Boiler: Lossy compression of RNA-seq alignments using coverage vectors

Jacob Pritt [1,2,*], Ben Langmead [1,2*]

[1]Department of Computer Science, Johns Hopkins University, Baltimore, MD [2]Center for Computational Biology, Johns Hopkins University, Baltimore, MD

ABSTRACT

We describe Boiler, a new software tool for compressing and querying large collections of RNA-seq alignments. Boiler discards most per-read data, keeping only a genomic coverage vector plus a few empirical distributions summarizing the alignments. Since most per-read data is discarded, storage footprint is often much smaller than that achieved by other compression tools. Despite this, the most relevant per-read data can be recovered; we show that Boiler compression has only a slight negative impact on results given by downstream tools for isoform assembly and quantitation. Boiler also allows the user to pose fast and useful related queries without decompressing the entire file. Boiler is free open source software available from `github.com/jpritt/boiler`.

INTRODUCTION

Sequence Alignment/Map (SAM/BAM) (21) is a ubiquitous file format for storing RNA (and DNA) sequencing read alignments. Aligners and downstream analysis tools almost universally use SAM/BAM. For each aligned read, SAM/BAM stores the alignment's location, shape (described by the CIGAR string), base and quality sequences, and other data. BAM files are the binary equivalent of SAM, and BAM files are often sorted along the genome and compressed.

A drawback of SAM/BAM, and of any format that stores data on a per-read basis, is that file size grows close to linearly with the number of reads. But as sequencing continues to improve (25), and as public archives fill with more datasets, the burden of storing aligned sequencing data also increases rapidly. The Sequence Read Archive (19), which stores raw sequencing reads, grew from 3 to 4 petabases from February to August of 2015. It is increasingly common for RNA-seq studies to span hundreds or thousands of samples, with tens of millions of reads per sample (2, 16).

Compressed formats eliminate redundant data across reads or alignments, decreasing file size and allowing size to grow sub-linearly (rather than linearly) with the number of reads. CRAM (10), NGC (24), Goby (4) and REFEREE (7) use reference-based compression, which was proposed earlier (6, 15), to replace a read sequence with a concise description of how it differs from a substring of the reference. Quip (11) uses arithmetic coding together with a sequence model trained on-the-fly to compress losslessly and without a reference. Goby uses a range of strategies, including column-wise compression and detailed modeling of relationships between columns. REFEREE uses separable streams and clustering of quality strings. In these formats, the alignments and the fields are largely preserved, but are compressed along with neighbors row-wise (together with the other fields of the same alignment), or column-wise (with other instances of the field across alignments).

These studies also explore lossy compression schemes, in which less important data, such as read names and quality strings, is selectively discarded. Many tools optionally discard read names and quality values, and REFEREE clusters quality strings and replaces each with a single representative from its cluster.

Boiler takes a radically lossy approach to compressing RNA-seq alignments, yielding very small compressed outputs. Inspired by the notion of transform coding, Boiler converts alignment data from the "alignment domain," where location, shape and pairing information are stored for every alignment, to the "coverage domain," where the coverage vector is stored and alignment information is inferred where needed. Specifically, Boiler keeps only a set of coverage vectors and a few empirical distributions that partially preserve fields such as `POS` (offset into chromosome) `TLEN` (genomic outer distance), `XS:i` (strand) and `NH:i` (number of hits). Consequently, Boiler is lossy in an unusual sense: compressing and decompressing might cause alignments to shift along the genome, change shape, or become matched with the wrong mate. Table 1 presents a comparison of how CRAM, Goby, and Boiler preserve read information.

Despite being lossy, Boiler is lossless with respect to most of the data that is relevant to downstream RNA-seq tools for quantification and assembly; for example, the coverage vector, distribution of read lengths, and distribution of genomic outer distances (i.e. fragment lengths plus the lengths of all the spanned introns) are all preserved. We show that popular RNA-seq tools for isoform assembly and quantification – Cufflinks (26) and StringTie (23) – yield near-identical results when the input is Boiler-compressed.

Boiler yields extremely small file sizes, about 4-to-8-fold smaller than files produced by CRAM and Goby for samples with at least 5M paired-end reads. Unlike other compression tools, Boiler's compression ratio improves

*To whom correspondence should be addressed. Email: jacobpritt@gmail.com

substantially as input file size grows, growing from about 10-fold for lower-coverage unpaired samples to over 50-fold for higher-coverage samples. Speed and memory footprint are comparable to other compression tools despite the fact that, as we show, recovering alignments from a coverage vector is computationally hard. Also, because nucleotide data is removed, Boiler-compressed data is effectively de-identified, making it easier to pass between parties securely.

Boiler also provides a range of speedy queries. Many compression tools provide a way for the user to extract alignments spanning a particular genomic interval from the compressed file. REFEREE goes a step further by enabling faster queries when the user is concerned with only a subset of the fields. Boiler goes further still by providing fast queries that are directly relevant to downstream uses of RNA-seq data. Boiler allows user and downstream tools to (a) iterate over "bundles" of alignments according to inferred gene boundaries, (b) extract the coverage vector across a genomic interval, and (c) extract alignments overlapping a genomic interval.

MATERIALS AND METHODS

Compression

Boiler implements a lossy compression scheme that preserves only the data needed by downstream isoform assembly tools such as Cufflinks and StringTie. For this reason, read names and quality strings are discarded, along with other data that has little or no bearing on downstream RNA-seq analysis.

Given a set of alignments to a reference genome, Boiler first partitions the alignments into "bundles" of overlapping reads. Bundles are computed in the same manner as Cufflinks' initial bundling step: As sorted reads are processed, if the current read starts within 50 bases of the end of the current bundle, the read is added to the bundle. Otherwise, the current bundle is compressed and a new bundle is initialized beginning with the current read.

Boiler converts each bundle into a set of coverage vectors and tallies of observed read lengths. Note that for most Illumina sequencing datasets, reads are uniform-length (or nearly so, e.g. after trimming), yielding a concise tally. If any alignments in the bundle are paired-end, Boiler also stores a tally of observed genomic outer distances as reported by the aligner in the `TLEN` SAM field. Note that `TLEN` includes the lengths of all the introns spanned by the alignment, so we refer to this as "genomic outer distance," rather than "fragment length." The coverage vector is compressed using run-length encoding, which is particularly effective in low-coverage regions.

More specifically, each bundle is compressed as follows:

1. Boiler scans the bundle's spliced alignments and finds all splice sites spanned by at least one alignment. Boiler divides the portion of the genome spanned by the bundle into "partitions" formed by cutting at every splice site (Figure 1a).

2. Boiler assigns each alignment to a *bucket* according to: (a) the subset of partitions spanned by the alignment, (b) the value in the alignment's `NH:i` field, indicating the number of distinct locations where the read aligned to the reference, and (c) the value in the `XS:A` field, indicating whether spanned splice motifs are consistent with the sense (+) or anti-sense (−) strand of the gene. Alignments not spanning a junction usually lack the `XS:A` field; Boiler treats these as though the `XS:A` field contains a "dummy" value indicating the strand is unknown.

3. For each bucket, Boiler computes the coverage vector from the alignments assigned to it. Boiler writes the run-length encoded coverage vector (Figure 1b) followed by the read and genomic outer distance distributions (Figure 1c) for the junction.

Each bundle, which consists of many buckets, is compressed independently using the DEFLATE algorithm as implemented in the `zlib` package from the Python Standard Library. Each bundle is compressed separately to make targeted queries efficient, as discussed in the "Queries" section.

Unbundled alignments

Prior to compression, Boiler must identify and handle paired-end alignments that span bundles in unexpected ways. We call these *unbundled* alignments. Unbundled alignments fall into four categories: (a) one end falls within an intron spanned by the other end, (b) the two ends align to different chromosomes, (c) the two ends align to the same chromosome but very far from each other, (d) one end is assigned to the sense strand, while its mate is assigned to the anti-sense strand. Both TopHat and HISAT report such alignments, though they constitute only a small fraction of the alignments in a typical dataset.

These alignments are hard to fit into the bundling scheme described previously. Reads in case (a), also called "discordant reads", are biologically implausible. Boiler treats them as unpaired reads by default, however the user may choose to preserve these pairs.

Categories (b) and (c) could be scientifically relevant and should be preserved. For instance, alignments in category (b) may be evidence of gene fusions. Boiler stores all alignments in categories (b) and (c) in a special "unbundled alignments" section of the compressed file. Unbundled alignments are

Table 1. Comparison of the SAM fields stored by different compression tools. CRAM and Goby can preserve some fields through configurable options, summarized in the "Config" columns.

