SPRING: FASTQ compression - Supplementary Data

Shubham Chandak, Kedar Tatwawadi, Idoia Ochoa, Mikel Hernaez, Tsachy Weissman

Contents

1 Extended methods

1.1 FASTQ files

The FASTQ format Cock *et al.* (2010) is used to represent data obtained from a sequencing experiment. This data is not aligned to a reference genome and consists of reads, quality values and read identifiers. Every read and the corresponding metadata is represented by a block of four lines. The first line is the read identifier, the second line is the read itself and the fourth line is the quality value for each base in the read. The third line contains the symbol '+' to separate the read and the quality values. This line can contain some metadata or comments, but they are rarely used and hence most compressors (including SPRING) discard them (Numanagić *et al.*, 2016). The read is a string of DNA symbols (typically A, C, G, T and N, where N represents no call). The quality value represents the confidence in each base call, encoded in ASCII using the Phred scale (Illumina, a). The read identifiers store various fields related to the sequencing process, such as lane number, instrument name, etc. For paired-end sequencing, two FASTQ files are produced, with the i^{th} read in the first file being the pair of the i^{th} read in the second file. As an example, the first two reads for the two files in the ERP001775 dataset are shown in Figure 1 (note that the read identifiers for the

paired reads differ only in the last character).

File 1:

```
@ERR174324.1 HSQ1009_86:1:1101:1192:2116/1
ATTCNGTCACTTCTCACCAGGCCCCTCATTCAACACTGGGAATTAAAATTCGACATGAGATACCATGTCAATATATGGTATAAAATCTAAGATGTTGCCAC
+
CCCF#2ADHHHHHJJJIJJJJIJJJJJJJJGIJJJJJJJJIJJJIJJJJJGIJJJJJJJIJJJJJHHHHGHHFFFBEFFEEEEEEEEDDDDDDDEDDCCD>
@ERR174324.2 HSQ1009_86:1:1101:1079:2130/1
CAGANAGAGACTCTGTCTCAAAAAAACAAACAAACAAACAAACAAAAAGTCTTAGTCCCAGCACTTTGGGAGGCCGAGGCAGGCGGATCACAAGGTCAGGA
+
CCCF#2ADHFHHHJIJJJJJJJJJJJJJJJJJIJJJHIIJJJJJJJJIIIJJHHHHHHFFFFFDEEEDDDDDD?BDDDDDDDDDBDDDDDDDB>CCDD3
```

File 2:

```
@ERR174324.1 HSQ1009_86:1:1101:1192:2116/2
CTTTGATCACCATTACCTATAATGTCTCGCTTCCACTGACTTCTCAGTCTCTTCTGTGTCTCATCTTCTCAAGCAATTTTTTTCCAGCCCTACATCAATGT
+
CCCFFFFFHHHHHJJJJJIJJIDIJIJJJJJJJJJIJJJJJJGHGHIIIIHIIJJJIJIIJJJJJJIIJJIIIJI;DGIJIFD@CDDCDDDDDDDDDDDC
@ERR174324.2 HSQ1009_86:1:1101:1079:2130/2
NAAAGACTTTTTTTTTTTGAGACGGAGTCTCACTCTGTTGCCCAGGCTGGAGTGCAGTGGTGCGATCTTGGCTGACTGCAACCTCCGCCTCCCGGGTTCAA
+
#1=DFFFFHHHHHJJJJJICEGGHIBHHIIIJJJGEHHHHEHFFFEDCCDDDAACDDACC:?CB=BDDDCC?C?<A@CCACCB?CDDBDBDDDD>D#####
```

Figure 1: Paired-end FASTQ files in ERP001775 dataset.

1.2 SPRING

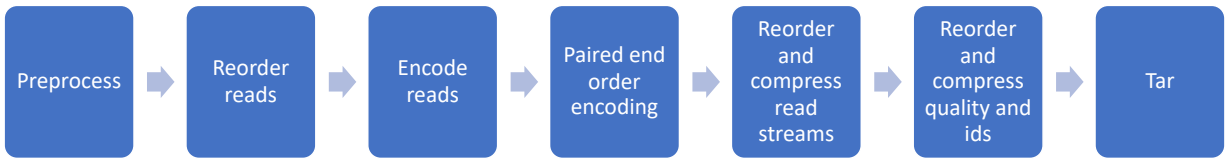

Figure 2: Compression flow for SPRING.

Figure 2 shows the overall compression flow for SPRING. Before describing the steps in more detail, we define some terms and discuss certain components:

- *BSC* - BSC (`https://github.com/IlyaGrebnov/libbsc/`) is a general-purpose compressor based on the Burrows-Wheeler Transform (BWT) built for achieving high compression ratios while being computationally efficient. We use BSC in a number of places in SPRING, with block length of 64 MB and `-p` (no preprocessing) flag activated.
- *Read compression* - There are two modes for read compression:
  - *Order preserving compression* - Compression preserving the order of reads in the FASTQ file. This is the default mode for SPRING.
  - *Order non-preserving (pairing only) compression* - In this mode, the order of reads in the FASTQ file is not preserved but the pairing information is preserved. Single end FASTQ files are arbitrarily reordered in this mode. Paired end FASTQ files are reordered such that the reads in file 1 remain in file 1, reads in file 2 remain in file 2, and the paired reads still remain paired. This mode is activated by the `-r` flag.

- *Quality value compression* - SPRING uses BSC for quality value compression and allows several options for quantization:
  - *Lossless* - default mode.
  - *QVZ* - QVZ (Malysa *et al.*, 2015) models quality values as a first-order Markov process with position-dependent transition probabilities. This allows QVZ to capture both the degradation of quality values at the end of the read and the correlation between the quality values within a read. QVZ first computes statistics for this model and generates quantization codebooks using a variant of Lloyd-Max algorithm. Note that we use QVZ only for quantization of quality values, which are then compressed using BSC. SPRING supports QVZ with mean square error distortion, where the user needs to specify the desired rate in bits/quality value. It was shown in Ochoa *et al.* (2017) that an average rate of 1 bit per quality value retains the performance of variant calling. This mode is activated by the `-q qvz qvz_rate` flag.
  - *Illumina binning* - SPRING supports Illumina's standardized 8-level binning scheme (Illumina, b) for lossy compression of quality values. The Illumina binning scheme maps the 40-level quality values to 8 levels by clustering together similar values. This mode is activated by the `-q ill_bin` flag.
  - *Binary thresholding* - SPRING supports binary quantization of quality values which was shown in Roguski *et al.* (2018) to significantly reduce the compressed size of quality values without sacrificing variant call accuracy (for high coverage datasets). The user needs to provide three parameters: *thr*, *high* and *low*. Quality values less than *thr* are quantized to *low* and quality values greater than or equal to *thr* are quantized to *high*. This mode is activated by the `-q binary thr high low` flag.
- *Read Identifier Compression* - As the read identifiers consist of heterogeneous but structured data, SPRING uses a token-based approach for their compression. The read identifiers are split into tokens and each token is encoded separately. For the numeric tokens, SPRING uses delta encoding if the difference from the previous value is small, otherwise it stores the token value as it is. For string type tokens, SPRING uses a special symbol to denote a perfect match with the previous value, otherwise the full string is stored. Finally these streams are compressed using an adaptive arithmetic encoder. The read identifier compression in SPRING is based on Samcomp (Bonfield and Mahoney, 2013) and GeneComp (Long *et al.*, 2017).

