $S_{\text{no search}}$. Otherwise, the sequence is added to S_{search}. The parameter *min_exon_size* may be configured by the user and is 9 by default. Here, the idea is that a soft-clipped end has to be relatively long to be uniquely mappable to a site tens to hundreds of thousands of positions away from the alignment reported by Bowtie 2. In subsequent steps, read sequences from S_{search} are searched for introns and realigned to references containing transcript fragments—contiguous exonic base sequences from the genome. Reads whose nucleotide sequences are in $S_{\text{no search}}$ are never searched for introns but are also realigned to transcript fragments.

Primary alignments of reads in A_2 are found on a second run of Bowtie 2 in local mode. If a primary alignment is perfect, its associated read is promoted to F; if it contains at least one edit, the read is placed in C. Alignments of reads in C are saved for comparison with realignments of these reads to transcript fragments in later steps.

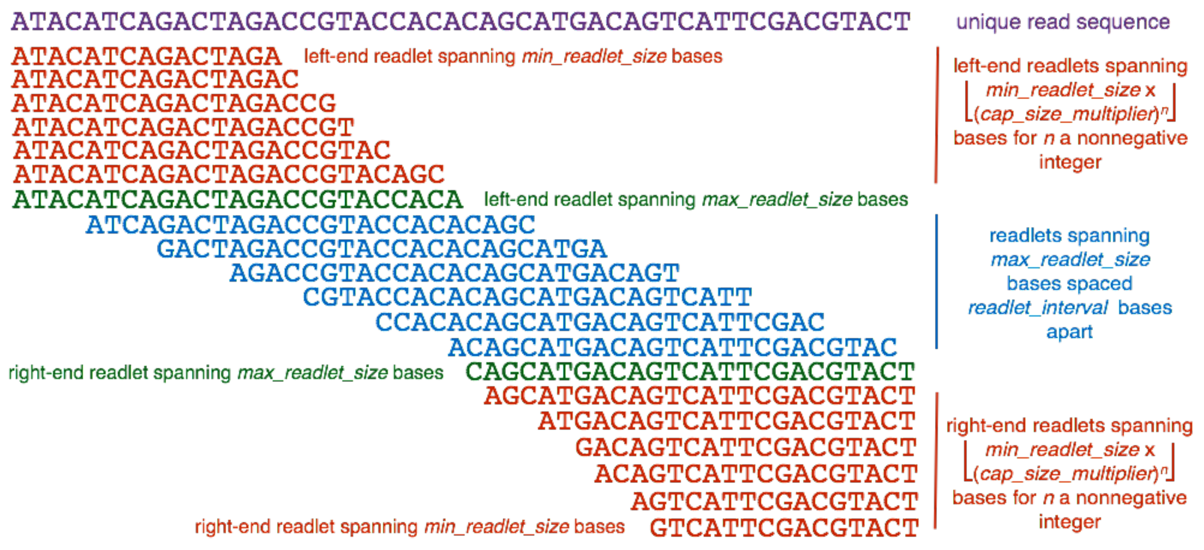

ATACATCAGACTAGACCGTACCACACAGCATGACAGTCATTCGACGTACT
unique read sequence
ATACATCAGACTAGA
left-end readlet spanning min_readlet_size bases
ATACATCAGACTAGAC
ATACATCAGACTAGACCG
ATACATCAGACTAGACCGT
ATACATCAGACTAGACCGTAC
ATACATCAGACTAGACCGTACAGC
ATACATCAGACTAGACCGTACCACA
left-end readlet spanning max_readlet_size bases
left-end readlets spanning ⌊min_readlet_size x (cap_size_multiplier)^n⌋ bases for n a nonnegative integer
ATCAGACTAGACCGTACCACACAGC
GACTAGACCGTACCACACAGCATGA
AGACCGTACCACACAGCATGACAGT
CGTACCACACAGCATGACAGTCATT
CCACACAGCATGACAGTCATTCGAC
ACAGCATGACAGTCATTCGACGTAC
readlets spanning max_readlet_size bases spaced readlet_interval bases apart
right-end readlet spanning max_readlet_size bases
CAGCATGACAGTCATTCGACGTACT
AGCATGACAGTCATTCGACGTACT
ATGACAGTCATTCGACGTACT
GACAGTCATTCGACGTACT
ACAGTCATTCGACGTACT
AGTCATTCGACGTACT
right-end readlet spanning min_readlet_size bases
GTCATTCGACGTACT
right-end readlets spanning ⌊min_readlet_size x (cap_size_multiplier)^n⌋ bases for n a nonnegative integer

Figure 8: Rail-RNA's scheme for readletizing (segmenting) a read sequence into short subsequences for alignment to the genome to infer intron positions. In this example, *min_readlet_size* is 15, *max_readlet_size* is 25, *cap_size_multiplier* is 1.1, and *readlet_interval* is 4.