SAM Feature	CRAM		Goby		Boiler
	Default	Config.	Default	Config.	
Read Name	Yes[1]	No	No	Yes	No
Flags	Yes	No	Yes	No	No
Mapping Quality	Yes	No	Yes	No	No
Read Sequence	Yes	No	Yes	No	No
Quality Scores	No	Yes	Yes[2]	Yes	No
Tags	No	Yes	MD	Yes[3]	XS, NH

[1]CRAMtools documentation claims that by default read names should not be preserved, however we were not able to replicate this functionality.
[2]For mismatches only
[3]Goby preserves either all tags or just the MD tag.

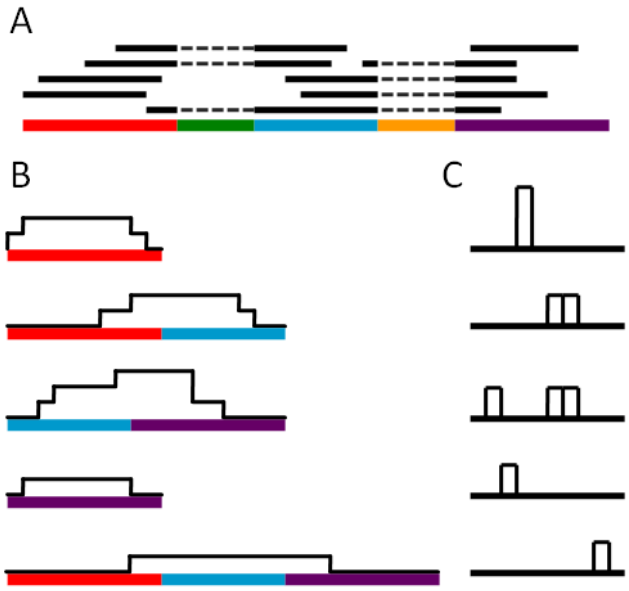

Figure 1. Illustration of how Boiler compresses alignments in a bundle, for a dataset with unpaired reads. (a) The genome is divided into "partitions" (colored segments) based on the processed splice sites. A bucket is defined by the subset of partitions spanned (as well as the values of the NH:i and XS:A fields, though these are omitted from the figure for simplicity). Each bucket stores (b) the coverage vector and (c) the length tally of the reads assigned to the bucket.

stored in *bundle-spanning buckets*. A bundle-spanning bucket is identical to a normal bucket, but includes the indices of the two bundles it spans in addition to the list of partitions spanned from each bundle. The bundle-spanning buckets are stored as a contiguous list, compressed in small chunks using zlib, and indexed to reduce work for targeted queries.

Treatment of pairs in (d) is configurable by the user. These pairs can either be treated as unpaired reads (which is consistent with Cufflinks and StringTie). Alternately they can remain paired, but with one end (selected randomly) modified to match the other end's strand.

Decompression

To decompress, Boiler first expands each bundle with the INFLATE algorithm as implemented in the Python `zlib` module, then expands each bucket.

When decompressing a bucket, Boiler's goal is to recreate the set of alignment intervals that yielded the bucket's coverage vector and read and genomic outer distance tallies. This is a two-step process; first the reads must be recovered from the coverage vector and read length tally ("read recovery"), then the recovered reads must be paired according to the paired length tally ("pairing").

The *read recovery* problem may not have a unique solution; e.g., consider a compressed dataset with read lengths l_1 and l_2 ($l_1 \neq l_2$) and a coverage vector containing 1 at all positions in the range $[0, l_1 + l_2)$. This case has two valid solutions:

$r_1 = [0, l_1),\ r_2 = [l_1, l_1 + l_2)$

and

$r_1 = [0, l_2),\ r_2 = [l_2, l_1 + l_2)$

Thus, we cannot guarantee perfect recovery of the compressed reads.

We define the read recovery problem as follows. Given a coverage vector and tally of read lengths, we seek a list of decompressed reads (genomic intervals) such that

1. the decompressed read lengths are a subset of those given in the tally,
2. at no position does the coverage vector produced by the decompressed reads exceed the value in the original coverage vector, and
3. the sum of the lengths of all decompressed reads is maximized.

This formulation is general enough to tolerate an input where the read length tally and coverage vector are not compatible, i.e., where no solution fits both precisely. In this case, the algorithm might decompress only some of the reads in the input tally.

We observe that the read recovery problem is NP-hard in general (proved by reduction from the Multiple Subset Sum Problem in Supplementary Note 1), but that some special cases are easily solved. When all reads are the same length, for example, the solution is unique and can be found efficiently. We also observe that second-generation sequencing produces datasets with uniform or near-uniform (e.g. after trimming) read-length tallies. These facts lead us to propose the greedy algorithm described below. The algorithm is not optimal in general – no polynomial-time algorithm can be – but it is well suited to cases where the input read lengths are uniform or almost uniform.

Greedy algorithms for extracting and pairing reads. The algorithm works from one end of the coverage vector to the other, extracting reads that are "consistent" with the coverage vector. A read is consistent with the vector if removing the read and decrementing the corresponding coverage-vector elements does not cause any vector elements to fall below zero. When a consistent read is selected for extraction, the corresponding coverage-vector elements are decremented and the process repeats. The process stops when the far end of the coverage vector is reached.

When reads have uniform length, the algorithm yields the correct solution. When reads have various lengths, the problem is harder and the algorithm may fail to yield the optimal solution. In the case where reads are various lengths, Boiler's algorithm uses heuristics to arrive at a solution where (a) the coverage vector induced by the extracted reads matches the true vector as closely as possible, and (b) the distribution of extracted read lengths matches the true distribution of read lengths as closely as possible. Boiler favors (a) over (b); i.e. it will artificially lengthen or shorten the extracted reads to fix small coverage discrepancies. The algorithm's heuristics are described in Supplementary Note 2.

If the data contains paired-end reads, we must also solve the *read pairing* problem. We would like Boiler to restore the paired-end relationships in a way that matches the original genomic outer distances as closely as possible. We start with

(a) a collection of reads (ends), all of which are initially unpaired, and (b) the "true" genomic outer distance tally, storing the frequencies of each distance, which was compiled and stored during compression.

We use the following greedy algorithm to pair up reads in a way that closely matches the true genomic outer distance tally. Each read is examined, working inward from the extremes, alternating between the left and right extremes. For each read, we seek the most distant read such that the resulting pairing is compatible with distances remaining in the tally. When two reads are paired in this way, they are removed from future consideration, and the corresponding element of the tally is decremented. Reads that are not paired in this way are matched up randomly in a second pass.

Queries

Boiler allows the user to query a compressed RNA-seq dataset to (a) iterate over genomic intervals delimiting regions of non-zero coverage, roughly corresponding to genes, (b) extract the genomic coverage vector across a specified genomic interval, and (c) extract alignments overlapping a specified genomic interval. Because each bundle of alignments is compressed separately, Boiler can answer such queries without decompressing the entire file.

Bundle boundaries are stored in the index at the beginning of the compressed file, so skipping to a particular bundle can be accomplished with a single uncompressed index lookup. To compute the coverage query (query b, above), Boiler combines the relevant portions of the coverage vectors for all the buckets overlapping the specified region. This requires that Boiler decompress the DEFLATED and run-length encoded coverage vectors, but does not require the more work-intensive read and pair recovery algorithms. Alignment-level queries (queries a and c above) are more expensive, requiring Boiler to run the greedy read recovery algorithm on each of the buckets overlapping the specified region.

Downstream tools like Cufflinks can be modified to query Boiler-compressed files directly, removing the need for an intermediate SAM/BAM file. When a downstream tool requires access only to information about gene boundaries (query a, above) or about targeted regions of the coverage vector (query b), Boiler's queries can be much faster than directly querying a sorted and indexed BAM file.

Implementation

Boiler is implemented in Python and is compatible with Python interpreters version 3 and above. All of the Python modules used by Boiler are in the Python Standard Library, making Boiler quite portable across Python installations and interpreters. For example, we use the fast PyPy interpreter for our experiments.

RESULTS

We used Flux Simulator v1.2.1 (18) to simulate 10 RNA-seq samples from the BDGP5 build of the *D. melanogaster* genome and the Ensembl release 70 (27) gene annotation. We simulated both paired-end and unpaired RNA-seq samples for a series of library sizes: 0.5, 1, 2.5, 5, 10, and 20 million reads. We also simulated two samples from the hg19 build of the human genome and Gencode v12 gene annotation (9) containing 20 and 40 million paired-end reads. We also used a real human RNA-seq sample from the GEUVADIS (16) project containing approximately 20 million paired-end reads. All samples were aligned to the reference genome using Tophat 2 v2.1.0 (13) with default parameters. The *D. melanogaster* samples were aligned to the BDGP5 reference genomes and the human samples were aligned to hg19.

We compared Boiler's speed, compression ratio, and peak memory usage to Goby and CRAMTools. Boiler and Goby remove read names by default, but CRAM does not. (CRAMtools has an option to preserve read names, but we cannot find a working mechanism in version 3 to remove them.) For a fair comparison, we stripped the read names before compressing.