  For paired-end datasets, typically the corresponding identifiers in the two files differ only in a single character. During the preprocessing stage, we check if the identifiers have this structure. In that case, identifiers for only one of the files are compressed and those for the other file are reconstructed during the decompression.
- *Short read mode* - This mode supports short reads with read lengths up to 511. The short read compression in SPRING is based on HARC (Chandak *et al.*, 2018), with added support for paired end reads and several other improvements (discussed later in more detail). This mode is optimized for relatively accurate short reads containing substitution errors and is the default mode for SPRING.
- *Long read mode* - This mode supports long reads with read lengths up to 4.2 billion. Here we use BSC for read compression. This mode is activated by the `-l` flag and is always order preserving. Since the short read mode is designed for low error rates with most errors being subsitutions, the long read mode is also recommended for short read datasets with large number of indel errors.

Preprocess

In the long read mode, the reads, quality values and read identifiers are separated and compressed in blocks (reads and quality values using BSC, identifiers using specialized identifier compressor described above). By default, the block length for long reads is set to 10,000 reads. The read lengths are also stored as 32-bit integers in a separate stream which is compressed using BSC. Preprocessing is directly followed by the Tar stage for long reads.

In the order preserving mode, the quality values and read identifiers are compressed in blocks (quality values using BSC, identifiers using specialized identifier compressor). QVZ quantization is applied before quality compression if the corresponding flag is specified. By default the block length for short reads is set to 256,000 reads (see Section 3.8 for impact of block length on compression). The reads are written to temporary files after separating out the reads containing the character 'N'. The reads containing 'N' are considered directly in the "Encode reads" stage.

In the order non-preserving mode, the quality values and read identifiers are written to temporary files. The reads are handled exactly as in the order preserving mode.

If the Illumina binning or binary thresholding flag is activated, the qualities are binned before compression/writing to temporary file.

Reorder reads

This step in read compression is based on HARC (Chandak *et al.*, 2018), with several extensions and improvements. Here, we will provide a brief overview of the step, more details and parameters can be found in Chandak *et al.* (2018). In this step, SPRING reorders the reads so that they are approximately ordered according to their position in the genome. The reordering is done in an iterative manner: given the current read, SPRING tries to find a read which matches the prefix or the suffix of the current read with a small Hamming distance. To do this efficiently, a hash table is used which indexes the reads according to certain substrings of the read. SPRING makes the following improvements to this stage:

- While HARC searched for matching reads in only one direction (matching the suffix of the current read), SPRING looks for matches in both directions. This boosts read compression by 5-10% on most datasets (see Section 3.10).
- While HARC only supported fixed length reads of maximum length 255, SPRING adds support to variable length short reads of maximum length 511. For this, SPRING stores an array containing the read lengths, which is used to ensure that the Hamming distance between reads of different lengths is computed correctly.
- We observed that most of the time in the reordering stage is spent on a small fraction of remaining reads and the attempts to find matches to these reads usually fails. To save time in this step, SPRING introduces early stopping to this stage. Each thread maintains the fraction of unmatched reads in the last 1 million reads and stops looking for matches once this fraction crosses a certain threshold (50% by default). Since this stage is the most time-consuming step in SPRING compression, early stopping can reduce compression times by as much as 20% without affecting the compression ratio (see Section 3.10).

Encode reads

In this step, the sequence of reordered reads is used to obtain a majority-based reference sequence. The reference sequence is then used to encode the reordered reads. The final encoding includes the reference sequence, the positions of the reads in the reference sequence, and the mismatches of reads with respect to the reference sequence. An index mapping the reordered reads to their position in the original FASTQ file is also stored. This step is almost unchanged from HARC (Chandak *et al.*, 2018) and more details can be found there. The only major addition in SPRING is the support for variable-length reads of lengths up to 511.

This stage produces a majority-based reference sequence and encoded streams for reads aligned to the reference. A small fraction of reads usually remains unaligned to the reference to the reference and are stored separately. However, the encoded streams do not correspond to the original order of reads in the FASTQ file. Furthermore, the reordering and encoding stages from HARC consider the paired end FASTQ files as a single end FASTQ file obtained by concatenating the two files. Thus, for both the order preserving and order non-preserving modes, we need to transform these streams using the information in the index mapping the reordered reads to their position in the original file. This is done in the next two steps.

Paired end order encoding

This step is used only in the order non-preserving mode. Here we generate a new ordering of the reads which preserves the pairing information while achieving the optimal compression. This step generates an index mapping the reordered reads to their position in the new ordering. The reads in file 1 are kept in the same order as obtained after the previous stage (Encode reads), i.e., the reads in file 1 are sorted according to their position in the majority-based reference. This allows us to store the positions of these reads in the majority-based reference using delta-coding leading to improved compression. The ordering of the reads in file 2 is automatically determined by the ordering of reads in file 1 (since pairing information is preserved).

For single end files (in the order non-preserving mode), the reads are kept in the same order as obtained after the encoding stage (i.e., sorted according to their position in the majority-based reference).

Reorder and compress read streams

In this step, the final encoded streams are generated and compressed in blocks using BSC. For this, first the streams generated by the encoding stage are loaded into the memory. These are then reordered according to the mode. In the order preserving mode, the streams are ordered according to the original order of reads in the FASTQ files. In the order non-preserving mode, the streams are ordered according to the new order generated in the paired end order encoding step. The final streams are described below:

- *seq* - stores the majority-based reference sequence. This is packed into a 2 bits/base representation before compression.
- *flag* - indicates whether the reads are aligned or not as well as the distance between them on the reference.
  - 0 - Both reads aligned and gap between alignment positions is $< 32{,}767$ (for single end datasets, flag 0 means that the read is aligned).
  - 1 - Both reads aligned and gap between alignment positions is $\geq 32{,}767$.
  - 2 - Both reads unaligned (for single end datasets, flag 2 means that the read is unaligned).
  - 3 - read 1 of pair aligned, read 2 unaligned
  - 4 - read 1 of pair unaligned, read 2 aligned
- *pos* - in the order preserving mode, stores the position of the first read of the pair (and possibly the second read) on the reference using 8 bytes. If flag is 0 or 3, only the position of the first read is stored. If flag is 1, positions of both the first and the second reads are stored. If flag is 2, nothing is stored.

  In the order non-preserving mode, the position of the first read of the pair is stored as the difference from the first read of the previous pair (except for the first pair in the block). Note that the difference is always positive because of the way the new order is defined in the paired end order encoding step. This difference is stored as a 2 byte unsigned integer as long as it is $< 65{,}535$. Otherwise we store 65,535 using 2 bytes followed by the actual difference using 8 bytes. Storing differences rather than the absolute position allows SPRING to achieve significantly better compression in the order non-preserving mode.
- *pos_pair* - for paired end datasets, store the gap between the paired reads on the reference using a 16 bit signed integer when the flag is 0. Since the paired reads are sequenced from nearby portions of the genome (paired reads are typically separated by 50-250 bases), they are likely to appear close in the reference. Using a separate stream for the gap between the paired reads allows us to exploit this fact.
- *noise* - store the noisy bases in the aligned reads with respect to the reference. The encoding depends on both the base in the reference and in the read, allowing us to exploit the fact that certain errors are more likely in Illumina sequencing. For example, the most probable transitions for each reference symbol are encoded as 0, next most probable transitions as 1 and so on. This leads to more 0's in the encoded stream leading to better compression. A newline character separates the noise for consecutive reads.