Every unique read sequence in S is hashed to place it one of $P \times N_s$ "index" bins, where N_s is the number of samples under analysis in the Rail-RNA run and P is the number of FM indexes that will be built per sample in a future realignment step. More specifically, all reads whose sequences are in the same bin are aligned to the same index in the step described in Section 3.9. Every unique read sequence in S is also "readletized": it is divided into short overlapping segments, called readlets. In the next step, readlets are aligned to the genome so any introns that occur between alignments of successive nonoverlapping readlets to the same contig can be inferred. **Information about in**

which samples each read sequence occurs is passed on to subsequent steps.

The readletizing scheme is detailed in Figure 8. There is always a readlet at each end of the read spanning *max_readlet_size* bases. Intervening readlets spanning *max_readlet_size* bases are spaced *readlet_interval* bases apart. There are also smaller readlets ("capping readlets") at each end of the read. Their sizes are spanned by the set

$$\left\{s : \exists n \in \mathbf{N}^0 \;\; \left(s = \lfloor \mathit{min\_readlet\_size} \times (\mathit{cap\_size\_multiplier})^n \rfloor \wedge s < \mathit{max\_readlet\_size}\right)\right\} .$$

Parameters from the previous paragraph are positive integers constrained as follows.

$$\begin{aligned} \mathit{min\_readlet\_size} &\geq 4 \\ \mathit{max\_readlet\_size} &\geq \mathit{min\_readlet\_size} \\ \mathit{cap\_size\_multiplier} &> 1 \\ \mathit{readlet\_interval} &\geq 1 \,. \end{aligned}$$

Their default values of, respectively, 15, 25, 1.1, and 4 may be toggled by the user.

5.14 Detail: Search for introns using readlet alignments

To explain Rail-RNA's algorithm for searching for introns, we restrict our attention to a single read $\mathbf{r}$ and its readlets.[2] To be precise in our discussion, we use the (slightly redundant) notation $(\mathbf{d}, p_{\mathbf{r}}, s, p_s, \ell, k)$ to denote the kth *exact match to the reference* of a readlet $\mathbf{d}$ spanning l bases that is displaced $p_{\mathbf{r}}$ bases away from the left end of $\mathbf{r}$; the alignment is to a position displaced p_s bases away from the left end of strand s of the reference assembly. When two readlet alignments $(\mathbf{d}_1, p_{\mathbf{r},1}, s_1, p_{s_1,1}, \ell_1, k_1)\,, (\mathbf{d}_2, p_{\mathbf{r}}, s_2, p_{s_2,2}, \ell_2, k_2)$ are *compatible*:

1. $\mathbf{d}_1 \neq \mathbf{d}_2$; the alignments are not of the same readlet.

2. $s_1 = s_2 = s$ for some strand s; the two readlets align to the same strand.

3. $p_{\mathbf{r},1} < p_{\mathbf{r},2} \iff p_{s,1} < p_{s,2}$; if their positions along the read are different, the readlets are ordered in the same way along the read as their alignments along s.

4. $p_{\mathbf{r},1} = p_{\mathbf{r},2} \iff p_{s,1} = p_{s,2}$; if their positions along the read are the same, the readlets align to the same position along s.

Ideally, the readlet alignments derived from $\mathbf{r}$ are mutually compatible, as depicted in the top half of Figure 9: all of read 1's mapped readlets align uniquely to the same strand (here, the forward strand of *chr1*) in the order in which they appear along the read; and further, if two readlets begin at the same position along the read, they align to the same position along the reference. Suppose $i \in [N]$ indexes a set $R = \{(\mathbf{d}_i, p_{\mathbf{r},i}, s, p_{s,i}, \ell_i, k_i)\}$ of mutually compatible readlet alignments. Note that each alignment in R corresponds to a distinct readlet. Let $S(i) = \{j : \exists j \in [N]\; p_{\mathbf{r},i} = p_{\mathbf{r},j}\}$; that is, $S(i)$ indexes the set of readlets in R whose displacements from the left end of $\mathbf{r}$ are the same as $\mathbf{d}_i$'s. When two readlet alignments $(\mathbf{d}_m, p_{\mathbf{r},m}, s, p_{s,1}, \ell_m)\,, (\mathbf{d}_q, p_{\mathbf{r},q}, s, p_{s,q}, \ell_2) \in R$ are *consecutive*:

1. $p_{\mathbf{r},m} \neq p_{\mathbf{r},q}$; the readlets do not occur at the same position along the read.