Efficiency and compression ratio

We compressed each set of alignments with Boiler v1.0.0, CRAMTools v3.0, and Goby v2.3.5. Boiler was run with PyPy v2.4 and CRAMTools and Goby were run with Java v1.7. All were run with default parameters on the Homewood High Performance Compute Cluster at Johns Hopkins University. Each cluster computer has 2 Intel Xeon X5660 2.80GHz processors and 48 GB of RAM. We measure running time by adding the `user` and `sys` times reported by the Linux `time` command. Each tool runs predominantly on a single thread and processor. We measure peak memory usage in Python by spawning a new child process for the command and polling maximum resident set size (RSS) using the Python `resource` package's `getrusage` function. Peak memory usage for Boiler and Goby was consistent across runs, but CRAM memory usage varied widely between runs. We report the median peak memory of 10 runs for greater consistency.

Boiler takes roughly 2 times longer than CRAMTools and Goby to compress the D. melanogaster samples and about 2-7 times longer for the human samples (Table 2, Figure 2). It requires less memory than Goby for all samples and less than CRAM for small datasets, however for larger datasets CRAM memory is capped at around 2 GB (Table 3).

Importantly, Boiler usually produces far smaller compressed files than CRAMTools or Goby. We measure both compressed file size (Table 4) and the "compression ratio" of original to compressed file size (Figure 3). The "original" file is a sorted BAM file with read names removed. For low-coverage unpaired datasets, CRAM's compression ratio is superior to Boiler's. However, we observe that while CRAMTools and Goby's compression ratios remain flat as the *D. melanogaster* library size increases, Boiler's ratios improve substantially (Figure 3), achieving its best compression ratios for the 20M-read samples: 56-fold for unpaired and 39-fold for paired-end samples. Boiler's compression ratio is consistently better than the other tools' for paired-end samples and improves as library size increases. For high coverage *D. melanogaster* and all human dataset, Boiler achieves compression ratios 3-4-fold higher than CRAM and 7-8-fold higher than Goby.

Fidelity

Boiler discards read nucleotide and quality-value data. So while Boiler is not appropriate for pipelines where

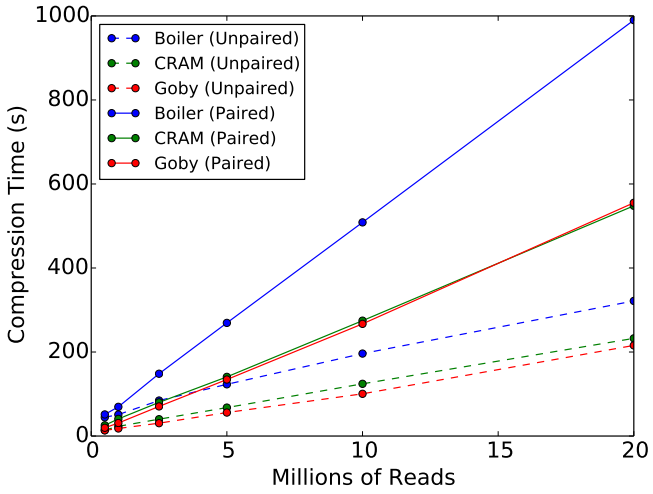

Figure 2. The compression time for simulated Drosophila paired-end datasets compressed by Boiler, CRAM and Goby.

downstream tools measure non-reference alleles – e.g. for variant calling, allele-specific expression, or RNA editing – we show that Boiler is appropriate for the common case where downstream tools are concerned with assembling and quantifying isoforms, e.g. Cufflinks and StringTie.

Boiler tends to "shuffle" alignment data in certain ways during compression. Some of the shuffling is harmless, having no adverse effect on downstream results from Cufflinks and StringTie. But some shuffling could in theory be harmful, negatively impacting Cufflinks and StringTie. Using both simulated and real data, we (a) establish the nature of the shuffling introduced by Boiler, (b) show there is only slight harmful shuffling in practice, and (c) show that the overall amount of shuffling is smaller (often much smaller) than the shuffling that results from substituting one technical replicate for another.

Table 2. Compression times in seconds.

Dataset	Boiler	CRAM	Goby
Drosophila, Simulated Unpaired			
0.5M	44.6	14.1	13.4
1M	51.1	23.1	18.0
2.5M	84.5	40.1	30.6
5M	123.1	67.7	55.9
10M	196.3	124.3	100.5
20M	321.6	232.5	215.7
Drosophila, Simulated Paired			
0.5M	51.4	25.5	19.6
1M	69.5	40.8	31.0
2.5M	148.3	80.1	70.5
5M	269.3	140.8	134.5
10M	508.6	274.6	267.0
20M	990.0	548.0	555.2
Human			
GEUVADIS 20M	4553.8	684.4	841.7
Simulated 20M	2298.6	864.1	1110.9
Simulated 40M	7792.4	1566.1	1935.5

Alignment-level fidelity

Boiler compression can change where alignments lie on the genome and how they are paired. Here we ask how well alignment locations are preserved after Boiler compression of the TopHat 2-aligned samples. We measure alignment-level precision and recall in two ways. First we ignore read pairings. For each aligned unpaired read (or end of a paired-end read) in the original file, we seek a corresponding alignment in the compressed file where the genomic position of the alignment and of all overlapped splice junctions are identical. This counts as a true positive, and the alignments involved are "matched." An alignment can be matched with at most one other alignment. An alignment in the original file that fails

Table 3. Peak memory usage (GB) reported by Python. Numbers reported are the median across 10 runs.

Dataset	Boiler	CRAM	Goby
Drosophila, Simulated Unpaired			
0.5M	0.38	0.74	0.47
1M	0.29	1.37	0.45
2.5M	0.24	1.12	0.48
5M	0.24	1.13	0.81
10M	0.24	0.96	1.31
20M	0.50	1.53	3.03
Drosophila, Simulated Paired			
0.5M	0.32	1.15	0.50
1M	0.31	1.91	1.71
2.5M	0.30	1.96	0.85
5M	0.32	1.18	2.55
10M	0.79	1.11	3.66
20M	1.57	0.99	4.39
Human			
GEUVADIS 20M	3.23	2.08	6.86
Simulated 20M	6.11	2.08	7.63
Simulated 40M	9.14	2.08	9.69

Table 4. Size of compressed files (MB) compared to the original sorted BAM with read names removed.

Dataset	BAM	Boiler	CRAM	Goby
Drosophila, Simulated Unpaired				
0.5M	26.0	2.7	0.9	3.8
1M	49.4	3.6	1.6	7.4
2.5M	120.3	5.6	3.8	18.0
5M	222.2	7.6	6.7	32.9
10M	436.1	11.0	12.6	62.6
20M	883.0	15.0	23.2	114.0
Drosophila, Simulated Paired				
0.5M	56.12	4.3	7.7	13.4
1M	104.6	6.7	14.2	25.6
2.5M	255.6	12.9	33.9	63.5
5M	488.1	20.0	61.9	121.0
10M	955.0	31.6	116.9	234.2
20M	1902.6	49.0	226.2	456.3
Human				
GEUVADIS 20M	2017.9	78.0	288.8	609.6
Simulated 20M	2117.3	71.6	273.5	552.5
Simulated 40M	3858.4	118.0	491.4	1007.4

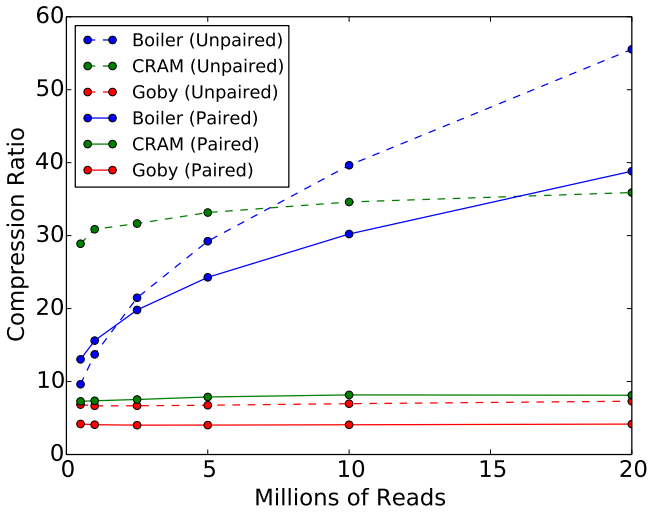

Figure 3. The compression ratio for simulated Drosophila paired-end datasets compressed by Boiler, CRAM and Goby, compared to the original sorted BAM with read names removed.

to match an alignment in the compressed file counts as a false negative and the converse is a false positive Given these definitions, precision and recall are shown in the left-hand columns of Table 5 (labeled "Ignoring pairings"). Both range from about 96% to about 99% across the samples tested.