- *noisepos* - stores the position of the noisy bases encoded in the *noise* stream. These are delta encoded to exploit the fact that most sequencing errors occur towards the end of the read. The delta coded noise positions are stored as 16 bit unsigned integers.
- *RC* - store the orientation (forward/reverse) of aligned reads with respect to the reference. If flag is 0, this does not store the orientation of the second read in the pair (see *RC_pair* stream).
- *RC_pair* - for paired end datasets, store the relative orientation of the second read with respect to the first read when the flag is 0. If the paired reads have opposite orientation, store 0, otherwise store 1. Since the paired end reads have opposite orientation of the genome, we expect to get mostly 0's in this stream and hence this stream is highly compressible.
- *unaligned* - stores the unaligned reads without any encoding.
- *length* - store the read lengths as 16 bit unsigned integers.

Reorder and compress quality and ids

This step is used only in the order non-preserving mode. Here the quality and ids are reordered to match the new ordering of the reads. After reordering, they are compressed in blocks (as done in preprocess stage for the order preserving mode). To reduce the memory consumption during reordering, SPRING makes multiple passes over the quality and ids. In each pass, a subset of quality and ids are loaded into memory and these are compressed according to the new ordering. If QVZ is being used for quality value quantization, it is applied before compression. Note that Illumina binning and binary thresholding of quality values are already applied in the preprocessing step, so they are not required in this step. In case the qualities and/or identifiers are not to be preserved, this step simply ignores them.

Tar

All the compressed streams are converted to a tar archive at the end.

Decompression

During decompression, first the *seq* stream is decompressed (not applicable for long read mode). Then, multiple threads decompress the blocks in parallel which are then written to the output files by the master thread. SPRING supports decompression of a subset of reads by specifying the `--decompress-range` flag. In this case, the entire *seq* stream is decompressed and then only the blocks corresponding to the desired range of reads are decompressed.

2 Extended main results

The proposed algorithm, SPRING, was tested on a variety of datasets and compared to various algorithms. For lossless compression, we compare SPRING with pigz (`https://zlib.net/pigz/`), FaStore (Roguski *et al.*, 2018) and DSRC 2 (Roguski and Deorowicz, 2014). pigz (parallelized Gzip) was chosen as it is currently the standard FASTQ compressor. FaStore does not preserve the order of the reads and hence is not lossless in general. However, for these datasets, the original order can be recovered from the read identifiers since they are sequentially ordered. For the recommended lossy mode, we compare SPRING with FaStore. The compression script for FaStore was modified so that the information retained in this mode is the same for SPRING and FaStore (details in Section 5). For both modes, we tested both the default and fast mode of FaStore. While several tools such as SCALCE (Hach *et al.*, 2012) and Fqzcomp (Bonfield and Mahoney, 2013) provide much better compression than pigz, we decided to test only FaStore and DSRC 2 since they represent the state-of-the-art in terms of compression ratio and compression speed, respectively (Roguski *et al.*, 2018). Moreover, the fast mode of FaStore still achieves better compression than the other tools, while achieving similar or faster compression speed.

All the experiments were run on a server with a 40-core Intel(R) Xeon(R) 2.20 GHz processor, 258 GB of RAM, 7.3 TB disk space and Ubuntu 18.04. All tools were run with 8 threads. 1 MB denotes 10^6 bytes

Dataset	Genome length (Mb)	Read length	#reads (M)	Coverage	PE/SE	Technology	Accession no.
E. coli	4.6	301	1.3	85	PE	MiSeq	SRR1770413
P. aeruginosa	6	100	3.3	50	PE	GAIIx	SRR554369
S. cerevisiae	12.1	63, 75	30	175	PE	GAII	SRR327342
T. cacao	350	74	69	15	SE	GAIIx	SRR870667_2
PhiX	0.0054	100	200	3.7×10^6	PE	NovaSeq	PhiX
Metagenomic	-	100	72	-	PE	HiSeq 2000	ERR532393
H. sapiens 1	3137	100	48.9	1.6	PE	GAII	SRR062634
H. sapiens 2	3137	101	879	28	PE	HiSeq 2000	ERP001775
H. sapiens 3	3137	147	540	25	PE	NovaSeq	NA12878 Rep 1, Lane 1
H. sapiens 4	3137	147	2173	100	PE	NovaSeq	NA12878 Rep 1 & 2

Table 1: Short read datasets used for evaluation. PE denotes paired-end, SE denotes single-end. For SRR327342, the read length of the first read in each pair is 63, and that of the second read is 75. Instructions for obtaining these datasets are provided in Section 4.

and 1 GB denotes 10^9 bytes throughout the paper. Instructions for installing and using SPRING and other tools along with the commands used for the experiments are available in Section 5.

The datasets (listed in Table 1) include viral, bacterial, metagenomic, plant and human sequencing data and cover a range of coverages, Illumina sequencing technologies, and read lengths. Some of these datasets are part of a compilation by MPEG HTS compression working group for benchmarking purposes (Numanagić *et al.*, 2016). The human NovaSeq datasets were obtained from Illumina BaseSpace public data and consisted of variable-length ($\approx$ 150bp) paired-end reads along with 4-level quality values. These were trimmed to 147bp as FaStore does not support variable-length reads. Results for SPRING for the original variable-length datasets are in Section 3.5. All the datasets used in this work are publicly available and links to these are provided in Section 4.

Dataset	Uncompressed size	pigz	DSRC 2	FaStore (fast)	FaStore	SPRING	Improvement over FaStore
E. coli	827	253	189	-	-	**106**	-
P. aeruginosa	768	279	198	142	145	**115**	1.26x
S. cerevisiae	5,986	2,062	1,507	-	-	**954**	-
T. cacao	13,847	4,926	3,540	2,755	2,714	**2,444**	1.11x
Metagenomic	19,284	6,911	5,155	3,628	3,602	**3,206**	1.12x
PhiX	50,090	6,402	6,594	1,552	1,457	**1,420**	1.03x
H. sapiens 1	12,861	3,920	2,702	2,293	2,299	**2,118**	1.09x
H. sapiens 2	227,246	74,250	52,049	36,042	35,662	**28,901**	1.23x
H. sapiens 3	195,748	36,131	26,520	11,380	11,101	**6,971**	1.59x
H. sapiens 4	787,616	144,927	106,665	35,129	33,734	**25,883**	1.30x

Table 2: Sizes in MB for lossless compression. FaStore wasn't run on *S. cerevisiae* since it does not support variable length reads. On *E. coli*, FaStore exited with a segmentation fault. Best results are boldfaced.

Tables 2 and 3 show the compression results for the lossless and recommended lossy modes, respectively. We see that SPRING consistently achieves the best compression ratios for both modes across the selected datasets, except for lossy compression of the extremely high coverage (3.7×10^6x) *PhiX* dataset. For the 28x human dataset (*H. sapiens 2*) from the Platinum Genomes Project (ERP001775) (Eberle *et al.*, 2017), SPRING achieves 1.2-1.3x better compression than FaStore. The space required for the recommended lossy mode is less than half of the lossless mode, primarily due to Illumina binning of quality values (see Section 3.1).