[2]Here, a read is really a unique read sequence. In this discussion, we take the "left end of a read" to mean the left end of either the sequence or its reversed complement, whichever is first in lexicographic order.

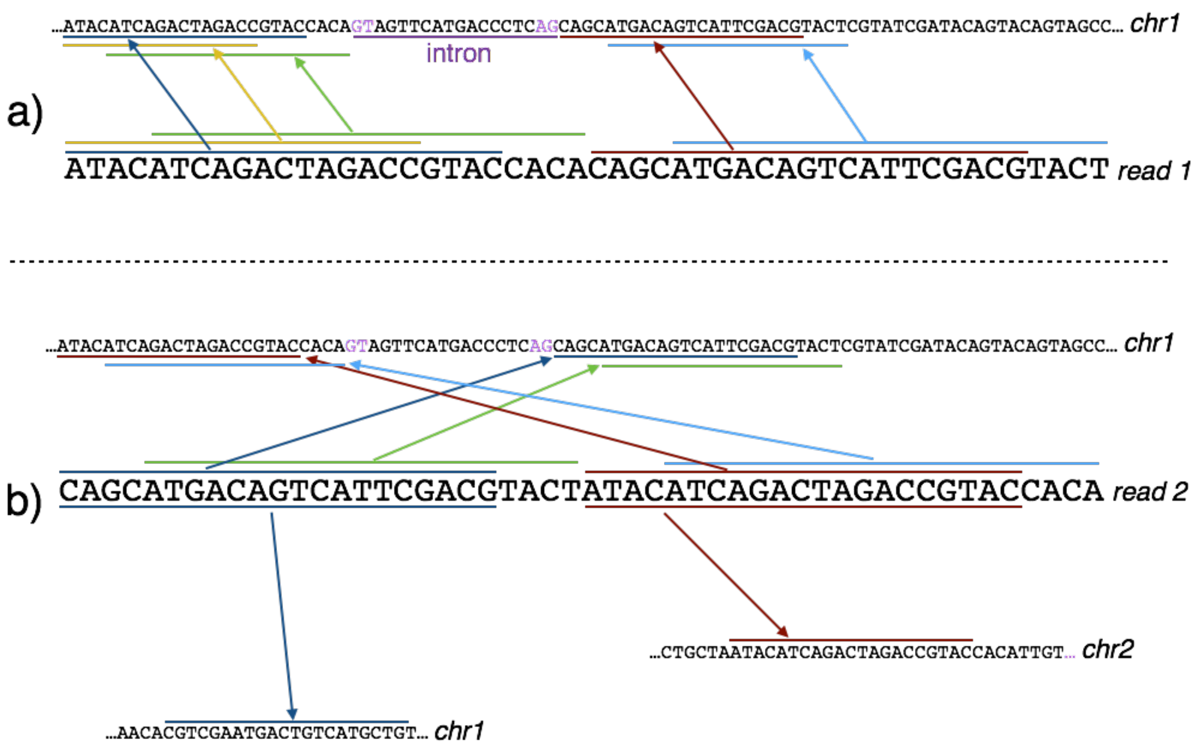

...ATACATCAGACTAGACCGTACCACAGTAGTTCATGACCCTCAGCAGCATGACAGTCATTCGACGTACTCGTATCGATACAGTACAGTAGCC... chr1
intron
a)
ATACATCAGACTAGACCGTACCACACAGCATGACAGTCATTCGACGTACT read 1
...ATACATCAGACTAGACCGTACCACAGTAGTTCATGACCCTCAGCAGCATGACAGTCATTCGACGTACTCGTATCGATACAGTACAGTAGCC... chr1
b)
CAGCATGACAGTCATTCGACGTACTATACATCAGACTAGACCGTACCACA read 2
...CTGCTAATACATCAGACTAGACCGTACCACATTGT... chr2
...AACACGTCGAATGACTGTCATGCTGT... chr1

Figure 9: a) Ideally, every readlet of a read aligns uniquely to the same strand of a contig (e.g., the forward strand of chr1). b) Often, the ideal case depicted in a) is not realized and one or more of the following occurs: readlets do not align in the order in which they appear along the read (top); readlets align to multiple strand (the readlets that are underlined and overlined with arrows pointing to their alignments); and readlets align in the right order along the same contig but to different strands. (Note that the readlet mapping at the bottom left of the figure illustrates the reversed complement of a readlet aligning to the forward strand.)