We also measure precision and recall in a way that takes pairing into account: for each aligned pair, we seek a corresponding pair in the compressed file where both ends match their counterparts in terms of their genomic position and the positions of splice junctions. These results are shown in the right-hand columns of Table 5 (labeled "Including pairings"). Here precision and recall are lower, ranging from about 20% to about 50% across samples.

We also measured genomics outer distance distribution (excluding discordant alignments) before and after Boiler compression and found that they match closely as illustrated in Figure 4. The results are emblematic of Boiler's strategy: aggregate distributions are preserved, but links between particular alignments and particular points in the distribution are lost. As a result, some data is "shuffled;" ends themselves are largely unchanged, but pairings between the ends are shuffled in a way that preserves the aggregate genomic outer distance distribution.

Isoform fidelity

Having established Boiler's lossy-ness and shuffling behavior, we now assess the degree to which loss and shuffling have an adverse effect on downstream results obtained by Cufflinks v2.2.1 and StringTie v1.0.3. StringTie was run with default parameters. Cufflinks was run with the `--no-effective-length-correction` parameter to avoid variability due to an issue (recently resolved) in how Cufflinks performs effective transcript length correction (5).

Let T be the true simulated transcriptome, including abundances for each transcript, which we can extract from the Flux-generated `.pro` and `.gtf` files. Let $\hat{T}$ be the transcriptome assembled and quantified from the original alignments, which we extract from the Cufflinks/StringTie output. Let $\hat{T}'$ be the same but for the Boiler-compressed alignments. Here we ask whether $\hat{T}$ and $\hat{T}'$ are approximately equidistant from T, indicating Boiler's loss and shuffling are not having an adverse effect.

We define a function for measuring the distance between two transcripts t_1 and t_2 assembled with respect to a reference genome. The function outputs a value between 0 and 1, with 0 indicating the transcripts do not match and 1 indicating a perfect match.

A transcript t can be represented as a set of exons $\{e_1,...e_n\}$, each defined by its start and stop positions. We first define a scoring function for two exons $e_1=(x_1,y_1)$ and $e_2=(x_2,y_2)$:

$$s(e_1,e_2)=1-\frac{min(|x_2-x_1|,k)}{2k}-\frac{min(|y_2-y_1|,k)}{2k}$$

for some threshold k. We further define function $max(e,\hat{t})$ to be the exon $\hat{e}$ from $\hat{t}$ with the highest score $s(e,\hat{e})$.

The weighted precision for pre-compression alignments is:

$$\sum_{\hat{t}\in\hat{T}}\max_{t\in T}\big(score(\hat{t},t)\big)\cdot c(\hat{t})$$

Where $c(\hat{t})$ is the predicted coverage level of t as reported by Cufflinks/StringTie. This measure is weighted both by the coverage of the assembled transcript and by the similarity of the matched-up transcripts. Precision for the post-compression alignments is calculated similarly, using $\hat{T}'$ instead of $\hat{T}$.

Similarly, weighted recall is

$$\sum_{t\in T}\max_{\hat{t}\in\hat{T}}\big(s(t,\hat{t})\big)\cdot c(t)$$

Table 5. Precision and recall of SAM reads.

Dataset	Ignoring Pairings		Including Pairings	
	Precision	Recall	Precision	Recall
Drosophila, Simulated Unpaired				
0.5M	0.980	0.989	–	–
1M	0.971	0.983	–	–
2.5M	0.960	0.976	–	–
5M	0.954	0.971	–	–
10M	0.953	0.967	–	–
20M	0.960	0.970	–	–
Drosophila, Simulated Paired				
0.5M	0.977	0.986	0.538	0.542
1M	0.968	0.980	0.453	0.458
2.5M	0.957	0.971	0.333	0.338
5M	0.957	0.970	0.260	0.263
10M	0.960	0.970	0.217	0.219
20M	0.963	0.970	0.175	0.176
Human				
GEUVADIS 20M	0.982	0.984	0.252	0.253
Simulated 20M	0.972	0.977	0.331	0.332
Simulated 40M	0.974	0.977	0.318	0.319

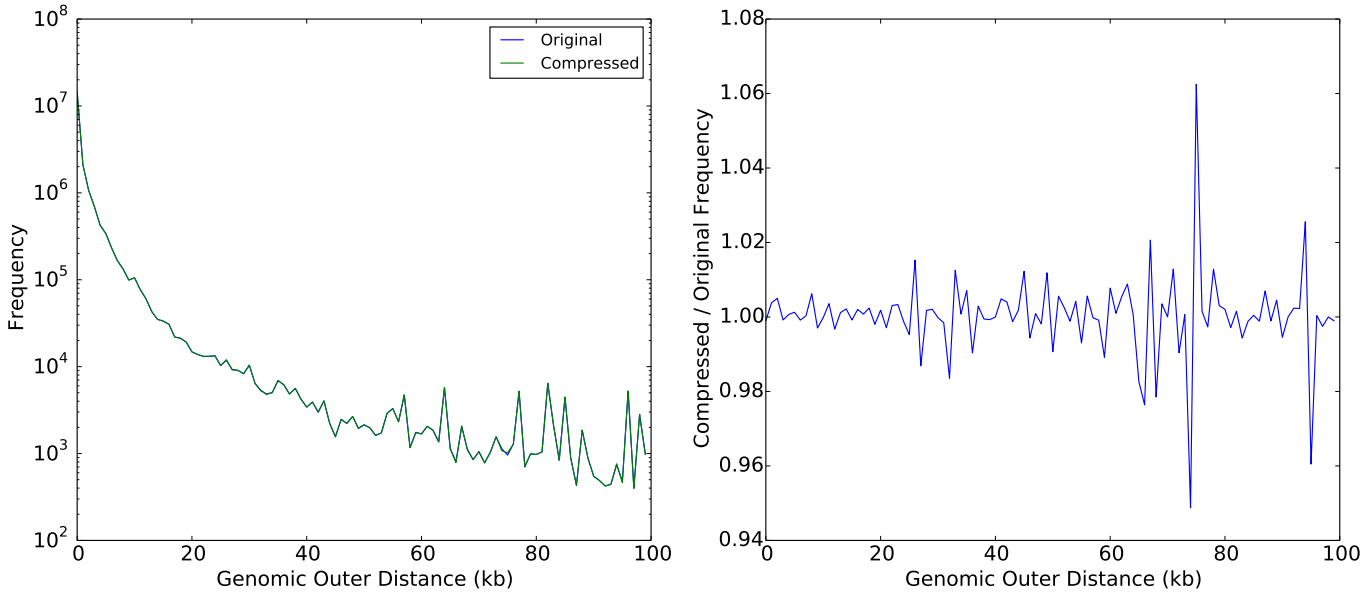

Figure 4. Comparison of genomic outer distances for the 10M paired-end *D. melanogaster* sample. **Left:** Distribution of paired-end genomic outer distances up to 100,000 bases, before and after compression. **Right:** The ratio of genomic outer distances for each distance before and after compression, up to 100,000 bases.

Algorithm 1 Transcript scoring algorithm

procedure SCORE(t,$\hat{t}$)
  $s \leftarrow 0$ ▷ Score before normalization
  $n \leftarrow 0$ ▷ Number of matching exons found
  for $e \in t$ **do**
    $\hat{e} = max(e,\hat{t})$
    if $e = max(\hat{e},t)$ **then**
      $s \leftarrow s + s(e,\hat{e})$
      $n \leftarrow n+1$
    end if
  end for
  return $\frac{s}{|t|+|\hat{t}|-n}$
end procedure

Where $c(t)$ is the true coverage level for t as reported by Flux.

We calculated the weighted precision (Tables 6) and recall (Table 7) for all simulated samples. While there are small differences in all experiments (as expected due to Boiler's shuffling behavior), overall they indicate that the Boiler-compressed alignments are not adversely impacting weighted precision and recall.

Other Isoform Fidelity Scores

Supplementary Note 3 describes a complementary experiment using weighted k-mer recall (WKR), which does not depend on a distance function. Supplementary Note 4 describes a third accuracy measure called the Tripartite Score, which uses the same distance, but can compare two assembled transcriptomes to a third "reference" transcriptome. The results from those methods substantially agree with the results above.

Shuffling relative to technical replicates

Next we investigate the amount of "shuffling" introduced by Boiler – causing some reads to shift along the genome and scrambling some paired-end relationships – and whether the effect is large or small compared to the shuffling that occurs when switching from one technical replicate to another.