The improvement is even more significant for the NovaSeq datasets, with close to 1.75x improvement for

Dataset	Uncompressed size	FaStore (fast)	FaStore	SPRING	Improvement over FaStore
E. coli	827	-	-	**63**	-
P. aeruginosa	768	83	88	**62**	1.41x
S. cerevisiae	5,986	-	-	**366**	-
T. cacao	13,847	1,339	1,300	**1,215**	1.07x
Metagenomic	19,284	1,937	1,935	**1,736**	1.11x
PhiX	50,090	1,226	**1,099**	1,160	0.95x
H. sapiens 1	12,861	1,244	1,251	**1,223**	1.02x
H. sapiens 2	227,246	17,846	17,417	**13,460**	1.29x
H. sapiens 3	195,748	10,246	9,927	**5,657**	1.75x
H. sapiens 4	787,616	30,379	28,846	**20,316**	1.42x

Table 3: Sizes in MB for recommended lossy compression. FaStore wasn't run on *S. cerevisiae* since it does not support variable length reads. On *E. coli*, FaStore exited with a segmentation fault. Best results are boldfaced.

the recommended lossy mode on the 25x-coverage dataset (*H. sapiens 3*). For the 100x NovaSeq dataset (*H. sapiens 4*), SPRING can save around 8 GB ($\approx 25\%$) storage space as compared to FaStore in both modes. The improvement provided by SPRING is not as significant for extremely high (*PhiX*) or low (*H. sapiens* 1) coverages, but these cases are less common in practice. The difference between HiSeq 2000 and NovaSeq datasets, and the contribution of reads, quality values and read identifiers to the compressed sizes are discussed further in Section 3.1. Finally, we observe that FaStore and SPRING achieve close to 5x better compression than pigz and 2-3x better compression than DSRC 2.

Dataset	Lossless					Recommended lossy		
	pigz	DSRC 2	FaStore (fast)	FaStore	SPRING	FaStore (fast)	FaStore	SPRING
E. coli	10s	2s	-	-	41s	-	-	41s
P. aeruginosa	31s	4s	35s	2m2s	23s	28s	1m50s	27s
S. cerevisiae	1m17s	25s	-	-	3m3s	-	-	2m55s
T. cacao	3m	1m10s	5m12s	18m	9m	3m30s	15m	9m
Metagenomic	4m38s	1m27s	7m	17m	10m	5m	14m	10m
PhiX	6m	2m8s	13m	30m	14m	11m	25m	17m
H. sapiens 1	2m37s	36s	4m37s	25m	11m	3m54s	24m	11m
H. sapiens 2	49m	13m	1h19m	3h35m	2h30m	1h	3h9m	2h32m
H. sapiens 3	33m	9m	58m	2h36m	2h	53m	2h28m	2h13m
H. sapiens 4	2h17m	43m	4h10m	9h51m	6h39m	3h50m	8h52m	7h33m

Table 4: Compression times. All tools were run with 8 threads.

Tables 4 and 5 contain the compression times and RAM usage, respectively, for both modes. We observe that pigz and DSRC 2 require significantly lesser computational resources at the cost of worse compression ratios. FaStore (fast) is more than twice as fast as FaStore, while providing similar compression ratios. SPRING is competitive in terms of compression time and memory, requiring less time and memory than FaStore in most cases. SPRING is slower in the recommended lossy mode because of the additional step of reordering qualities and identifiers according to the new order of the reads.

The high memory consumption of SPRING is primarily due to the read reordering step, where SPRING loads all the reads and two hash tables into memory. The previous work on HARC (Chandak *et al.*, 2018) discusses a strategy for reducing the memory consumption by splitting the FASTQ file into multiple parts and applying the compressor independently on each part. Since the memory consumption for SPRING/HARC is linear in the number of reads, this strategy can reduce the memory consumption significantly at the cost

Dataset	Lossless					Recommended lossy		
	pigz	DSRC 2	FaStore (fast)	FaStore	SPRING	FaStore (fast)	FaStore	SPRING
E. coli	0.008	0.13	-	-	1.4	-	-	1.1
P. aeruginosa	0.008	0.13	2.3	2.3	1.5	2.1	2.1	0.84
S. cerevisiae	0.008	0.13	-	-	2.3	-	-	2.3
T. cacao	0.008	0.13	4.2	4.1	3.3	3.4	3.6	3.7
Metagenomic	0.008	0.13	11	11	3.6	9.3	9.2	5.0
PhiX	0.008	0.12	25	26	18	20	24	21
H. sapiens 1	0.008	0.18	17	18	4.9	13	14	5.3
H. sapiens 2	0.008	0.42	35	31	45	25	26	45
H. sapiens 3	0.008	0.13	40	41	32	38	32	31
H. sapiens 4	0.008	0.15	158	137	119	145	122	119

Table 5: Compression memory (RAM) in GB. All tools were run with 8 threads.

of worse read compression.

To achieve better read compression, SPRING also devotes more time to read compression stages (reorder reads and encode reads). Together, these two stages take about 4 hours (out of total 6h39m) for the lossless compression of 100x *H. sapiens 4* dataset. We note here that SPRING introduces early stopping to the reordering stage, which reduces the overall compression time by up to 20% (see Section 3.10). Further improvements in compression time can be achieved by using more threads (see Section 3.7).

Dataset	Lossless					Recommended lossy		
	pigz	DSRC 2	FaStore (fast)	FaStore	SPRING	FaStore (fast)	FaStore	SPRING
E. coli	3s	2s	-	-	17s	-	-	15s
P. aeruginosa	4s	2s	12s	18s	9s	7s	12s	7s
S. cerevisiae	27s	10s	-	-	1m	-	-	43s
T. cacao	1m13s	23s	2m5s	2m14s	2m20s	1m9s	1m11s	1m46s
Metagenomic	1m46s	37s	2m42s	3m	3m18s	1m21s	1m36s	2m29s
PhiX	2m23s	39s	3m3s	3m47s	5m32s	2m33s	2m11s	5m34s
H. sapiens 1	1m	18s	1m27s	1m39s	2m25s	58s	59s	2m
H. sapiens 2	20m	14m	24m	25m	38m	15m	16m	28m
H. sapiens 3	11m	9m	11m	12m	26m	9m	10m	22m
H. sapiens 4	1h21m	41m	40m	45m	1h47m	32m	36m	1h37m

Table 6: Decompression times. All tools were run with 8 threads.

Tables 6 and 7 contain the decompression times and RAM usage, respectively, for both modes. SPRING achieves reasonably fast decompression, while using much less memory as compared to FaStore. By using more threads, SPRING can achieve faster decompression at the cost of higher memory usage (see Section 3.7). SPRING also supports the ability to decompress a subset of reads without needing to decompress the whole file (see Section 3.4).

3 Additional results

Unless otherwise specified, all tools were run with 8 threads.