2. $\left(\ell_m \in \operatorname{argmax}_{j \in S(m)} \ell_j\right) \wedge \left(\ell_q \in \operatorname{argmax}_{j \in S(q)} \ell_j\right)$ holds; each readlet is the longest mapped readlet occurring at its respective position along the read.

3. $(p_{\mathbf{r},m} < p_{\mathbf{r},q} \implies \{i : \exists i \in [N]\ p_{\mathbf{r},m} < p_{\mathbf{r},i} < p_{\mathbf{r},q}\} = \varnothing)$
   $\wedge\ (p_{\mathbf{r},q} < p_{\mathbf{r},m} \implies \{i : \exists i \in [N]\ p_{\mathbf{r},q} < p_{\mathbf{r},i} < p_{\mathbf{r},m}\} = \varnothing)$ holds; there is no mapped readlet between $\mathbf{d}_m$ and $\mathbf{d}_q$ exclusive.

An example of two consecutive readlet alignments is provided by the subsequences overlined and underlined in green and brown in the top half of Figure 9. These alignments also illustrate one way an intron overlapped by a read is recovered by Rail-RNA: when two readlets align consecutively to the reference on either side of exactly one intron, its presence is inferred by searching for characteristic two-base motifs denoting its donor and acceptor sites. In order of descending prevalence in mammalian genomes according to a survey of GenBank annotated genes [54], the (donor, acceptor) site pairs Rail-RNA recognizes are (GT, AG), (GC, AG), and (AT, AC). Such sites are reproduced in the reference sequence if the sense strand is the forward strand; the (GT, AG) combination appears in Figure 9. If the sense strand is the reverse strand, the (acceptor, donor) signals may still be read off the reference from left to right as (CT, AC), (CT, GC), and (GT, AT).

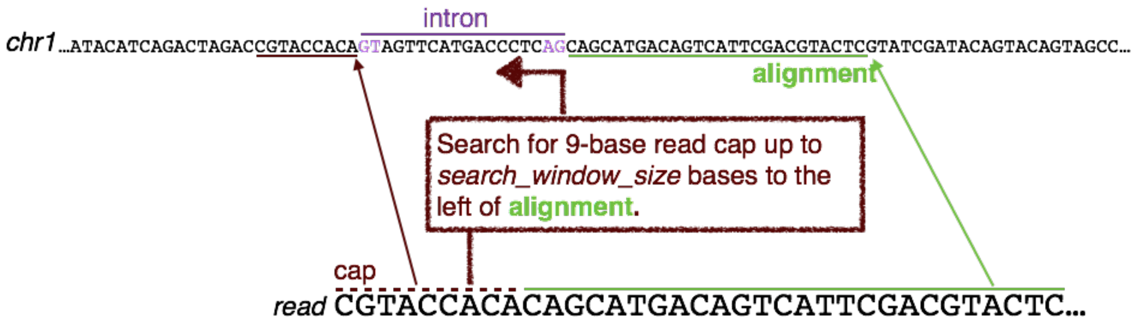

intron
chr1 ...ATACATCAGACTAGACCGTACCACAGTAGTTCATGACCCTCAGCAGCATGACAGTCATTCGACGTACTCGTATCGATACAGTACAGTAGCC...
alignment
Search for 9-base read cap up to search_window_size bases to the left of alignment.
cap
read CGTACCACACAGCATGACAGTCATTCGACGTACTC...

Figure 10: Rail-RNA searches for a match to the read cap that overlaps no aligned readlets if the cap spans at least *min_exon_size* bases, where *min_exon_size* is by default 9.

Figure 10 displays a case where a read has an unmapped 9-base "cap" to the left of a readlet aligned by Bowtie in R: the nine bases do not belong to any mapped readlet. In this event, Rail-RNA searches the reference *search_window_size* bases upstream of the leftmost readlet alignment for maximum matching prefixes of the cap. By default, *search_window_size* is 1000. If a prefix found spans fewer than *min_exon_size* bases, it is ignored. by default, *min_exon_size* is 9. Otherwise, Rail-RNA treats the maximum matching prefix closest to the leftmost readlet alignment as a readlet alignment itself. This procedure is mirrored if there is a cap at the right end of the read, where the search is then downstream of the rightmost aligning readlet. So the set of mutually compatible readlet alignments R may be augmented by up to two caps.