We first construct five artificial technical replicates by generating five times the desired number of reads with Flux Simulator and randomly partitioning the resulting read file five ways. We then assemble and quantify each using Cufflinks and StringTie. Let $\hat{T}_1$, $\hat{T}_2$, ..., $\hat{T}_5$ be the corresponding transcriptomes. We also pick a technical replicate ($\hat{T}_1$, say) to compress with Boiler. Let $\hat{T}_1'$ be the result of running Cufflinks/StringTie on the Boiler-compressed alignments. We calculate the weighted precision and recall of $\hat{T}_1'$ relative to $\hat{T}_1$. Finally, we calculate weighted precisions and recalls between all 10 ordered pairs of technical-replicate transcriptomes: $(\hat{T}_1,\hat{T}_2),(\hat{T}_1,\hat{T}_3),...,(\hat{T}_4,\hat{T}_5)$. Results are presented in Tables 8 and 9. Precision and recall between technical replicates is shown as a range from the minimum to the maximum observed among the 10 ordered pairs.

Precision and recall after Boiler-compression are consistently higher than precision and recall between technical replicates. This indicates that the amount of shuffling due to Boiler compression is consistently smaller than the amount due to switching between technical replicates.

It is worth noting how precision and recall change relative to per-sample coverage. Precision and recall between technical replicates increases as per-sample coverage increases, indicating that as more transcripts become deeply covered, the adverse effect of technical-replicate shuffling decreases. On the other hand, precision and recall of $\hat{T}_1'$ versus $\hat{T}_1$ decreases as per-sample coverage increases. This is notable because there may be very high levels of coverage for which

the Boiler precision/recall drops below the technical-replicate precision/recall.

However, for the realistic levels of coverage that we tested, Boiler's shuffling remains significantly more accurate than technical replicates.

Queries

Recovering coverage vectors from a Boiler-compressed file requires that Boiler decompress and combine coverage vectors for all the overlapping bundles. The decompression involves use of the DEFLATE algorithm and run-length decoding, but does not involve the more expensive read and pair recovery algorithms. SAMtools, on the other hand, does not explicitly represent the coverage vectors in a BAM file. Instead, coverage information must be recovered from the BAM file by first extracting the relevant alignments, then composing the coverage vector using another tool like BEDTools:

```
samtools view -b -h x.bam c:start-end |
genomeCoverageBed -bga -split -ibam stdin
-g chromosomes.txt
```

We compared the time required for Boiler to respond to coverage queries to the time required for SAMtools/BEDTools. Specifically, we iterated over all bundle boundaries in several *D. melanogaster* samples and queried for the coverage vector within those boundaries using both Boiler and SAMtools/BEDTools. Figure 5 compares the tools

Table 6. Reference-based Precision.

Dataset	Cufflinks		StringTie	
	Original	Compressed	Original	Compressed
Drosophila, Simulated Unpaired				
0.5M	0.364	0.363 (-0.1%)	0.427	0.427 (+0.0%)
1M	0.432	0.432 (-0.0%)	0.507	0.506 (-0.1%)
2.5M	0.527	0.525 (-0.3%)	0.604	0.604 (-0.0%)
5M	0.564	0.563 (-0.1%)	0.618	0.618 (-0.0%)
10M	0.583	0.583 (-0.1%)	0.624	0.623 (-0.1%)
20M	0.602	0.601 (-0.3%)	0.631	0.629 (-0.3%)
Drosophila, Simulated Paired				
0.5M	0.583	0.581 (-0.4%)	0.511	0.512 (+0.1%)
1M	0.611	0.606 (-0.9%)	0.587	0.586 (-0.2%)
2.5M	0.632	0.629 (-0.4%)	0.623	0.623 (+0.0%)
5M	0.635	0.633 (-0.4%)	0.619	0.617 (-0.4%)
10M	0.644	0.642 (-0.3%)	0.620	0.617 (-0.6%)
20M	0.639	0.634 (-0.8%)	0.613	0.610 (-0.4%)
Human, Simulated Paired				
20M	0.552	0.552 (+0.1%)	0.537	0.534 (-0.5%)
40M	0.554	0.553 (-0.1%)	0.540	0.537 (-0.5%)

Table 7. Reference-based Recall

Dataset	Cufflinks		StringTie	
	Original	Compressed	Original	Compressed
Drosophila, Simulated Unpaired				
0.5M	0.583	0.583 (+0.0%)	0.521	0.521 (+0.0%)
1M	0.708	0.707 (-0.1%)	0.675	0.675 (-0.1%)
2.5M	0.791	0.784 (-0.8%)	0.794	0.794 (-0.0%)
5M	0.822	0.821 (-0.0%)	0.835	0.835 (+0.0%)
10M	0.824	0.824 (-0.0%)	0.843	0.843 (+0.0%)
20M	0.827	0.826 (-0.1%)	0.848	0.847 (-0.1%)
Drosophila, Simulated Paired				
0.5M	0.732	0.729 (-0.4%)	0.688	0.687 (-0.2%)
1M	0.799	0.797 (-0.2%)	0.789	0.789 (+0.1%)
2.5M	0.825	0.827 (+0.2%)	0.840	0.840 (+0.0%)
5M	0.840	0.839 (-0.1%)	0.856	0.855 (-0.2%)
10M	0.828	0.830 (+0.2%)	0.849	0.850 (+0.1%)
20M	0.826	0.821 (-0.6%)	0.849	0.848 (-0.1%)
Human, Simulated Paired				
20M	0.762	0.762 (-0.0%)	0.791	0.789 (-0.2%)
40M	0.792	0.789 (-0.4%)	0.822	0.816 (-0.7%)

Table 8. Non-reference-based precision.

Dataset	Cufflinks		Stringtie	
	Calculated	Tech Reps (min–max)	Calculated	Tech Reps (min–max)
Drosophila, Simulated Unpaired				
0.5M	0.996	0.246–0.255	0.996	0.277–0.286
1M	0.996	0.322–0.333	0.996	0.358–0.372
2.5M	0.995	0.430–0.437	0.996	0.493–0.504
5M	0.994	0.504–0.512	0.997	0.581–0.587
10M	0.991	0.570–0.577	0.996	0.656–0.661
20M	0.990	0.629–0.634	0.995	0.707–0.716
Drosophila, Simulated Paired				
0.5M	0.979	0.425–0.440	0.995	0.362–0.376
1M	0.977	0.522–0.529	0.996	0.463–0.474
2.5M	0.967	0.624–0.632	0.994	0.587–0.598
5M	0.964	0.674–0.684	0.992	0.653–0.671
10M	0.957	0.713–0.722	0.991	0.704–0.715
20M	0.953	0.746–0.754	0.989	0.744–0.756
Human				
GEUVADIS 20M	0.919		0.990	
Simulated 20M	0.968	0.765–0.788	0.990	0.753–0.777
Simulated 40M	0.962	0.786–0.809	0.986	0.778–0.801

Table 9. Non-reference-based recall.

Dataset	Cufflinks		Stringtie	
	Calculated	Tech Reps (min–max)	Calculated	Tech Reps (min–max)
Drosophila, Simulated Unpaired				
0.5M	0.997	0.246–0.255	1.000	0.277–0.286
1M	0.995	0.322–0.333	0.999	0.358–0.372
2.5M	0.996	0.430–0.437	0.998	0.493–0.504
5M	0.995	0.504–0.512	0.998	0.581–0.587
10M	0.993	0.570–0.577	0.997	0.656–0.661
20M	0.990	0.629–0.634	0.997	0.707–0.716
Drosophila, Simulated Paired				
0.5M	0.980	0.425–0.440	0.997	0.362–0.376
1M	0.979	0.522–0.529	0.997	0.463–0.474
2.5M	0.971	0.624–0.632	0.994	0.587–0.598
5M	0.970	0.674–0.684	0.993	0.653–0.671
10M	0.959	0.713–0.722	0.991	0.704–0.715
20M	0.954	0.746–0.754	0.990	0.744–0.756
Human				
GEUVADIS 20M	0.921		0.990	
Simulated 20M	0.968	0.765–0.788	0.991	0.753–0.777
Simulated 40M	0.961	0.786–0.809	0.978	0.778–0.801

both in terms of average query time (left) and per-bundle query time (right). As expected, Boiler is consistently faster than SAMtools/BEDTools, with Boiler taking under 0.1 seconds on average, and SAMtools/BEDTools taking close to 0.75 seconds.

We also compared alignment query times for Boiler to those for the indexed BAM. An alignment query is easy to handle with SAMtools:

```
samtools view -h x.bam chrom:start-end
```

Boiler must both decompress the relevant bundles and run the greedy read and pair recovery algorithms. Figure 6 compares the tools both in terms of average query time (left) and per-bundle query time (right). As expected, because of the need to run read and pair recovery, Boiler is consistently slower than SAMtools. Even so, Boiler's average response time is under 0.15 seconds.

