# Supplementary Note

## Methods

The key observation used by DeeZ, especially for low error sequencing technologies such as Illumina, is that the vast majority of the nucleotide differences between each read, and the locus on the reference genome it is mapped to, is shared with other reads mapping to the same locus. If our goal is simply to compress the sequence content of a set of reads by encoding the nucleotide differences of each read and the locus it maps to, the only additional information we would need to store is the mapping locus of the read, which, collectively can be compressed very efficiently by the use of run length encoding. As a result, the number of bits used to encode the differences between each read and its mapping loci would dominate the overall encoding. DeeZ aims to lower the cost of representing differences between reads and their mapping locus through a "collective" encoding. The reads mapped to a particular genome region are locally assembled into contigs and for each read DeeZ only encodes the particular locus of each read within the contig (in case there are some rare read errors or complex allelic differences, additional information is encoded), and encodes the differences between the contig and the reference genome once. Such a saving is especially noticeable on a high-coverage data set, where a single difference (SNV or indel) between the donor genome and the reference will not be redundantly encoded in every read that includes that difference; see Figure 1.

DeeZ represents the donor genome based on the limited assembly of the mapped reads as follows. First, DeeZ partitions the reads into blocks according to their mapping loci, where each block contains a fixed number of reads (the default setting is 1 million–which, on a 40x coverage data set, corresponds to about 25 KB of genome). Then DeeZ processes each block independently from the others and constructs a contig which (i) covers all of the reads that map to the block, and (ii) has the fewest number of edit operations with respect to the reads mapping to the block. In order to achieve this, DeeZ starts with a draft contig, which satisfies property (i) but not (ii)–this contig happens to be the substring of the reference genome which covers all reads mapping to the given block. Then, DeeZ edits the draft contig in a manner that the number of edit operations between the contig and the reads is minimized, as follows.

Given a draft contig $Z$, we say potential mutation $M_i(x)_Z$ (in the donor genome) substitutes $Z[i]$ with nucleotide $x$, where $x$ can represent either a single nucleotide or an indel. We say that there is "substantial evidence" for $M_i(x)_Z$ if and only if the number of the read mappings supporting $M_i(x)_Z$ is larger than that supporting any other $M_i(y)_Z$ for $y \neq x$ or that supporting no