Assuming a pair of consecutive readlet alignments is correct and that only one intron lies between them, they uniquely determine the length L of the intron along the reference in the absence of intervening indels whose net length os nonzero: for consecutive readlets $(\mathbf{d}_m, p_{\mathbf{r},m}, s, p_{s,m}, \ell_m), (\mathbf{d}_q, p_{\mathbf{r},q}, s, p_{s,q}, \ell_q) \in R$,

$$L = |p_{s,q} - p_{s,m}| - \ell_m \, .$$

If more than one intron lies between two consecutive readlet alignments, L as defined above is the *sum* of the lengths of the introns. For every pair of consecutive readlet alignments in R, Rail-RNA searches for up to two intervening introns with valid (donor, acceptor) motifs that satisfy the constraint on L. When searching for two introns, Rail-RNA requires that the size of the exon between them is above an adjustable parameter *min_exon_size*. Denote as F the region of the reference framed by the two consecutive readlets. Search windows are confined to the *search_window_size* bases at either end of F. Several candidate introns or intron pairs may be uncovered within F and in the vicinity of its ends. Each candidate is placed in one of three preference classes according to (donor, acceptor) signal: (GT, AG) is preferred to (GC, AG), which is preferred to (AT, AC). Rail-RNA also ranks each candidate within its respective class: its score is determined from global alignment of the read subsequence overlapped by the two consecutive readlet alignments to the exonic bases surrounding the candidate intron or intron pair. Here, we assign +1 for each matched base and -1 for each single-base gap and mismatched base. Rail-RNA selects candidate introns and intron pairs with the highest score in the most-preferred class with at least one alignment. For every intron, a (key, value) pair is written for each sample in which $\mathbf{r}$ occurs. More specifically, the output of this step is:

key: strand of origin of intron, intron start position, intron end position
value: a list of tuples $[(i, \beta_i)]$, where i indexes a sample and β_i is the number of reads in the sample with the searched read sequence in which the intron was found.

The discussion in this section is so far predicated on how all the readlet alignments derived from a read $\mathbf{r}$ are mutually compatible, forming a valid set R. This ideal situation is not always realized. The bottom half of Figure 9 displays how readlet alignments may be incompatible: readlets may not align to the same strand in the same order in which they appear along the read, or a readlet may have more than one alignment. In such cases, Rail-RNA selects a "consensus" set of mutually compatible alignments from all the readlet alignments as follows. Consider the set $V = \{(\mathbf{d}_i, p_{\mathbf{r},i}, s, p_{s,i}, \ell_i, k_i)\}$ of all alignments of all readlets reported by Bowtie. Form a complete signed graph $\mathcal{G} = (V, E)$ whose vertices correspond to readlet alignments:

1. Start with the vertices disconnected.
2. Place a "+" edge between every pair of alignments that are the closest compatible alignments of their corresponding readlets. Here, "closest" means "having the smallest number of bases between their start positions along the reference."
3. Complete the graph with "−" edges.

Correlation clustering finds the clustering of vertices that maximizes the number of agreements—that is, the sum of the number of "+" edges within clusters and the number of "−" edges between clusters. A consensus set of compatible alignments may be derived from the largest such cluster. However, correlation clustering is an NP-hard problem [55], so we use a randomized 3-approximation algorithm introduced in [56]:

```
Algorithm 1: KwikCluster
    Input  : a signed graph $\mathcal{G} = (V, E)$ whose vertices represent readlet alignments
    Output: a set of clusters $\{C_i\}$ of readlet alignments
    $V' \leftarrow V$;
    $i \leftarrow 0$;
    while $V' \neq \varnothing$ do
        $C_i \leftarrow \varnothing$;
        $\mathbf{d}_p \leftarrow$ randomly selected pivot alignment from $V'$;
        Move $\mathbf{d}_p$ from $V'$ to $C_i$;
        Move all alignments from $V'$ that share a "+" edge with $\mathbf{d}_p$ on $\mathcal{G}$ to $C_i$;
        $i \leftarrow i + 1$;
```

After this procedure, the pivot alignment in each cluster C_i is compatible with every other alignment in that cluster. However, it is not guaranteed that all the alignments in a given C_i are mutually compatible. So Rail-RNA sorts the clusters in order of descending size and loops through it: it first forms a unweighted graph $\mathcal{G}_i$ for a given C_i, placing an edge between two alignments if and only if they are compatible. The Bron-Kerbosch algorithm [57] is then used to obtain a maximum clique $\mathcal{C}_i$ for C_i. (If the algorithm finds more than one maximum clique for a given cluster, the maximum clique is chosen at random.) If $\mathcal{C}_i$ spans more nodes than C_{i+1}—the next cluster from the sorted list—or if C_i is the last cluster from the list, *and if there are no cliques of* *the same size among those computed so far*, Rail-RNA selects $\mathcal{C}_i$ as the set of mutually compatible readlet alignments from which it infers introns. If there are multiple cliques of the same size, the software does not search the read sequence for introns. So when there are no such ties, Rail-RNA selects the largest clique from among the $\{C_i\}$. While the time complexity of the Bron-Kerbosch algorithm is exponential in the number of graph nodes, it executes quickly in practice: 1) it is run only on each cluster of readlet alignments where at least one alignment is compatible with other alignments from the cluster, so there are few nodes; and 2) the number of possible of alignments Rail-RNA studies for a given readlet is limited by Bowtie's `-m` parameter; as mentioned above, no readlet alignments are reported when more than by default 30 alignments are uncovered.