Dataset	Lossless					Recommended lossy		
	pigz	DSRC 2	FaStore (fast)	FaStore	SPRING	FaStore (fast)	FaStore	SPRING
E. coli	0.003	0.23	-	-	1.7	-	-	1.7
P. aeruginosa	0.003	0.24	0.78	0.8	1.7	0.53	0.61	1.7
S. cerevisiae	0.003	0.43	-	-	2.2	-	-	1.9
T. cacao	0.003	0.29	1.7	2.3	2.1	1.2	1.5	1.7
Metagenomic	0.003	0.29	1.9	1.9	2.6	1.3	1.4	3.1
PhiX	0.003	0.33	19	16	2.3	15	13	2.3
H. sapiens 1	0.003	0.30	2	1.7	3.2	1.4	1.3	3.7
H. sapiens 2	0.003	0.42	26	19	5.5	21	15	5.5
H. sapiens 3	0.003	0.34	39	23	6.1	30	17	6.3
H. sapiens 4	0.003	0.36	141	85	6.6	110	81	6.7

Table 7: Decompression memory (RAM) in GB. All tools were run with 8 threads.

3.1 Field-wise compression results

Tables 8 and 9 provide the field-wise compression results for lossless and recommended lossy mode, respectively. Since pigz does not provide field-wise compression results, it is not included in these tables. We also exclude FaStore (fast) since it is very similar to FaStore in terms of compression results (the two modes differ only in read compression). Note that the sizes for SPRING are before the Tar step which adds a small overhead. Recall that *H. sapiens 2* is 28x dataset sequenced on HiSeq 2000, while *H. sapiens 3* and *H. sapiens 4* are sequenced on NovaSeq, with coverages 25x and 100x, respectively.

Dataset	Tool	Reads	Quality	Identifier
H. Sapiens 2	DSRC 2	22,188	27,810	2,051
	FaStore	6,968	24,868	3,826
	SPRING	4,253	23,774	858
H. Sapiens 3	DSRC 2	19,845	4,576	2,098
	FaStore	6,152	3,789	1,160
	SPRING	3,040	3,630	292
H. Sapiens 4	DSRC 2	79,850	18,346	8,468
	FaStore	13,741	15,178	4,815
	SPRING	10,125	14,553	1,165

Table 8: Sizes (in MB) of individual fields for lossless compression.

Dataset	Tool	Reads	Quality	Identifier
H. Sapiens 2	FaStore	6,917	10,500	0
	SPRING	2,553	10,892	0
H. Sapiens 3	FaStore	6,138	3,789	0
	SPRING	2,022	3,625	0
H. Sapiens 4	FaStore	13,668	15,178	0
	SPRING	5,722	14,558	0

Table 9: Sizes (in MB) of individual fields for recommended lossy compression.

From Table 8, we see that FaStore and SPRING provide significant improvement in read compression over DSRC 2, while the improvement in quality compression is smaller. Since FaStore reorders the reads even in its lossless mode, the read order information is effectively preserved in the identifiers. Due to this,

the identifiers take much larger size for FaStore as compared to SPRING. Note that SPRING takes less space for read compression than FaStore even though it stores information about the read order in the read field. SPRING achieves slightly better quality compression than FaStore due to the use of BSC instead of QVZ. Comparing the HiSeq 2000 dataset (*H. sapiens 2*) to the other two NovaSeq datasets, we observe that the quality takes up a much smaller fraction of the total size for NovaSeq datasets which have only 4 quality levels. For the NovaSeq data, the size required for reads is quite comparable to that required for qualities. Due to this, SPRING provides greater improvement for these datasets.

In Table 9, both FaStore and SPRING reorder the reads and only retain the pairing information. We see that SPRING requires 2.5-3x less space for compressing the reads in this mode. Comparing with the lossless mode (Table 8), we see that Illumina binning reduces the space needed to store the quality values by more than 2x for *H. sapiens 2*. For SPRING, storing only the pairing information rather than the complete order of reads boosts the read compression by a factor of 1.5-1.7, with more improvement for higher coverage datasets.

3.2 Comparison with alignment + SAM compression

In applications such as sequencing of a new organism or metagenomics, a reference is typically not available and hence reference-free compression of FASTQ files is important. Even if a reference is available, the FASTQ file is typically retained (at least temporarily) and again FASTQ compression becomes necessary. Since the reference and the FASTQ file are usually obtained from different individuals, using a reference-free compressor like SPRING can be beneficial since it is more robust to variations between individuals. To understand this better, we compared SPRING to reference-based alignment. We first used BWA-MEM (Li, 2013) to align the *H. sapiens 3* dataset to the hg19 reference. Then we removed some irrelevant fields from the SAM file (MAPQ, RNEXT, PNEXT, TLEN and optional fields), since these are not used to compress data in the FASTQ file. Finally, we used CRAM v3 (from SAMtools) to compress the SAM file, both before and after sorting according to the genome position. The commands used for these operations are listed in Section 5.3. The parameters were chosen to achieve best compression.

The compressed sizes for the unsorted and sorted SAM files are 7,644 MB and 7,793 MB, respectively. The compressed size for the sorted SAM file with the reference embedded in the CRAM file is 8,488 MB. In comparison, SPRING achieves 6,971 MB without using any external reference. While the CRAM compression step is quite fast (around 25m), the alignment took around 8 hours. SPRING needs 2 hours for compressing this dataset. Thus, we see that directly compressing FASTQ files can be advantageous even when a reference is available. For species with larger variation between individuals, SPRING can provide even greater improvements over alignment + SAM compression.

3.3 Long read compression

Accession no.	Species	Genome length (Mb)	Maximum read length	#reads (M)	Coverage	Technology
SRR1284073	*E. coli*	4.6	49424	0.65	140	PacBio
ERR637420	*E. coli*	4.6	47422	0.08	86	Oxford Nanopore MinION

Table 10: Long read datasets used for evaluation. Both datasets are single end.

Accession no.	Uncompressed	pigz	SPRING
SRR1284073	1,304	546	420
ERR637420	264	120	94

Table 11: Sizes in MB for long read compression.

We compared the long read compression (lossless) mode with pigz, since DSRC 2 and FaStore do not support long reads. We evaluated the tools on two datasets (Table 10) from the most popular long read platforms. The compression results are shown in Table 11. We observe that SPRING achieves better compression than pigz on these datasets, but the improvement is not as pronounced as that for short read datasets. This is because SPRING uses the general-purpose compressor BSC for long read compression rather than the specialized compression method employed for short reads, and hence it is unable to exploit much of the redundancy in the reads. Building a specialized read compressor for long reads is part of future work.

3.4 Decompressing subset of reads

SPRING allows decompression of a subset of reads by specifying a range of reads to decompress. For paired end files, the parameters refer to read pairs rather than reads. Table 12 shows the times needed to decompress 1M, 10M and 100M read pairs from losslessly compressed *H. sapiens 3*, along with the time needed to decompress all read pairs ($\approx$ 270M). The results were obtained by using the `--decompress-range` flag. We see that SPRING allows fast decompression of small subsets of reads, with a slight constant overhead due to decompression of the *seq* stream.

Start pair	End pair	Number of pairs	Decompression time
100M	101M	1M	1m13s
100M	110M	10M	1m48s
100M	200M	100M	9m4s
-	-	270M	22m

Table 12: Time required to decompress subset of read pairs for *H. sapiens 3*. Last row represents decompression of entire file.