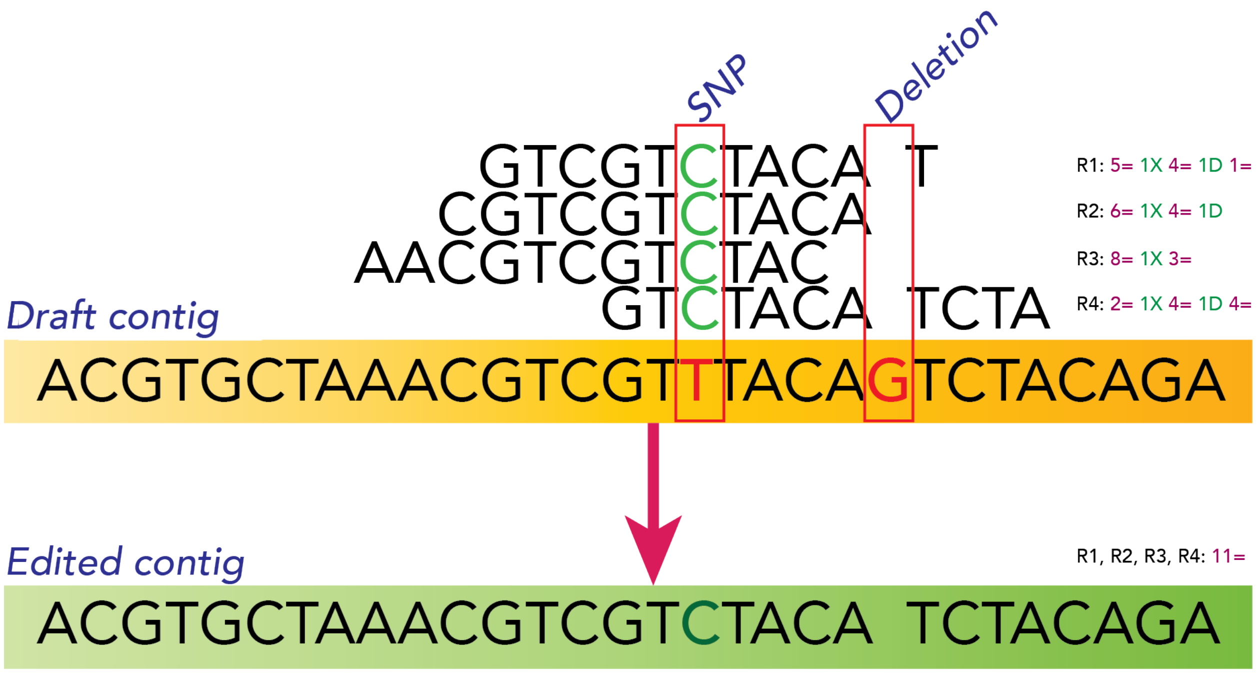

Figure 1: DeeZ approach for draft contig editing. Each common mutation supported by the reads is encoded in the resulting contig. The CIGAR strings of mappings are provided on the right side of each read (= represents a match, X a mismatch and D a deletion). After modifying the underlying contig, the originally complex CIGAR strings become simple (only a sequence of = symbols), which boosts the compression significantly.

mutation at all. Once DeeZ identifies all potential mutations with substantial evidence, it edits the draft contig $Z$ to include all such mutations and obtain contig $W$.

**Theorem 1.** *Given the reference genome and the set of read mappings, a contig $W$ obtained by DeeZ satisfies both properties (i) and (ii).*

*Proof.* It is trivial to observe that $W$ satisfies (i). To prove that it satisfies (ii), let $f(W)$ denote the number of edit operations necessary to map the reads from the block to contig $W$. Suppose that there is another contig $W'$, for which $f(W') < f(W)$ and $f(W') \leq f(W'')$ for every other contig. Then, $W' \neq W$ and there should exist a position $i$ on the draft contig $Z$ such that if $x_W$ and $x'_W$ are respectively the nucleotide or indel in $W$ and $W'$ that corresponds to $Z[i]$, then $x_W \neq x'_W$. Obviously $M_i(x_W)_Z$ has more support than $M_i(x'_W)_Z$; otherwise, the DeeZ would have replaced $Z[i]$ with $x'_W$. But this means that by replacing $x'_W$ with $x_W$ in $W'$, one would end up with a new contig $W''$ for which $f(W') > f(W'')$, implying a contradiction. $\square$

Thus, by applying the above procedure on the draft contig, DeeZ is able to obtain the optimal contig $W$ which satisfies the above two properties. Since the mapping information and the draft contig are known in advance, this procedure requires linear time and can be performed very fast in practice (unlike typical *de novo* assembly tasks). In addition, DeeZ's block-based design enables one to instantly seek any region in the genome and extract all reads mapping to the region without having to decompress the entire data set.

DeeZ is currently designed to work with both SAM and BAM files. Typical SAM files contain large amounts of additional metadata in addition to the basic read alignment (mapping) information. Such metadata is stored in a different field of the "SAM record", as described in the SAM file reference document [1]. DeeZ groups each field of the SAM file in a separate stream and compresses it independently for each stream. For most fields LZ77 [2] (gzip) is used as the compression method of choice since (i) it is fast, and (ii) it has a small overhead as it does not require any *a priori* information about the the data set for decompressing a block at any position in the file. In contrast, AC usually needs *a priori* model information (i.e. context) from previously compressed blocks, in order to decompress each given block; such a model needs to be represented for each block independently in order to provide random access capability. Although AC based tools above perform some kind of implicit assembly themselves, they do it by constructing complex genome models with a large memory footprint and thus are unable to support random access capability.

**Reads and CIGAR strings.** In the SAM format CIGAR strings describe the read alignment information needed to correctly map the read to the reference genome (Figure 1). DeeZ internally modifies the CIGAR strings to reflect the changes on the edited contigs and to accommodate indels precisely (since original CIGAR string does not display the insertion details). DeeZ also stores the differences between the resulting contigs and draft contigs (i.e. the reference genome) for accurate reconstruction of the read contents. With these changes, the need of storing reads vanishes completely, since we can reconstruct each read from the resulting contig, its differences from the draft contig and read's corresponding CIGAR string. Note that, because of this encoding scheme DeeZ uses the reference genome for decompression purposes as well.

**Read names.** Most sequencers produce unique read identifiers, subsequently referred to as read names, which contain, in addition to the unique read ID, information about the sequencing process and sequencing hardware. This means that many read names share significant amount of information. Thus, we divide each read name into tokens, and compare each token with the token at the same position in the previously processed read name. A new token is encoded only if it differs from the previously processed token. In this way, we are able to significantly decrease the size of read name stream, which is one of the main space consumers in the original SAM file.