DISCUSSION

Boiler applies principles of lossy compression and transform coding to the problem of compressing RNA-seq alignments. Beyond the lossy methods used in CRAMTools and Goby, Boiler additionally discards most of the data that ties individual reads to their aligned positions and shapes. Boiler instead stores coverage vectors and read- and outer-distance tallies, effective shifting from the "alignment domain" to the "coverage domain." While this can cause alignments to shift along the genome or pair with the wrong mate, the shuffling effect is modest compared to the shuffling induced by switching between technical replicates, and adverse effects on downstream tools for isoform assembly and quantification are minimal.

Boiler is not a general-purpose substitute for RNA-seq SAM/BAM files, but it is an extremely space-efficient alternative that works well with tools like Cufflinks and StringTie. Though we have not shown it here, we also expect Boiler compression to work well with annotation-based tools like featureCounts (17) and HiTseq (1), as well as downstream differential-expression tools. Also, because Boiler discards information about non-reference alleles, Boiler archives are more readily sharable even when the input data is protected by privacy provisions such as dbGaP.

In future work, it is important to explore alternatives to the greedy decompression algorithms described here. For example, Boiler's greedy algorithm for extracting reads from a coverage distribution assumes that the distribution of input read lengths has a dominant mode. This is a reasonable assumption for Illumina data (including data that has passed through a read trimmer beforehand), but not so for other sequencing technologies. Also, the compression and decompression algorithms currently implemented in Python could be accelerated by moving to a compiled language such as C/C++. In general, there are many opportunities to make the read extraction and pairing algorithms more accurate, faster, and less hampered by assumptions about the data.

Finally, we note that the methods used here to characterize Boiler's shuffling effect are more generally useful for evaluating any upstream tool that modifies the data. For example, one could apply the same techniques to evaluate a tool for read trimming or digital normalization, or to compare many parameterizations of the spliced aligner.

Boiler is available from `github.com/jpritt/boiler` and is distributed under the open source MIT license.

ACKNOWLEDGEMENTS

We are grateful to Abhinav Nellore and Chris Wilks for commenting on the software and manuscript.

FUNDING

BTL and JP were supported by a Sloan Research Fellowship to BL and by National Science Foundation grant IIS-1349906 to BL.

Conflict of interest statement. None declared.

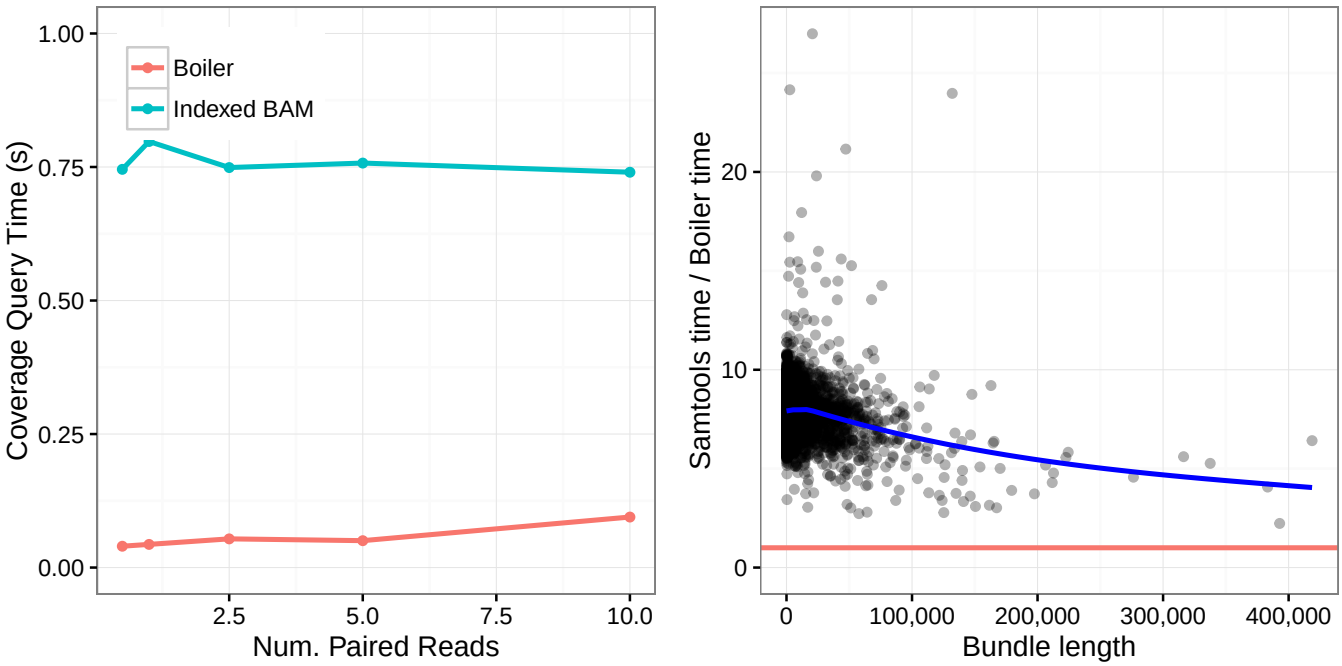

Figure 5. Comparison of coverage query times for Boiler the indexed BAM for all bundles. **Left:** Average query time for varying *D. melanogaster* paired-end datasets. **Right:** Ratio of Samtools / Boiler query time for each bundle in the 10M *D. melanogaster* paired-end dataset plotted as a function of bundle length. The blue line denotes the best fit line for the points, the red line is $y = 1$.

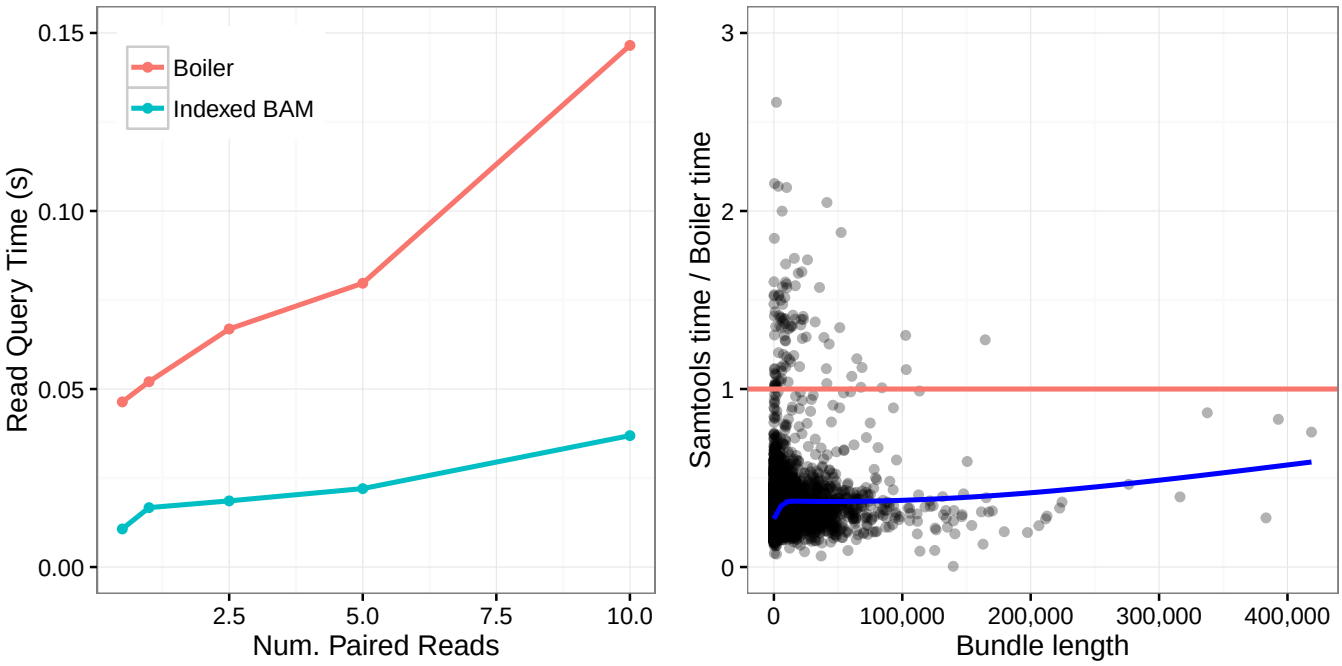

Figure 6. Comparison of alignment query times for Boiler the indexed BAM for all bundles. **Left:** Average query time for varying *D. melanogaster* paired-end datasets. **Right:** Ratio of Samtools / Boiler query time for each bundle in the 10M *D. melanogaster* paired-end dataset plotted as a function of bundle length. The blue line denotes the best fit line for the points, the red line is $y = 1$.