3.5 Results for variable length short reads

Table 13 contains compression results for NovaSeq variable length reads. FaStore does not support variable length reads, so only SPRING, pigz and DSRC 2 were tested. On these datasets, SPRING provides 3-5x better compression than pigz and DSRC2.

Sample	NA12878 Rep 1, Lane 1 (original)	NA12878 Rep 1 & 2 (original)
Organism	*H. sapiens*	*H. sapiens*
Technology	NovaSeq	NovaSeq
Coverage	26x	105x
Maximum Read length	151	151
Uncompressed Size	205,386	826,117
Lossless		
pigz	38,007	152,243
DSRC 2	28,448	114,393
SPRING	7,565	29,020
Recommended lossy		
SPRING	6,193	22,954

Table 13: Compression sizes in MB for the variable-length NovaSeq datasets. Only tools supporting variable length reads were tested.

3.6 Quality value lossy compression modes

Mode	Parameters	Compressed quality size
Lossless	-	23,774
Illumina binning	-	10,892
QVZ	Rate = 1 bit/quality value	7,237
Binary thresholding	thr=20, high=40, low=6	1,034

Table 14: Sizes in MB for different quality value compression modes for *H. sapiens 2*.

While the recommended lossy mode for SPRING uses Illumina 8-level binning for quality values, SPRING supports two other schemes for lossy compression of quality values. Table 14 shows the compressed sizes for these schemes for *H. sapiens 2* (HiSeq 2000, 28x). In previous works (Roguski *et al.* (2018) and Ochoa *et al.* (2017)), all these were shown to have no detrimental effect on variant calling (binary thresholding can hurt variant calling for low coverage datasets). We see that binary thresholding can reduce the space needed by qualities significantly. Since QVZ uses an optimized context-dependent quantizer dependent on the input data, it is slightly slower, but provides more flexibility in terms of the desired rate and should provide lower distortion at the same rate than other schemes.

3.7 Impact of number of threads

Number of threads	Compressed size (MB)	Compression time	Compression memory (GB)	Decompression time	Decompression memory (GB)
4	6,977	3h26m	31	42m	4.6
8	6,971	2h	32	26m	6.1
16	6,960	1h20m	32	18m	9.3
32	6,968	1h6m	32	13m	15

Table 15: Impact of number of threads on compressed size and time/memory consumption for lossless compression of *H. sapiens 3*.

Table 15 shows the compressed sizes and computational requirements for lossless compression *H. sapiens 3* dataset for three values of number of threads. The results were obtained by using the `-t` flag. We observe that the compression and decompression times improve as the number of threads increase. For very high number of threads, the disk I/O becomes the bottleneck leading to diminishing returns. The impact of increasing the number of threads on the compressed size and compression memory usage is negligible. The decompression memory increases with the number of threads because more blocks are now decompressed in parallel.

3.8 Impact of block size

Block size	Compressed size (MB)	Compression time	Compression memory (GB)	Decompression time	Decompression memory (GB)
128,000	7,011	1h59m	31	26m	5.5
256,000	6,971	2h	32	26m	6.1
512,000	6,950	1h56m	32	24m	8.9

Table 16: Impact of block size on compressed size and time/memory consumption for lossless compression of *H. sapiens 3*.

SPRING compresses the streams in blocks to allow random access and efficient decompression. The number of reads (read pairs for PE datasets) per block is set to 256,000 by default (for short reads). Table 16 shows the compressed sizes and computational requirements for lossless compression *H. sapiens 3* dataset for three values of block size. The results were obtained by modifying the parameter `NUM_READS_PER_BLOCK` in `src/params.h`. We observe that using higher block sizes yields slightly better compression but needs more memory during decompression. The impact on compression and decompression times is negligible for this range of block sizes.

3.9 Impact of read reordering on ID compression

While read identifiers are not used in most downstream applications, some applications like Picard MarkDuplicates (`http://broadinstitute.github.io/picard/`) might use it. In such cases, we recommend that SPRING be used without the `-r` flag which allows read reordering (with pairing information preserved). When the `-r` flag is specified, the read identifiers are reordered and then compressed. Due to this, the consecutive read identifiers now contain large differences, leading to poor compression. Even though the size needed for the reads reduces, the increase in the read identifier size slightly outweighs this reduction. Table 17 shows this for *H. sapiens 3* dataset.

Flag	Read	Read identifier	Read + Read identifier
`-r` used	2,020	1,371	3,391
`-r` not used	3,040	292	3,332

Table 17: Impact of using `-r` flag on read and read identifier compression for *H. sapiens 3*. Sizes are in MB.

3.10 Improvements in reordering stage

Here we discuss the impact of two improvements made in SPRING to the reordering stage of HARC.

Searching for matches in both directions

Mode	Compressed size of reads (MB)	Compression time	Compression memory (GB)
Search in one direction	3,251	1h59m	32
Search in both directions	3,040	2h	32

Table 18: Impact of bidirectional search on compressed size and time/memory consumption for lossless compression of *H. sapiens 3*.

While HARC searched for matching reads in only one direction (matching the suffix of the current read), SPRING looks for matches in both directions. Table 18 shows the impact of this on read compression for lossless compression of *H. sapiens 3* dataset (note that the reported times include time for quality and identifier compression). The results for the first row were obtained by replacing `src/reorder.h` in the SPRING repository by `src/old_src/reorder_1d/reorder.h`. We observe that bidirectional search improves the read compression by around 6% without significantly affecting the compression time/memory.

Early stopping

SPRING maintains the fraction of unmatched reads in the last 1 million reads and stops looking for matches once this fraction crosses a certain threshold (50% by default). The maximum impact of this step is on the largest dataset *H. sapiens 4*. Table 19 shows results for lossless compression of *H. sapiens 4* where the reported times includes the time for quality and identifier compression. The results for the first row were

Mode	Compressed size of reads (MB)	Compression time	Compression memory (GB)
No early stopping	10,107	9h15m	119
Early stopping	10,125	6h39m	119

Table 19: Impact of early stopping on compressed size and time/memory consumption for lossless compression of *H. sapiens 4*.

obtained by setting `STOP_CRITERIA_REORDER` to 1.0 in `src/params.h`. We see that early stopping reduces the compression time by more than 20% for this dataset while having negligible impact on compressed size and memory usage.

4 Datasets

We list below the links for the datasets used for evaluation. After downloading, the files were unzipped using gunzip command. In some cases FASTQ files were concatenated to get higher coverage datasets.

***E. coli* - SRR1770413**

This was downloaded using the SRA toolkit (`https://www.ncbi.nlm.nih.gov/sra/docs/toolkitsoft/`) by running the command
`./fastq_dump --split-files SRR1770413`

***P. aeruginosa* - SRR554369**

`ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR554/SRR554369/SRR554369_1.fastq.gz`
`ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR554/SRR554369/SRR554369_2.fastq.gz`

***S. cerevisiae* - SRR327342_1**

`ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR327/SRR327342/SRR327342_1.fastq.gz`
`ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR327/SRR327342/SRR327342_2.fastq.gz`

Metagenomic - ERR532393

`ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR532/ERR532393/ERR532393_1.fastq.gz`
`ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR532/ERR532393/ERR532393_2.fastq.gz`

***T. cacao* - SRR870667_2**

`ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR870/SRR870667/SRR870667_2.fastq.gz`

***H. sapiens* - ERP001775**

`ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174324/ERR174324_1.fastq.gz`
`ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174325/ERR174325_1.fastq.gz`
`ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174324/ERR174324_2.fastq.gz`
`ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174325/ERR174325_2.fastq.gz`
The first two files were concatenated and the last two files were concatenated to obtain a 28x coverage paired-end dataset.