**Quality scores.** The quality scores are the main obstacle in compression of any NGS data, due to their higher variability and larger alphabet [3], especially compared to the remainder of a SAM file (for a detailed discussion, please refer to the Results section). Thus, for each quality score string, we first preprocess it by stripping the trailing sequence of usually low-quality values at the end of the string, as proposed in [4]. By doing this, we can decrease the length of quality score string while being able to easily reconstruct it during the decompression phase. After that, we pass the string to the simple order-2 arithmetic coder. While more effective schemes for quality score compression exist (e.g. [4]), we found that such schemes require keeping track of various context lengths and build complex models. These context models require high amount of metadata for each block, which are constructed in reference to the previously processed blocks; as a result, in order to decompress the quality score string of a particular block, the entire set of previously processed blocks may need to be decompressed. For users in need of high compression ratios but not random access capability (for quality scores), DeeZ provides the option of using the AC model from sam_comp [4]. In such cases, the users can still seek through the compressed file, but the quality scores will be not available (although the quality scores can still be obtained by performing full decompression of the compressed file). In addition to this scheme, DeeZ also provides a lossy quality compression scheme as proposed earlier by the SCALCE compression tool [5], which can improve the compression factor by an order of magnitude without any significant impact on a

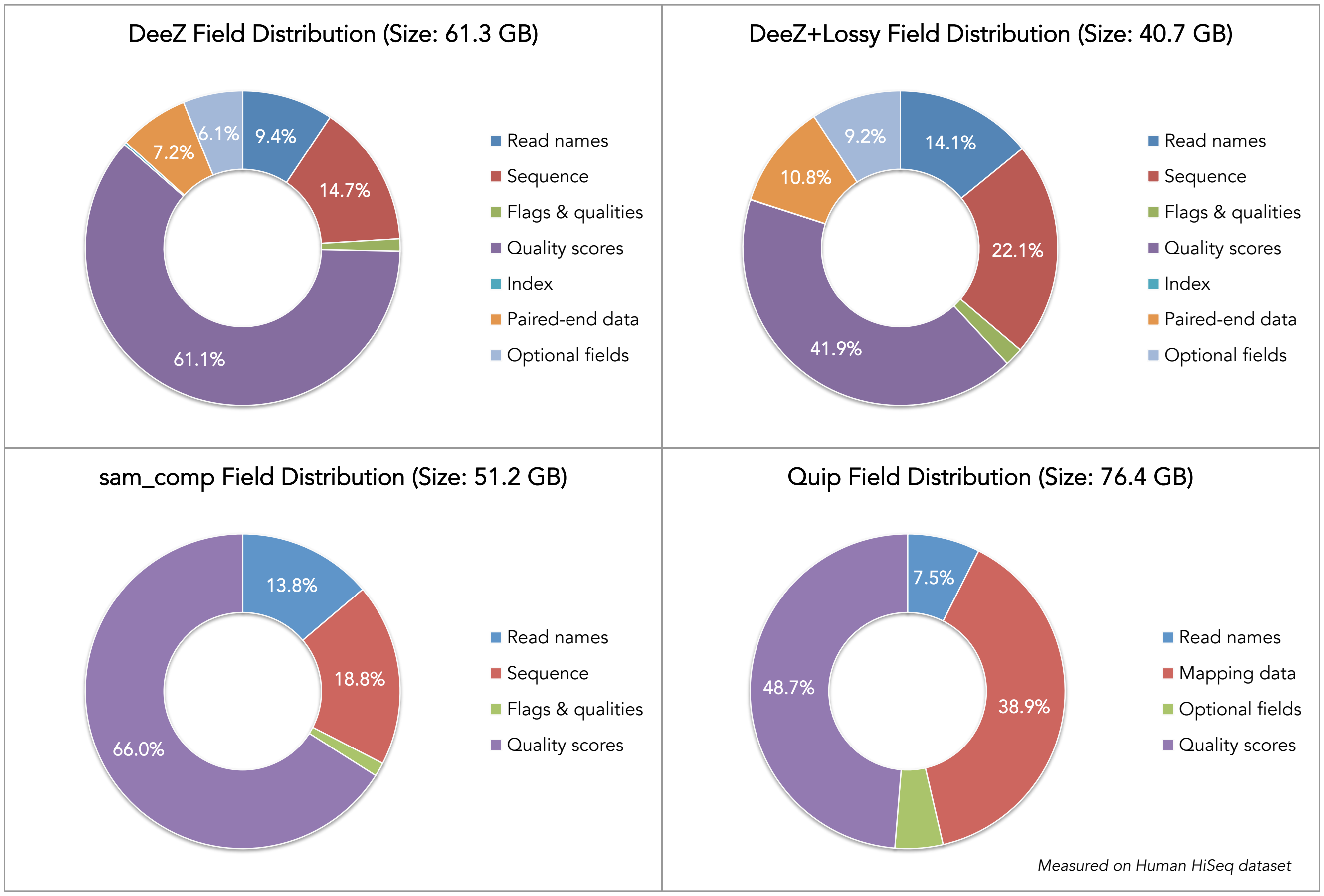

Figure 2: Quality score footprint in compressed files in comparison to other fields (measured on Human HiSeq dataset). Note that different tools internally organize the fields in different manner, thus the difference between chart sections.

typical downstream analysis.