5.15 Detail: Enumerate intron configurations along read segments

Intron configurations are obtained as follows. A directed acyclic graph (DAG) is constructed for every combination of sample and strand for which introns were detected. Each vertex of the DAG represents a unique intron $k_i = (s_i, e_i)$, where s_i is the coordinate of the first base of the intron, and e_i is the coordinate of the first base that follows the intron. An edge occurs between two introns k_1 and k_2 if and only if they do not overlap—that is, the coordinates spanned by the introns are mutually exclusive—and no intron k_3 occurs between k_1 and k_2 such that k_1, k_2, and k_3 do not overlap. An edge always extends from an intron with smaller coordinates to an intron with larger coordinates. Each edge is weighted by the number of exonic bases between the introns it connects. Figure 11 depicts a portion of an example DAG.

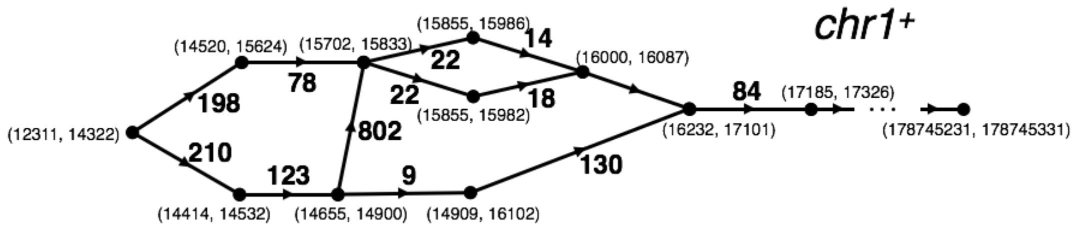


Figure 11: An example DAG. Vertices correspond to introns and are labeled by their start and end positions. An edge extends from one intron to another that comes after it along the strand. Every edge is weighted by the number of exonic bases between the introns it connects.

The paths through the DAG span all possible combinations of nonoverlapping introns along the strand for a given sample. Finding all subpaths (sequences of introns), each of whose weights is less than *readlet_config_size*, enumerates all possible combinations of introns a read segment $\mathbf{s}$(*readlet_config_size*) can overlap. In fact, the parameter *readlet_config_size* could theoretically be set to equal the maximum length of all reads across samples. Then, after concatenating and indexing the exonic sequences surrounding intron configurations to form transcript fragments and indexing, Bowtie 2 could realign entire reads to fragments end-to-end to immediately obtain candidate spliced alignments. Unfortunately, finding intron combinations in "hot spots," where there are many alternative splicings and short exons, can become computationally intractable for large *readlet_config_size*. Even taking *readlet_config_size* $= 50$ can become challenging in later steps, substantially increasing the time it takes to build a Bowtie 2 index of transcript fragments. This explains our choice *readlet_config_size* $= 35$: it's large enough so Bowtie 2 can successfully identify which read sequences are derived from which transcript fragments, soft-clipping as necessary, but also small enough to make intron configuration enumeration and subsequent index construction manageable.

A worker handles one given strand for a given sample at a time. Enumerating combinations of introns that could cooccur on read segments proceeds for each (sample, strand) combination. Rail-RNA uses a streaming algorithm that alternates between **construction** and **consumption** the DAG. Only a portion of the DAG is in memory at a time. Call this portion the mDAG. For a given worker, the input to the construction-consumption algorithm is a sorted list of introns detected by Rail-RNA in a given sample along a given strand, where the sort key is intron start position. The construction operation reads a chunk of this list and augments the mDAG. A subsequent consumption operation erases a part of the mDAG after the intron combinations for that part have been enumerated. Construction and consumption operations are described in detail below.