***H. sapiens* - NA12878 Rep 1, Lane 1**

This dataset was downloaded from Illumina's BaseSpace public data (`https://basespace.illumina.com/datacentral`) from the project *NovaSeq S2: Nextera DNA Flex (8 replicates of NA12878)*. The following FASTQ files comprise this 25x paired-end dataset.

```
NA12878-Rep-1_S1_L001_R1_001.fastq
NA12878-Rep-1_S1_L001_R2_001.fastq
```

These datasets were variable length and were trimmed to 147bp for evaluation with FaStore. Trimming was done using util/trimmer.cpp, available in the SPRING source code.

***H. sapiens* - NA12878 Rep 1 & 2**

This dataset was downloaded from Illumina's BaseSpace public data (`https://basespace.illumina.com/datacentral`) from the project *NovaSeq S2: Nextera DNA Flex (8 replicates of NA12878)*. The following FASTQ files were downloaded.

```
NA12878-Rep-1_S1_L001_R1_001.fastq
NA12878-Rep-1_S1_L002_R1_001.fastq
NA12878-Rep-2_S2_L001_R1_001.fastq
NA12878-Rep-2_S2_L002_R1_001.fastq
NA12878-Rep-1_S1_L001_R2_001.fastq
NA12878-Rep-1_S1_L002_R2_001.fastq
NA12878-Rep-2_S2_L001_R2_001.fastq
NA12878-Rep-2_S2_L002_R2_001.fastq
```

The first four files were concatenated together and the last four files were concatenated together to obtain 100x paired-end data. These datasets were variable length and were trimmed to 147bp for evaluation with FaStore. Trimming was done using util/trimmer.cpp, available in the SPRING source.

PhiX

This dataset was obtained directly from a sequencing facility. The losslessly compressed SPRING archive can be downloaded from
`https://web.stanford.edu/~schandak/PhiX_100M_lossless.spring`

***E. coli* - SRR1284073**

This was downloaded using the SRA toolkit (`https://www.ncbi.nlm.nih.gov/sra/docs/toolkitsoft/`) by running the command
`./fastq_dump SRR1284073`

***E. coli* - ERR637420**

This was downloaded using the SRA toolkit (`https://www.ncbi.nlm.nih.gov/sra/docs/toolkitsoft/`) by running the command
`./fastq_dump ERR637420`

5 Installing and running tools

5.1 Installation

This section contains instructions for downloading and installing the tools that were tested in this work. Instructions for installing them in different environments can be found in the README documents for the tools.

SPRING

Commit:
https://github.com/shubhamchandak94/Spring/commit/490494c7d5bd7c55751e305d9ec6caedc3b66af7

```
git clone https://github.com/shubhamchandak94/SPRING.git
cd SPRING
mkdir build
cd build
cmake ..
make
```

FaStore

Commit:
https://github.com/refresh-bio/FaStore/commit/e16dfa91577a21144ca97ade7c5772983044187d

```
git clone https://github.com/refresh-bio/FaStore.git
cd FaStore
make
```

DSRC 2

Boost should already be installed.
Commit:
https://github.com/refresh-bio/DSRC/commit/5eda82cb1546b71cd3480bc0aba1d321b52bd0b4

```
git clone https://github.com/refresh-bio/DSRC.git
cd DSRC
make
```

pigz

```
wget https://zlib.net/pigz/pigz-2.4.tar.gz
tar -xzvf pigz-2.4.tar.gz
cd pigz-2.4
make
```

5.2 Running compression algorithms

5.2.1 SPRING

General usage:

```
Allowed options:
  -h [ --help ]                   produce help message
  -c [ --compress ]               compress
  -d [ --decompress ]             decompress
  --decompress-range arg          --decompress-range start end
                                  (optional) decompress only reads (or read
                                  pairs for PE datasets) from start to end
                                  (both inclusive) (1 <= start <= end <=
                                  num_reads (or num_read_pairs for PE)). If -r
                                  was specified during compression, the range
                                  of reads does not correspond to the original
                                  order of reads in the FASTQ file.
  -i [ --input-file ] arg         input file name (two files for paired end)
```

```
-o [ --output-file ] arg        output file name (for paired end
                                decompression, if only one file is specified,
                                two output files will be created by suffixing
                                .1 and .2.)
-w [ --working-dir ] arg (=.)   directory to create temporary files (default
                                current directory)
-t [ --num-threads ] arg (=8)   number of threads (default 8)
-r [ --allow-read-reordering ]  do not retain read order during compression
                                (paired reads still remain paired)
--no-quality                    do not retain quality values during
                                compression
--no-ids                        do not retain read identifiers during
                                compression
-q [ --quality-opts ] arg       quality mode: possible modes are
                                1. -q lossless (default)
                                2. -q qvz qv_ratio (QVZ lossy compression,
                                parameter qv_ratio roughly corresponds to
                                bits used per quality value)
                                3. -q ill_bin (Illumina 8-level binning)
                                4. -q binary thr high low (binary (2-level)
                                thresholding, quality binned to high if >=
                                thr and to low if < thr)
-l [ --long ]                   Use for compression of arbitrarily long read
                                lengths. Can also provide better compression
                                for reads with significant number of indels.
                                -r disabled in this mode. For Illumina short
                                reads, compression is better without -l flag.
```

For compression in the lossless mode (lossless quality compression, read identifiers retained and read order preserved), run

```
./spring -c -i in_1.fastq in_2.fastq -o compressed_file
```

For compression in the recommended lossy mode (Illumina binning of quality, read identifiers not retained and read order not preserved), run

```
./spring -c -i in_1.fastq in_2.fastq -r --no-ids -q ill_bin -o compressed_file
```

For the NovaSeq datasets, Illumina binning is not performed, hence run

```
./spring -c -i in_1.fastq in_2.fastq -r --no-ids -o compressed_file
```

For long read compression (lossless), run

```
./spring -c -i in.fastq -l -o compressed_file
```

For decompression (in either mode), run,

```
./SPRING/spring -d compressed_file -o out.fastq
```

More examples for usage of SPRING with various options are available in the Github README (`https://github.com/shubhamchandak94/SPRING/blob/master/README.md`).

5.2.2 Other algorithms

FaStore

Since the lossy mode in FaStore does not match with the recommended lossy mode in this work, we modified the compression script provided with FaStore. The modified script `fastore_compress.sh` is available in the `util` directory of SPRING repository.

To compress in_1.fastq and in_2.fastq in the lossless mode (lossless quality compression and read identifiers retained), run

```
./FaStore/scripts/fastore_compress.sh --lossless --threads 8 --in in_1.fastq\
--pair in_2.fastq --out compressed.fastore --verbose
```

The compressed files are compressed.fastore.cdata and compressed.fastore.cmeta.