**Mapping locations.** We use delta encoding for compressing mapping locations. In the high coverage samples, delta encoding will represent the mapping location of each read with a small integer (usually either 0 or 1), making the data highly suitable for a further order-0 arithmetic encoding.

**Other features.** For all streams, unless otherwise stated, we use the simple Lempel-Ziv 77 (LZ77) scheme to perform the compression on processed data. This is mainly because our preprocessing and grouping of the SAM fields is designed to increase a locality of reference for each stream, which causes a huge performance boost for LZ77. In addition, LZ77 provides both fast compression and decompression with little overhead, which allows fast block seeking through the compressed file. By default, DeeZ will use multiple threads for significantly improving compression

]

and decompression speed (especially the arithmetic coding part, which is an order of magnitude slower than LZ77 family of algorithms, particularly during decompression).

DeeZ stores the flag statistics of the mapped reads for easy and fast retrieval, and supports reading BAM files as well. Finally, DeeZ supports sorting of the input SAM file with respect to the mapping loci in case the file is unsorted.

## Experimental Results

The following data sets were used for evaluating the performance of DeeZ:

- *Pseudomonas aeruginosa* RNA-Seq library (51 bp, sequenced at 700x)
- *Escherichia coli* DH10B MiSeq sample (150bp, file MiSeq_Ecoli_DH10B_110721_PF)
- Human K562_cytosol_LID8465 RNA-seq sample (75bp, accession ID: ERX283488)
- Human NA12878 HiSeq DNA sample (100bp, sequenced at 40x, file NA12878_S1)

*P. aeruginosa* data set was mapped with bwa 0.7 mapper, in order to produce a valid SAM file. Other data sets were pre-mapped and publicly available as mapped BAM files. All data files are valid SAM files with the header and a significant number of unmapped reads (around 1%). The SAM files were sorted by the mapping location coordinate, contained paired-end information, and included several optional fields.

We compared DeeZ with the following compression tools:

1. gzip v1.3.12
2. SAMtools v0.1.19
3. CRAM Tools v2.0 [6]
4. Scramble v1.13.7
5. Quip v1.1.6
6. sam_comp v0.7 and v0.8

7. goby v2.3.4

SAMtools is the current standard tool for creating and compressing SAM files. Quip is primarily a FASTQ file compressor with additional support for compressing SAM files. sam_comp is an arithmetic coding based compressor, not able to compress SAM headers, paired-end information and optional fields. sam_comp and Quip support multiple modes: (i) normal mode, where reads are compressed using arithmetic coding with a specialised context model, and (ii) reference-based mode, where only differences between the reads and the reference are encoded.

CRAM Tools and Scramble are reference-based compressors which implement CRAM format. They are not lossless, i.e. certain fields in the SAM format are either changed or deleted by CRAM Tools or Scramble.

Note that in all of our the experiments, each tool was run in its default mode, unless otherwise stated. When possible, we chose the options that forces each tool to compress in a lossless fashion as many SAM fields as possible (since some of them discard or modify some fields by default). We provide detailed invocation parameters for each tool in the Appendix B.

Since sam_comp does not support compressing all fields, we compare it only with DeeZ with the option of compressing those fields which sam_comp supports. The compression and timing results are provided in tables 1 and 2.

As it can be seen from the tables, DeeZ outperforms all of the above mentioned tools, with exception of sam_comp, whose compression performance is comparable to DeeZ. However not only sam_comp does not provide random access ability but also it does not compress all fields in the SAM format. (See Table 3 for random access performance of DeeZ in comparison to the only other two methods that provide this capability: SAMtools and Scramble.) In addition, we were not able to decompress any of the files compressed with sam_comp (with the exception of *E. coli* MiSeq dataset). Note that in the HiSeq data set Quip performs slightly better than DeeZ with the default settings due to its use of high-order AC compression for quality scores, which is well suited for a large file. However Quip does not provide random access ability either. As can be expected, when DeeZ uses sam_comp AC quality score compression scheme, it outperforms Quip on this data set, while still providing partial random access ability. DeeZ also provides the fastest compression speed in the first, second and the fourth data sets. In the third data set, DeeZ compression speed is lower due to eukaryotic RNA-Seq including many reads mapping to exon/intron junctions. Reads

Table 1: Compression ratios provided by all tested tools. File sizes are reported in megabytes. One megabyte equals $1024 \times 1024$ bytes.