REFERENCES

1. Anders S, Pyl PT, Huber W (2014) HTSeq–A Python framework to work with high-throughput sequencing data. *Bioinformatics*, **31(2)**, 166–169.
2. Ardlie KG, Deluca DS, Segrè AV, Sullivan TJ, Young TR, Gelfand ET, Trowbridge CA, Maller JB, Tukiainen T, Lek M et al (2015) The Genotype-Tissue Expression (GTEx) pilot analysis: Multitissue gene regulation in humans. *Science*, **348(6235)**, 648-660.
3. Bonfield JK, Mahoney MV (2013) Compression of fastq and sam format sequencing data. *PLoS ONE*, **8(3)**, e59190.
4. Campagne F, Dorff K, Chambwe N, Robinson JT, Mesirov JP (2013) Compression of Structured High-Throughput Sequencing Data. *PLoS ONE*, **8(11)**, e79871.
5. Cufflinks pull request 32, https://github.com/cole-trapnell-lab/cufflinks/pull/32 (2015).
6. Daily K, Rigor P, Christley S, Xie X, Baldi P (2010) Data structures and compression algorithms for high-throughput sequencing technologies. *BMC bioinformatics*, **11(1)**, 514.
7. Filippova D, Kingsford C (2015) Rapid, separable compression enables fast analyses of sequence alignments. *ACM Conference on Bioinformatics*, 194–201.
8. Hach F, Numanagic I, Alkan C, Sahinalp SC (2012) SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics*, **28(23)**, 3051–3057.
9. Harrow J, Frankish A, Gonzalez JM, Tapanari E, Diekhans M, Kokocinski F, Aken BL, Barrell D, Zadissa A, Searle S et al (2012) GENCODE: the reference human genome annotation for The ENCODE Project. *Genome research*, **22(9)**, 1760–1774.
10. Hsi-Yang FM, Leinonen R, Cochrane G, Birney E (2011) Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Research*, **5**, 734–740.
11. Jones DC, Ruzzo WL, Peng X, Katze MG (2012) Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic Acids Research*, **40(22)**, e171.
12. Kent WJ, Zweig AS, Barber G, Hinrichs AS, Karolchik D (2010) BigWig and BigBed: enabling browsing of large distributed datasets. *Bioinformatics*, **26(17)**, 2204–2207.
13. Kim D, Pertea G, Trapnell C, Pimentel H, Kelley R, Salzberg SL (2013). TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions *Genome Biol*, **14(4)**, R36.
14. Kim D, Langmead B, Salzberg SL (2013). HISAT: a fast spliced aligner with low memory requirements *Nature methods*, **12(4)**, 357-360.
15. Kozanitis C, Saunders C, Kruglyak S, Bafna V, Varghese G (2011) Compressing genomic sequence fragments using SlimGene. *Journal of Computational Biology*, **18(3)**, 401-413.
16. Lappalainen T, Sammeth M, Friedländer MR, AC?t Hoen P, Monlong J, Rivas MA, Gonzàlez-Porta M, Kurbatova N, Griebel T, Ferreira PG et al (2013) Transcriptome and genome sequencing uncovers functional variation in humans. *Nature*, **501(7468)**, 506-511.
17. Liao Y, Smyth GK, Shi W (2014) featureCounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics*, **30(7)**, 923-930.
18. Griebel T, Zacher B, Ribeca P, Raineri E, Lacroix V, Guigó R, Sammeth M (2012) Modelling and simulating generic RNA-Seq experiments with the flux simulator. *Nucleic acids research*, **40(20)**, 10073-10083.
19. Leinonen R, Sugawara H, Shumway M (2011) The sequence read archive. *Nucleic acids research*, **39(suppl 1)**, D19-D21.
20. Li H, Durbin R (2009) Fast and accurate short read alignment with Burrows-Wheeler Transform. *Bioinformatics*, **25**, 1754–1760.
21. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecases G, Durbin R (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25(16)**, 2078–2079.
22. Ochoa I, Asnani H, Bharadia D, Chowdhury M, Weissman T, Yona G (2013) QualComp: a new lossy compressor for quality scores based on rate distortion theory. *BMC Bioinformatics*, **14**, 187.
23. Pertea M, Pertea GM, Antonescu CM, Chang TS, Mendell JT, Salzberg SL (2015). StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nature Biotechnology*, **33(3)**, 290-295.
24. Popitsch N, von Haeseler A (2013). NGC: lossless and lossy compression of aligned high-throughput sequencing data. *Nucleic Acids Research*, **41(1)**, e27.
25. Stephens ZD, Lee SY, Faghri F, Campbell RH, Zhai C, Efron MJ, Iyer R, Schatz MC, Sinha S, Robinson GE (2015) Big Data: Astronomical or Genomical? *PLoS Biol*, **13(7)**, e1002195.
26. Trapnell C, Williams B, Pertea G, Mortazavi A, Kwan G, van Baren J, Salzberg S, Wold B, Pachter L (2010). *Nature Biotechnology*, **28(5)**, 511–515.
27. Yates A, Akanni W, Amode MR, Barrell D, Billis K, Carvalho-Silva D, Cummins C, Clapham P, Fitzgerald S, Gil L et al (2015). *Nucleic Acids Research*, **28(5)**, gkv1157.

SUPPLEMENT

Supplementary Note 1

LEMMA 0.1. *The read decompression problem is strongly NP-hard.*

Proof Consider the Multiple Subset Sum Problem (MSSP), defined as follows. Given n items with weights $w_1,w_2,...,w_n$ and m knapsacks with capacities $c_1,c_2,...,c_m$, assign items to knapsacks such that:

1. each item is assigned to up to 1 knapsack
2. the capacity of each knapsack is not exceeded by the combined weights of the items assigned to it
3. the total weight of the items in all the knapsacks is maximized.

MSSP is known to be strongly NP-hard (28).

We reduce MSSP to a special case of the read decompression problem where the coverage vector never exceeds 1. We first construct a vector C encoding knapsack capacities in unary. We start with empty C then, for each i, append c_i 1s followed by a single 0. Note that because the length of C depends on the numeric knapsack weights, this is a pseudo-polynomial time reduction. Next, we let the read length tally equal the item weight tally. Finally, we run our decompression algorithm on the coverage vector C and read length tally. The algorithm packs reads into the nonzero stretches of C. This solution is converted to an MSSP solution by converting reads to the corresponding items and stretches of the coverage vector to the corresponding knapsacks.

The reduction satisfies the requirements of a pseudo-polynomial transformation (29). Hence, the read decompression problem for unpaired reads is strongly NP-hard. □

Supplementary Note 2

Greedy algorithm for obtaining reads from coverage vector. The algorithm works from one end of the coverage vector to the other, removing reads that remain consistent with the coverage vector. We take advantage of the homogeneous read length distribution produced by sequencing experiments by preferentially removing reads of the most common length. When necessary, we adjust the lengths of previously found reads by a few bases to match the coverage vector as closely as possible.

Initially, we extract reads in end-to-end sets of the form (a,b,n) where a and b are the starting and ending indices in the coverage vector and n is the number of end-to-end reads. Each read set must satisfy

$$n \cdot l_{min} \leq (b-a) \leq n \cdot l_{max}$$

where l_{min} and l_{max} are the minimum and maximum lengths in the read distribution, respectively. Each time we find a new read (b,c), we search for an existing read set matching (a,b,n) and update it to $(a,c,n+1)$. If no such read exists, we add a new read set $(b,c,1)$.

We define two helper functions $extend(x_0,x_1)$ and $shorten(x_0,x_1)$.

$extend(x_0,x_1)$ searches for a read set of the form (a,x_0,n) satisfying

$$n \cdot l_{min} \leq (x_1-a) \leq n \cdot l_{max}$$

and updates it to (a,x_1,n) and decrements the coverage vector in the range $[x_0,x_1)$ by 1.

$shorten(x_0,x_1)$ searches for a read set of the form (a,x_1,n) satisfying

$$n \cdot l_{min} \leq (x_0-a) \leq n \cdot l_{max}$$

and updates it to (a,x_0,n) and increments the coverage vector in the range $[x_0,x_1)$ by 1.

These functions allow us to adjust previous reads by small amounts to fit in later reads. The read extraction algorithm works as follows:

Last $start$ and end be the indices of the first and last nonzero elements in the coverage vector, respectively. We find a and b such that $cov[i]>0 \forall i \in [start,a)$, $cov[a]=0$ and $cov[i]=0 \forall i \in [a,b)$, $cov[b]>0$.

Special end case: if $a=end<start+l_{min}$, we first attempt to run $extend(start,a)$. If unsuccessful, we decrement the bases in the coverage vector in the range $[start,a)$ but do not add a new read.

If $a \geq l_{mode}$, we add a new read $(start,start+l_{mode})$ and update the coverage vector.