Construction: The edges of the DAG are output for consumption in an order consistent with a topological ordering of its vertices. In other words, an edge is not generated until every edge for which its source vertex is a sink vertex has already been generated. As mentioned above, the construction subroutine operates on a streamed list of intron positions $[\{(s_i, e_i)\}]$ sorted in order of ascending s_i, as provided by a previous aggregation step. Two data structures are needed to encode the DAG as it is constructed: the set U, containing intron vertices that do not yet have any children, and L, a dictionary that, where possible, maps each intron (the key) to its corresponding successive nonoverlapping intron *with the smallest end position read so far* (the value). **For each** new intron $k_i = (s_i, e_i)$ read:

1. The vertices in U and L are checked for whether k_i is their child—that is, whether vertices previously read are connected to k_i—and corresponding edges are **yield**ed. As edges are yielded, vertices from U may be promoted to L.
2. Each "value" vertex k_j in L is replaced with k_i if $e_i < e_j$.
3. Every vertex k_j in L is checked for whether its corresponding value is itself a key. If this condition is satisfied, an edge can *never* connect k_j with any introns streamed later, and k_j is removed from L.

Rail-RNA generates the DAG $10,000,000$ bases at a time. After a portion of the DAG is produced, all possible intron combinations that can be overlapped by a read segment $\mathbf{s}(\textit{readlet\_config\_size})$ are **output for processing in the next steps.** Rail-RNA accomplishes this by considering each vertex separately and finding the ways its associated intron can be the first intron on $\mathbf{s}$; that is, it walks every path starting from that vertex edge by edge until its weight exceeds *readlet_config_size*.

Consumption: The mDAG is dynamic: edges and vertices may have previously been consumed so that there are new source vertices and sink vertices. A source vertex k is removed from the mDAG when all paths originating at every child vertex k_i of k have been reviewed to find intron configurations that can be overlapped by *readlet_config_size* exonic bases. k can be removed at this time because it is no longer needed to find by how many exonic bases to the left of k_i a transcript fragment can possibly extend before running into an intron. More specifically, an edge from k to k_i is removed if and only if:

1. All edges with k_i as the source have been generated. By construction, this occurs if k_i has at least one grandchild.

2. Every path of maximal length originating at $k_i = (s_i, e_i)$ whose weight is less than $(readlet\_config\_size - 1)$ can be constructed. Suppose $\{(s_j, e_j)\}$ are the children of k_i, and let $\{s, e\}$ be the as-yet-unconsumed vertex with the largest s from the dynamic DAG. This occurs if for every j, $s - e_j \geq readlet\_config\_size - 1 - (s_i - e_j)$.

The alternation continues until the end of the strand is reached. At this point, the edges connecting remaining vertices in U and their new sinks are **yield**ed, and **remaining intron configurations are output.**

5.16 Detail: Retrieve and index transcript fragments that overlap introns

Consider the key-value pair $((t, \{(s_i, e_i)\}), (l_j, r_j))$, where $i \in [N]$ and j indexes samples. The key is an intron configuration composed of N introns, where the variable t stores the strand of the intron. The value is an ordered pair (l_j, r_j). The variable l_j stores the number of bases that must be traversed upstream of s_1 before another complete intron is found along s in some sample; the variable r_j stores the number of bases that must be traversed downstream of e_N before another complete intron is found along s in some sample. If no intron precedes (s_1, e_1), l is recorded as "not available," and if no intron succeeds (s_N, e_N), r is recorded as "not available."

In the first step referenced in the main text, each worker operates on an intron configuration at a time. In general, an intron configuration will have appeared in several samples, but the $\{(l_j, r_j)\}$ could vary from sample to sample because the introns identified in each sample are different. During the step, a given transcript fragment is assembled from reference exonic bases surrounding the intron configuration $(s, \{(s_i, e_i)\})$; the fragment starts at coordinate $s_1 - \min_j l_j$ and terminates at coordinate $e_1 + \min_j r_j$. Because we use the minima of l_j and r_j, there is no chance a given transcript fragment overlaps introns excluded from $(t, \{(s_i, e_i)\})$. Each transcript fragment is associated with a new reference name encoding its start and end coordinates and which introns are overlapped.

5.17 Detail: Compile coverage vectors and write bigWigs

Previous steps that have decided primary alignments (Sections 3.2 and 3.10) have output *exon differentials* associated with each read. The exon differentials of a read are assigned according to the following rules.

1. Divide each contig of a reference genome up into partitions, each spanning $genome\_partition\_length$ bases. Rail-RNA takes
$genome\_partition\_length = 5000$ by default.

2. Associate a $+1$ with the start coordinate of every exonic chunk of the read's primary alignment. Here, exonic chunks are separated by introns or, optionally, deletions from the reference.