| | | | *P. aeruginosa* RNA-Seq | | *E. coli* MiSeq | | Human RNA-Seq | | Human HiSeq | |
|---|---|---|---|---|---|---|---|---|---|---|
| Tool | RA | Lossless | Size | Ratio | Size | Ratio | Size | Ratio | Size | Ratio |
| Original size | | | 19,008 | 1.00 | 5,321 | 1.00 | 72,398 | 1.00 | 437,589 | 1.00 |
| **gzip** | N | Y | 3,210 | 5.92 | 1,279 | 4.16 | 12,236 | 5.92 | 99,180 | 4.41 |
| **SAMtools** | Y | Y | 3,340 | 5.69 | 1,341 | 3.97 | 13,119 | 5.52 | 106,596 | 4.11 |
| **CRAM Tools** | N[3] | N | 3,967 | 4.79 | N/A | N/A | 9,898 | 7.31 | 74,564 | 5.87 |
| **Scramble** | Y | N | N/A | N/A | 1,406 | 3.79 | 10,063 | 7.19 | 75,784 | 5.77 |
| **goby** | Y | N | N/A | N/A | N/A | N/A | 11,757 | 6.16 | N/A | N/A |
| **Quip** (non-reference based) | N | Y | 2,561 | 7.42 | 1,049 | 5.07 | 10,601 | 6.83 | 78,221 | 5.59 |
| **Quip** (reference-based) | N | Y | 2,181 | 8.72 | 1,135 | 4.69 | 8,271 | 8.75 | 61,905 | 7.07 |
| **DeeZ** | Y | Y | 1,921 | 9.89 | 831 | 6.40 | 8,010 | 9.04 | 62,808 | 6.97 |
| **DeeZ** (partial random access[2]) | P[2] | Y | 1,828 | 10.40 | 788 | 6.76 | 7,615 | 9.51 | 58,879 | 7.43 |
| **DeeZ** (lossy quality scores) | Y | N | 1,343 | 14.15 | 513 | 10.36 | 5,157 | 14.04 | 41,701 | 10.49 |
| **sam_comp**[1] (non-reference based) | N | N | 1,473 | 12.91 | 668 | 7.96 | 6,781 | 10.68 | 52,389 | 8.35 |
| **sam_comp**[1] (reference based) | N | N | N/A | N/A | 678 | 7.85 | 6,724 | 10.77 | 51,733 | 8.46 |
| **DeeZ** (sam_comp fields only) | Y | N | 1,623 | 11.71 | 720 | 7.39 | 7,136 | 10.14 | 54,435 | 8.04 |
| **DeeZ** (sam_comp fields only, partial random access[2]) | P | N | 1,531 | 12.41 | 677 | 7.86 | 6,746 | 10.73 | 50,536 | 8.66 |

[1] sam_comp v0.8 (results coloured in blue) was used to compress *P. aeruginosa* and Human RNA-Seq data set. In other cases, sam_comp v0.7 was used.

[2] Quality scores are compressed via sam_comp model and thus are not randomly accessible–the other fields are.

[3] Although CRAM format supports indexing, CRAM Tools does not provide random access interface.
Scramble, C implementation of CRAM specification, does support random access.

Table 2: Time and memory usage needed for compression and decompression. All figures are in (H:)MM:SS format. All sizes are in megabytes, unless otherwise specified.

| | *P. aeruginosa* RNA-Seq | | | | *E. coli* MiSeq | | | | Human RNA-Seq | | | | Human HiSeq | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Compression | | Decompression | | Compression | | Decompression | | Compression | | Decompression | | Compression | | Decompression | |
| | Time | Memory | Time | Memory | Time | Memory | Time | Memory | Time | Memory | Time | Memory | Time | Memory | Time | Memory |
| gzip | 13:35 | 4 | 02:15 | 4 | 04:44 | 4 | 00:45 | 4 | 0:47:24 | 4 | 08:35 | 4 | 7:31:42 | 4 | 2:12:10 | 4 |
| SAMtools | 14:25 | 11 | 02:59 | 11 | 04:53 | 11 | 00:54 | 11 | 0:53:13 | 11 | 10:59 | 11 | 6:49:42 | 11 | 2:11:33 | 11 |
| CRAM Tools[1,2] | 59:07 | >1,120 | 21:22 | >1,120 | N/A | N/A | N/A | N/A | 2:30:29 | >1,220 | 52:02 | >1,220 | 14:18:38 | >1,330 | 5:28:58 | >1,330 |
| Scramble[1] | N/A | N/A | N/A | N/A | 06:54 | 166 | 02:06 | 89 | 0:36:04 | 1,997 | 11:28 | 1,989 | 6:36:41 | 417 | 1:59:39 | 162 |
| goby[2,5] | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | 5:02:57 | >8,600 | 5:25:09 | >8,600 | > 1 day | N/A | N/A | N/A |
| Quip[3] (non-reference based) | 14:53 | 653 | 16:57 | 680 | 05:32 | 664 | 05:53 | 689 | 1:00:31 | 846 | 57:30 | 870 | 6:45:12 | 731 | 7:24:07 | 748 |
| Quip[3] (reference based) | 14:54 | 654 | 17:20 | 681 | 05:51 | 665 | 05:45 | 689 | 0:56:10 | 1,918 | 57:04 | 1,942 | 8:34:38 | 1,805 | 7:27:23 | 1,822 |
| DeeZ | 12:01 | 1,350 | 10:39 | 1,512 | 03:49 | 1,866 | 04:09 | 1,880 | 1:18:10 | 2,202 | 48:42 | 2,074 | 5:49:48 | 2,129 | 6:34:26 | 2,615 |
| DeeZ (partial random access) | 13:27 | 1,777 | 12:20 | 2,048 | 04:34 | 2,285 | 05:01 | 2,297 | 1:27:22 | 2,681 | 54:51 | 2,579 | 6:50:33 | 2,532 | 7:42:04 | 3,216 |
| sam_comp[4] (non-reference based) | 13:05 | 474 | N/A | N/A | 04:45 | 334 | 05:35 | 334 | 0:51:12 | 474 | 56:30 | 474 | 6:03:46 | 334 | N/A | N/A |
| sam_comp[4] (reference based) | N/A | N/A | N/A | N/A | 04:43 | 338 | 05:24 | 338 | 0:50:29 | 717 | 56:54 | 717 | 6:00:33 | 576 | N/A | N/A |