Otherwise, we attempt to run $extend(start,a)$. If unsuccessful, we attempt to run $shorten(a,b)$. If this is also unsuccessful, we do one of the following:

1. If $(a-start) \geq l_{min}$, we add a new read $(start,a)$ and update the coverage vector.
2. If $\frac{l_{min}}{2} \leq (a-start) < l_{min}$, we add a new read $(start,start+l_{mode})$ and update the coverage vector.
3. If $(a-start) < \frac{l_{min}}{2}$, we decrement the bases in the coverage vector in the range $[start,a)$ but do not add a new read.

We then update $start$ and end and repeat until the coverage vector is empty.

Supplementary Note 3

Weighted k-mer recall. We assess fidelity by measuring weighted k-mer recall (WKR), a component of the KC score developed by Li et al. (30) to assess transcriptome assemblies. WKR measures the degree to which an assembly recovers k-mers from the true simulated transcriptome, weighted by abundances of simulated transcripts containing the k-mer. For a k-mer r, its frequency profile $p(r)$ is defined as:

$$p(r) = \frac{\sum_{t \in T} n(r,t)c(t)}{\sum_{t \in T} n(t)c(t)}$$

where T is the simulated transcriptome and for each transcript $t \in T$:

- $n(r,t)$ is the number of times r occurs in t,
- $n(t)$ is the total number of k-mers in t, and
- $c(t)$ is the coverage of t.

Letting $R(T)$ be the set of all k-mers in transcriptome T:

$$WKR = \sum_{r \in R(T)} p(r)$$

WKR is defined with respect to the true transcriptome T, which we obtain from Flux Simulator's output. The GEUVADIS sample is not considered here, since it is not simulated. Figure 7 shows that WKR is largely unchanged after Boiler compression for various k-mer length settings.

It also shows that the difference in WKR is more pronounced for Cufflinks than for StringTie. Table 10 shows the WKR for $k=15$ for all datasets. Overall, the differences are slight, with the biggest difference at $k=15$ being an increase of 0.4% for the paired-end 2.5M-read *D. melanogaster* sample.

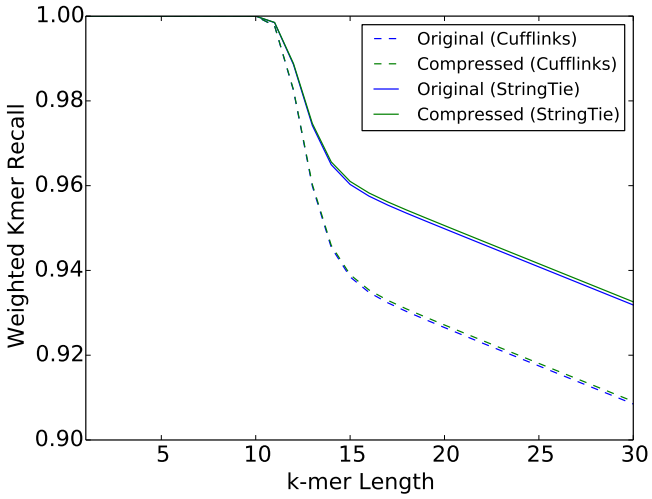

Figure 7. WKR with varying k-mer length for simulated Drosophila 10M paired-end reads, assembled with Cufflinks and StringTie.

Supplementary Note 4

Tripartite Score We developed a third scoring algorithm to compare the accuracy of alignments before and after compression, which we call the tripartite score. There are two versions of this score, strict and loose.

We first construct a tripartite graph containing a node for each transcript in the cufflinks output for the alignments both before and after compression, as well as for each transcript in the reference transcriptome. We add a connecting edge from each transcript from the original set to the best-matching transcript from the reference set, determined using the transcript scoring method described previously. Similarly, we add an edge from each transcript in the compressed set to the best match from the reference set of transcripts.

For the strict tripartite score, we take all the nodes from the set of reference transcripts that are connected to a single node A_i from the set of original transcripts and a single node B_i from the set of compressed transcripts. The final score is the average of the transcript scores for every pair A_i, B_i.

For the loose tripartite score, we take all the nodes from the set of reference transcripts that are connected to at least one node from the set of original transcripts and at least one node from the set of compressed transcripts. Let A_i be the original transcript with the highest score compared to the reference node, and let B_i be the compressed transcript with the highest score compared to the reference transcript. The final score is the average of the transcript scores for every pair A_i, B_i.

Tables 11 and 12 show the tripartite scores alongside the percentage of transcripts from the original and compressed set of transcripts that contribute to the score.

Table 10. WKR.

Dataset	Cufflinks		StringTie	
	Original	Compressed	Original	Compressed
Drosophila, Simulated Unpaired				
0.5M	0.745	0.745 (-0.1%)	0.627	0.627 (+0.1%)
1M	0.859	0.858 (-0.1%)	0.784	0.784 (+0.0%)
2.5M	0.925	0.923 (-0.2%)	0.902	0.902 (+0.0%)
5M	0.949	0.949 (+0.0%)	0.937	0.938 (+0.0%)
10M	0.957	0.958 (+0.1%)	0.955	0.955 (+0.0%)
20M	0.962	0.962 (+0.1%)	0.961	0.961 (+0.0%)
Drosophila, Simulated Paired				
0.5M	0.848	0.848 (-0.0%)	0.782	0.782 (+0.0%)
1M	0.909	0.908 (-0.1%)	0.882	0.883 (+0.1%)
2.5M	0.929	0.933 (+0.5%)	0.942	0.942 (+0.0%)
5M	0.948	0.945 (-0.3%)	0.957	0.958 (+0.0%)
10M	0.938	0.939 (+0.1%)	0.960	0.961 (+0.1%)
20M	0.936	0.937 (+0.0%)	0.964	0.964 (+0.0%)
Human, Simulated				
20M	0.882	0.883 (+0.2%)	0.933	0.931 (-0.2%)
40M	0.900	0.908 (+0.9%)	0.934	0.930 (-0.4%)

Table 11. Tripartite score for Cufflinks transcripts.

Dataset	Strict			Loose		
	Score	% True	% Comp	Score	%True	% Comp
Drosophila, Simulated Unpaired						
0.5M	1.000	27.5	27.5	0.984	43.9	43.9
1M	0.998	23.0	23.1	0.986	39.5	39.5
2.5M	0.995	23.8	23.8	0.991	34.8	34.8
5M	0.997	23.9	23.9	0.995	30.6	30.6
10M	0.997	23.1	23.1	0.994	28.0	28.0
20M	0.996	22.3	22.3	0.993	26.9	26.9
Drosophila, Simulated Paired						
0.5M	0.994	45.1	45.0	0.984	59.2	59.1
1M	0.990	35.5	35.4	0.983	47.1	46.9
2.5M	0.981	31.5	31.4	0.978	39.4	39.2
5M	0.978	27.9	27.7	0.975	34.4	34.2
10M	0.973	25.3	25.2	0.967	31.7	31.6
20M	0.972	23.3	23.2	0.965	29.5	29.5
Human, Simulated						
20M	0.986	16.7	16.7	0.978	22.6	22.6
40M	0.984	14.4	14.4	0.973	20.2	20.2

Table 12. Tripartite score for Stringtie transcripts.

Dataset	Strict			Loose		
	Score	% True	% Comp	Score	%True	% Comp
Drosophila, Simulated Unpaired						
0.5M	0.999	36.8	36.6	0.984	54.1	53.9
1M	0.998	37.8	37.7	0.992	53.6	53.4
2.5M	0.996	36.1	36.0	0.991	48.1	47.9
5M	0.998	31.5	31.5	0.997	39.8	39.7
10M	0.997	27.4	27.4	0.995	34.7	34.6
20M	0.997	23.2	23.1	0.995	30.8	30.7
Drosophila, Simulated Paired						
0.5M	0.999	39.8	39.7	0.991	54.5	54.4
1M	0.998	36.9	36.8	0.992	48.8	48.7
2.5M	0.996	32.0	32.1	0.995	40.3	40.3
5M	0.994	26.4	26.4	0.992	34.2	34.2
10M	0.993	23.1	23.1	0.990	31.0	31.0
20M	0.992	21.6	21.6	0.989	29.2	29.3
Human, Simulated						
20M	0.995	15.4	15.4	0.991	21.5	21.5
40M	0.989	12.8	12.9	0.985	18.9	19.1

REFERENCES

28. Caprara A, Kellerer H, Pferschy U (2000) A PTAS for the multiple subset sum problem with different knapsack capacities. *Information Processing Letters*, **73(3)**, 111-118.
29. Garey MR, Johnson DS (1978) "Strong" NP-Completeness Results: Motivation, Examples, and Implications. *Journal of the ACM*, **25(3)**, 499–508.
30. Li B, Fillmore N, Bai Y, Collins M, Thomson J, Stewart R, Dewey C (2014) Evaluation of de novo transcriptome assemblies from RNA-Seq data. *Genome Biology*, **15(12)**, 553.