3. Associate a -1 with the end position of every exonic chunk of the read's primary alignment.

4. Associate a $+1$ with the beginning of every genome partition spanned by the read.

These rules permit the reconstruction of the coverage of every base in a given genome partition *solely from the exon differentials within that partition.* This is conceptually simpler and requires less computational effort than compiling coverage from *intervals* representing the spliced alignments.

Figure 12 illustrates how coverage can be reconstructed from differentials. Initialize a variable v recording coverage at a given base to 0, and walk across a given partition base by base from left to right. Add the exon differentials at a given base to v to obtain the coverage at that base. Note that one exonic chunk lies across the border between partitions 2 and 3 and thus contributes an extra exon differential +1 to the beginning of partition 3. This special case illustrates the necessity of Rule 4: the extra +1 is necessary to obtain the correct coverage value of 1 at the beginning of the partition.

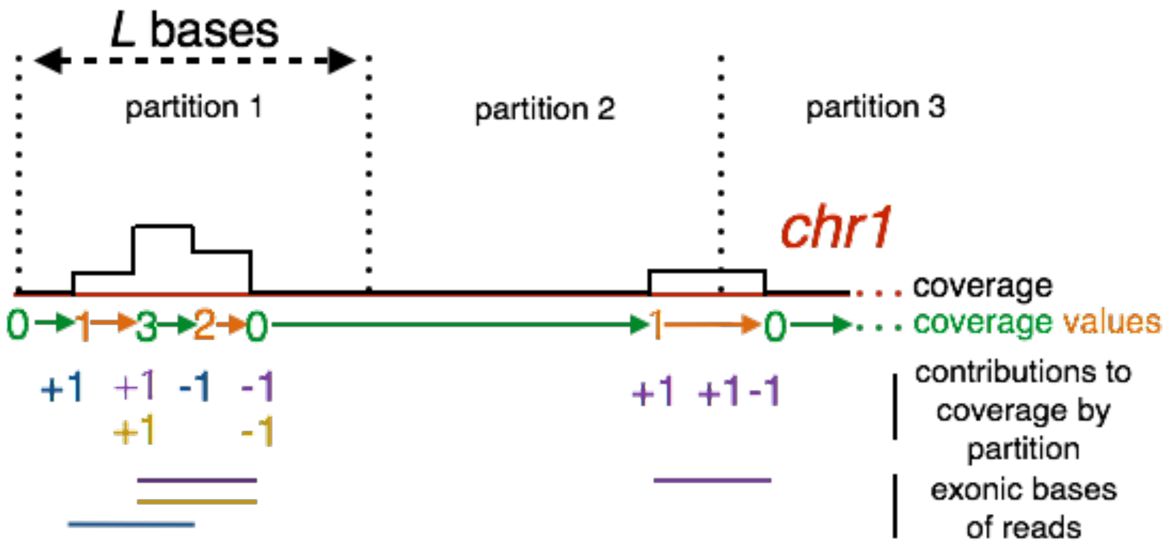


Figure 12: An illustration of how exon differentials are summed along a partition to give appropriate coverage values. Note that there is an extra +1 at the beginning of partition 3 so a worker operating on only partition 3 can obtain the right initial coverage value.

In the initial reduce step R_1, each worker operates on all of the exon differentials for a given sample and genome partition. The worker computes that sample's coverage at each base in the partition, a task made easier because the differentials are pre-sorted along the length of the partition. The output is compacted using run-length encoding: coverage is only written at a given position if it has changed from the previous position.

A challenge in this step is load balance. The number of exon differentials to be tallied can vary drastically across genome partitions. Therefore, an additional reduce step R_{1a} precedes R_1. R_{1a} simply sums exon differentials at a given position in the genome for a given sample. The way key/value pairs are paritioned in preparation for R_{1a}, a given task will process a random subset of genome positions. So for R_{1a}, the distribution of load across workers is close to uniform in practice. With the differentials pre-summed in this way, the number of key-value pairs processed in a given partition in step R_1 is upper-bounded by the number of positions in the partition, greatly improving load balance in that step.

The coverage output of R_1 is partitioned by sample and sorted by genome coordinate. In another reduce step R_2, each worker can thus operate on all the coverage output of a given sample at once. Coverage is written to a file in bedGraph format [58], which is converted to a compressed bigWig using the command-line tool bedGraphToBigWig [40]. In elastic mode, the bigWig for each sample is uploaded to S3, where it is subsequently accessible on the Internet. During the step R_2, Rail-RNA also computes the upper-quartile normalization factor [59] associated with each sample. In a final reduce step R_3, the normalization factors of all samples are written to a single output file, which in elastic mode is also uploaded to S3.