[1] CRAM Tools and Scramble decompressed SAM file was missing 1 GB in the first dataset, 4 GB in the third and 17 GB in the fourth data set

[2] CRAM Tools and goby are written in Java, and thus the virtual memory usage is heavily affected by the Java runtime (JRE). On our test machine, JRE was using around $> 10$ GB of virtual memory. Thus, in those cases we opted to report residential memory usage, which, although being less accurate than the virtual memory usage, provides better insight in the memory usage of Java tools.

[3] Quip decompressed SAM file was missing 1 GB in the fourth data set

[4] sam_comp v0.7 was not able to decompress any of the data sets above, with the exception of *E.coli* dataset.
sam_comp v0.8 (results colored in blue) succeeded in decompressing the Human RNA-Seq dataset.

[5] goby successfully compressed only RNA-Seq dataset. HiSeq dataset compression took more than one day, and thus we decided to omit its results due to the time constraints.

Table 3: Random access performance for three tools which support it. All figures are in (MM:)SS format. Second table indicates the index size (in megabytes) and additional pre-processing (i.e. index building) times needed for SAMtools and Scramble.

| | chr5 | | | | chrY 10,000–20,000 | | | | chr14 107,349,000–107,349,540 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Human RNA-Seq | | Human HiSeq | | Human RNA-Seq | | Human HiSeq | | Human RNA-Seq | | Human HiSeq | |
| | Time | # records | Time | # records | Time | # records | Time | # records | Time | # records | Time | # records |
| SAMtools | 60 | 9,116,311 | 08:26 | 71,141,857 | 1 | 26 | 1 | 759 | 1 | 0 | 1 | 0 |
| Scramble | 63 | 9,116,311 | 08:19 | 71,141,857 | 1 | 26 | 3 | 759 | 5 | 0 | 1 | 0 |
| DeeZ | 119 | 9,116,311 | 16:30 | 71,141,857 | 5 | 26 | 14 | 759 | 1 | 0 | 3 | 0 |
| DeeZ (without qualities) | 84 | 9,116,311 | 13:54 | 71,141,857 | 3 | 26 | 7 | 759 | 1 | 0 | 3 | 0 |

| | Human RNA-Seq | | Human HiSeq | |
|---|---|---|---|---|
| | Time | Size | Time | Size |
| SAMtools | 04:04 | 5.51 | 30:10 | 8.63 |
| Scramble | 02:55 | 0.39 | 05:10 | 2.03 |

mapping to junctions cause DeeZ spend more time analyzing and editing draft contigs.[1]

DeeZ's decompression speed is also on par with or better than its competitors with the exception of SAMtools and Scramble. This is due to the LZ77 decompression scheme employed by SAMtools and Scramble being much faster than AC decompression, which DeeZ uses for the quality scores. This issue is especially acute in the whole chromosome retrieval task in Table 3, where decompression of quality scores dominate the time for random access. In case the quality scores are not needed for an FRA task, the performance of DeeZ gets improved.