In the recommended lossy mode (Illumina binning for quality and read identifiers not retained), run

```
./FaStore/scripts/fastore_compress.sh --lossy_old --threads 8 --in in_1.fastq\
--pair in_2.fastq --out compressed.fastore --verbose
```

For NovaSeq datasets, Illumina binning is not applied, so run

```
./FaStore/scripts/fastore_compress.sh --lossy_novaseq --threads 8 --in in_1.fastq\
--pair in_2.fastq --out compressed.fastore --verbose
```

In the fast mode, just add the `--fast` flag.

To decompress compressed.fastore to out_1.fastq and out_2.fastq, run

```
./FaStore/bin/fastore_pack d -z -t8 -icompressed.fastore -o"out_1.fastq out_2.fastq"
```

For single-end datasets, the -z flag should be removed.

In the lossless mode, FaStore retains the order of the reads through the read identifiers. However, on decompression, the reads are not outputted in their original order. While it is possible to sort the FASTQ file using the identifiers to get back the original order, we did not include this step when measuring the decompression time/memory.

pigz

Compression:

```
./pigz-2.4/pigz -k -p 8 in_1.fastq in_2.fastq
```

Decompression:

```
./pigz-2.4/unpigz -k -p 8 in_1.fastq.gz in_2.fastq.gz
```

DSRC 2

Compression:

```
./DSRC/bin/dsrc -c -t8 -v in_1.fastq in_1.fastq.dsrc
./DSRC/bin/dsrc -c -t8 -v in_2.fastq in_2.fastq.dsrc
```

Decompression:

```
./DSRC/bin/dsrc -d -t8 -v in_1.fastq.dsrc out_1.fastq
./DSRC/bin/dsrc -d -t8 -v in_2.fastq.dsrc out_2.fastq
```

5.3 Alignment and SAM compression

Downloading and installing BWA-MEM and SAMtools (you may need to install libbzip2-dev and liblzma-dev packages on Linux before this):

```
wget https://github.com/samtools/samtools/releases/download/1.9/samtools-1.9.tar.bz2
tar -xjf samtools-1.9.tar.bz2
cd samtools-1.9/
./configure
make
```

```
cd ..
wget https://github.com/lh3/bwa/releases/download/v0.7.17/bwa-0.7.17.tar.bz2
tar -xjf bwa-0.7.17.tar.bz2
cd bwa-0.7.17/
make
cd ..
```

Downloading and indexing human genome:

```
wget ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/human_g1k_v37.fasta.gz
gunzip human_g1k_v37.fasta.gz
./bwa-0.7.17/bwa index human_g1k_v37.fasta
./samtools-1.9/samtools faidx human_g1k_v37.fasta
```

Aligning reads in file_1.fastq and file_2.fastq to the reference:

```
./bwa-0.7.17/bwa mem -t 8 human_g1k_v37.fasta file_1.fastq file_2.fastq > file.sam
```

Removing optional fields in the SAM file and replacing MAPQ, RNEXT, PNEXT and TLEN by their "information unavailable" values (see SAM format specifications: `https://samtools.github.io/hts-specs/SAMv1.pdf`):

```
cut -f1-11 < file.sam |\
awk -v OFS='\t' '{if($0 ~ "^@") {print $0} \
else {$5 = 255; $7 = "*"; $8 = 0; $9 = 0; print}}'\
> file_reduced.sam
```

Sort SAM file using SAMtools (we allow maximum 100 GB RAM usage to speed up this step):

```
./samtools-1.9/samtools sort -m 100G -O SAM -o file_reduced_s.sam file_reduced.sam
```

Compressing with CRAM (unsorted SAM file):

```
./samtools-1.9/samtools view -C -o file_reduced.cram -T human_g1k_v37.fasta \
--output-fmt-option nthreads=8 --output-fmt-option level=9 \
--output-fmt-option use_lzma --output-fmt-option use_bzip2 file_reduced.sam
```

Compressing with CRAM (sorted SAM file, without embedding reference in the CRAM file):

```
./samtools-1.9/samtools view -C -o file_reduced_s.cram -T human_g1k_v37.fasta \
--output-fmt-option nthreads=8 --output-fmt-option level=9 \
--output-fmt-option use_lzma --output-fmt-option use_bzip2 file_reduced_s.sam
```

Compressing with CRAM (sorted SAM file, embedding reference in the CRAM file):

```
./samtools-1.9/samtools view -C -o file_reduced_s_embed.cram -T human_g1k_v37.fasta \
--output-fmt-option nthreads=8 --output-fmt-option level=9 \
--output-fmt-option embed_ref  --output-fmt-option use_lzma --output-fmt-option use_bzip2 \
file_reduced_s.sam
```

References

Bonfield, J. K. and Mahoney, M. V. (2013). Compression of FASTQ and SAM format sequencing data. *PloS one*, **8**(3), e59190.

Chandak, S., Tatwawadi, K., and Weissman, T. (2018). Compression of genomic sequencing reads via hash-based reordering: algorithm and analysis. *Bioinformatics*, **34**(4), 558–567.

Cock, P. J. A., Fields, C. J., Goto, N., Heuer, M. L., and Rice, P. M. (2010). The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research*, **38**(6), 1767–1771.

Eberle, M. A., Fritzilas, E., Krusche, P., Källberg, M., Moore, B. L., Bekritsky, M. A., Iqbal, Z., Chuang, H.-Y., Humphray, S. J., Halpern, A. L., Kruglyak, S., Margulies, E. H., McVean, G., and Bentley, D. R. (2017). A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree. *Genome Research*, **27**(1), 157–164.

Hach, F., Numanagić, I., Alkan, C., and Sahinalp, S. C. (2012). SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics*, **28**(23), 3051–3057.

Illumina (a). Illumina technical note, "Quality Scores for Next-Generation Sequencing", `https://www.illumina.com/documents/products/technotes/technote_Q-Scores.pdf`.

Illumina (b). Illumina technical note, "Reducing Whole-Genome Data Storage Footprint", `https://www.illumina.com/documents/products/whitepapers/whitepaper_datacompression.pdf`.

Li, H. (2013). Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. *arXiv preprint arXiv:1303.3997*.

Long, R., Hernaez, M., Ochoa, I., and Weissman, T. (2017). Genecomp, a new reference-based compressor for sam files. In *2017 Data Compression Conference (DCC)*, pages 330–339.

Malysa, G., Hernaez, M., Ochoa, I., Rao, M., Ganesan, K., and Weissman, T. (2015). QVZ: lossy compression of quality values. *Bioinformatics*, **31**(19), 3122–3129.

Numanagić, I., Bonfield, J. K., Hach, F., Voges, J., Ostermann, J., Alberti, C., Mattavelli, M., and Sahinalp, S. C. (2016). Comparison of high-throughput sequencing data compression tools. *Nature Methods*, **13**(12), 1005.

Ochoa, I., Hernaez, M., Goldfeder, R., Weissman, T., and Ashley, E. (2017). Effect of lossy compression of quality scores on variant calling. *Briefings in Bioinformatics*, **18**(2), 183–194.

Roguski, L. and Deorowicz, S. (2014). DSRC 2-Industry-oriented compression of FASTQ files. *Bioinformatics*, **30**(15), 2213–2215.

Roguski, L., Ochoa, I., Hernaez, M., and Deorowicz, S. (2018). Fastore: a space-saving solution for raw sequencing data. *Bioinformatics*, **34**(16), 2748–2756.