**Discussion: Quality scores.** Quality scores usually consume the largest portion of a compressed file, due to their high entropy compared to other fields of a SAM file. Figure 2 depicts the size distribution of various SAM fields for Quip, sam_comp and DeeZ in their default settings; as can be seen, even with powerful AC methods, quality scores typically occupy more than half of the file size. As a result DeeZ provides an optional lossy quality transformation as described earlier [5]. In this way, DeeZ is able to significantly decrease the compressed file size in its default mode (without even using an advanced model), as well as the portion of the file occupied by the quality scores without significantly impacting standard downstream analyses.

## References

1. The SAM/BAM Format Specification Working Group. Sequence Alignment/Map Format Specification. `http://samtools.github.io/hts-specs/SAMv1.pdf`. Accessed: 2014-03-24.

2. Ziv, J. & Lempel, A. A universal algorithm for sequential data compression. *IEEE TRANSACTIONS ON INFORMATION THEORY* **23**, 337–343 (1977).

[1]Obviously the running times may vary due to I/O utilization and caching. Up to 25% variation between two runs of the same method, on the same machine and same data set is possible.

3. Wan, R., Anh, V. N. & Asai, K. Transformations for the compression of fastq quality scores of next-generation sequencing data. *Bioinformatics* **28**, 628–635 (2012).

4. Bonfield, J. K. & Mahoney, M. V. Compression of FASTQ and SAM Format Sequencing Data. *PLoS ONE* **8**, e59190 (2013).

5. Hach, F., Numanagic, I., Alkan, C. & Sahinalp, S. C. SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics* **28**, 3051–3057 (2012).

6. Hsi-Yang Fritz, M., Leinonen, R., Cochrane, G. & Birney, E. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Research* **21**, 734–740 (2011).

7. Margulies, M. *et al.* Genome sequencing in microfabricated high-density picolitre reactors. *Nature* **437**, 376–380 (2005).

8. Li, H. *et al.* The sequence alignment/map format and samtools. *Bioinformatics* **25**, 2078–2079 (2009).

9. Bentley, D. R. *et al.* Accurate whole human genome sequencing using reversible terminator chemistry. *Nature* **456**, 53–59 (2008).

10. McKernan, K. J. *et al.* Sequence and structural variation in a human genome uncovered by short-read, massively parallel ligation sequencing using two-base encoding. *Genome Research* **19**, 1527–41 (2009).

11. Pushkarev, D., Neff, N. F. & Quake, S. R. Single-molecule sequencing of an individual human genome. *Nature Biotechnology* **27**, 847–850 (2009).

12. Eid, J. *et al.* Real-time DNA sequencing from single polymerase molecules. *Science* **323**, 133–138 (2009).

13. Kozanitis, C., Saunders, C., Kruglyak, S., Bafna, V. & Varghese, G. Compressing Genomic Sequence Fragments Using SlimGene. *Journal of Computational Biology* **18**, 401–413 (2011).

14. Cox, A. J., Bauer, M. J., Jakobi, T. & Rosone, G. Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform. *Bioinformatics* **28**, 1415–1419 (2012).

15. Yanovsky, V. ReCoil - an algorithm for compression of extremely large datasets of DNA data. *Algorithms Mol Biol* **6**, 23 (2011).

16. Tembe, W., Lowey, J. & Suh, E. G-SQZ: compact encoding of genomic sequence and quality data. *Bioinformatics* **26**, 2192–2194 (2010).

17. Deorowicz, S. & Grabowski, S. Compression of DNA sequence reads in FASTQ format. *Bioinformatics* **27**, 860–862 (2011).

18. Jones, D. C., Ruzzo, W. L., Peng, X. & Katze, M. G. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic Acids Research* **40**, e171 (2012).

19. Bonfield, J. K. The scramble conversion tool. *Bioinformatics* (2014). URL http://bioinformatics.oxfordjournals.org/content/early/2014/07/01/bioinformatics.btu390.abstract.

20. Massie, M. *et al.* Adam: Genomics formats and processing patterns for cloud scale computing. Tech. Rep. UCB/EECS-2013-207, EECS Department, University of California, Berkeley (2013). URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-207.html.

21. Campagne, F., Dorff, K. C., Chambwe, N., Robinson, J. T. & Mesirov, J. P. Compression of structured high-throughput sequencing data. *PLoS ONE* **8**, e79871 (2013). URL http://dx.doi.org/10.1371%2Fjournal.pone.0079871.

22. Deorowicz, S. & Grabowski, S. Robust relative compression of genomes with random access. *Bioinformatics* 2979–2986 (2011).

23. Deorowicz, S., Danek, A. & Grabowski, S. Genome compression: a novel approach for large collections. *Bioinformatics* **29**, 2572–2578 (2013). http://bioinformatics.oxfordjournals.org/content/29/20/2572.full.pdf+html.

**Appendix A: Dataset locations**

The following data sets were used for evaluating the performance of DeeZ:

- *P. aeruginosa* RNA-Seq library (51 bp, sequenced at 700x)
  URL: `http://compbio.cs.sfu.ca/nwp-content/deez/P4.sam.dz`
- *E. coli* DH10B MiSeq sample (150bp, file MiSeq_Ecoli_DH10B_110721_PF)
  URL: `ftp://webdata:webdata@ussd-ftp.illumina.com/Data/SequencingRuns/DH10B/MiSeq_Ecoli_DH10B_110721_PF.bam`
- Human K562_cytosol_LID8465 RNA-seq sample (75bp, accession ID: ERX283488)
  URL: `http://www.ebi.ac.uk/arrayexpress/files/E-MTAB-1728/K562_cytosol_LID8465_TopHat_v2.bam`
- Human NA12878 HiSeq DNA sample (100bp, sequenced at 40x, file NA12878_S1)
  URL: `ftp://ftp.sra.ebi.ac.uk/vol1/ERA207/ERA207860/bam/NA12878_S1.bam`

We used following reference genomes:

- UCSC *H. Sapiens* hg19
  URL: `http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/chromFa.tar.gz`
- *P. aeruginosa* PAO1 chromosome
  URL: `http://www.ncbi.nlm.nih.gov/nuccore/110645304?report=fasta`
  or `http://compbio.cs.sfu.ca/nwp-content/deez/NC_002516.2.fa`
- *E. Coli* DH10B
  URL: `https://raw.githubusercontent.com/allanroscoche/PathTree/master/data/DH10B_WithDup_FinalEdit_validated.fasta`

**Appendix B: Tool invocation parameters**

1. gzip
   Compression: `gzip input.sam -c >input.gz`

Decompression: `gzip -d input.gz -c >input_dc.sam`

2. SAMtools v0.1.19
   Compression: `samtools view -bS input.sam >input.bam`
   Decompression: `samtools view -h input.bam >input.sam`

3. CRAM Tools v2.0
   Compression: `java -Xmx8g -jar cramtools-2.0.jar cram -I input.sam --capture-all-tags --input-is-sam -Q -n -R reference.fa >input.cram`
   Decompression: `java -Xmx8g -jar cramtools-2.0.jar bam -I input.cram --print-sam-header -R reference.fa >input_dc.sam`

4. Scramble v1.13.7
   Compression: `scramble -I sam -O cram -r reference.fa input.sam >input.scr`
   Decompression: `scramble -I cram -O sam -r reference.fa input.scr >input_dc.sam`

5. Quip v1.1.6
   - Non-reference based:
     Compression: `quip input.sam -c >input.qp`
     Decompression: `quip input.qp --output=sam -d -c >input_dc.sam`
   - Reference based:
     Compression: `quip -r reference.fa input.sam -c >input.qpr`
     Decompression: `quip --output=sam -d -r reference.fa input.qpr -c >input_dc.sam`

6. sam_comp v0.7 and v0.8
   - Non-reference based:
     Compression: `sam_comp <input.sam >input.zam`
     Decompression: `sam_comp -d <input.zam >input_dc.sam`
   - Reference based:
     Compression: `sam_comp -r reference <input.sam >input.zamr`
     Decompression: `sam_comp -r reference -d <input.zamr >input_dc.sam`

7. goby v2.3.4

   Compression:
   ```
   java -Xmx8g -jar goby.jar -m stc
   --preserve-all-mapped-qualities --preserve-all-tags
   --preserve-soft-clips --preserve-read-names
   -x AlignmentWriterImpl:permutate-query-indices=false
   -x SAMToCompactMode:ignore-read-origin=false
   -x MessageChunksWriter:codec=hybrid-1
   -x AlignmentCollectionHandler:enable-domain-optimizations=true
   -x MessageChunksWriter:compressing-codec=true -g reference.fa
   -i input.sam -o input.goby
   ```
   Decompression:
   ```
   java -Xmx8g -jar goby.jar -m cts -g reference.fa
   input.goby -o input_dc.sam.sam
   ```

8. DeeZ

   - Normal mode:
     Compression:
     ```
     deez -r reference.fa input.sam -c >input.dz
     ```
     Decompression:
     ```
     deez -r reference.fa input.dz -c >input_dc.sam
     ```
   - sam_comp mode:
     Compression:
     ```
     deez -r reference.fa input.sam -q1 -c >input.dzq
     ```
     Decompression:
     ```
     deez -r reference.fa input.dzq -c >input_dc.sam
     ```
   - Lossy quality mode:
     Compression:
     ```
     deez -r reference.fa input.sam -l30 -c >input.dzl
     ```
     Decompression:
     ```
     deez -r reference.fa input.dzl -c >input_dc.sam
     ```