# Compression of genomic sequencing reads via hash-based reordering: Supplementary Data

Shubham Chandak, Kedar Tatwawadi and Tsachy Weissman

June 12, 2017

In this document, some instructions for downloading datasets, installing and running the compression tools and some additional results are provided. The proposed algorithm HARC and related instructions can be found `https://github.com/shubhamchandak94/readcompression`.

## Datasets

For all the below datasets the FASTQ files from the provided links were downloaded, decompressed and concatenated into a single FASTQ file.

### *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
```

### *T. cacao* - SRR870667_2

```
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR870/SRR870667/SRR8700667_2.fastq.gz
```

### *H. sapiens* 1 - ERR174310

```
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174310/ERR174310_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174310/ERR174310_2.fastq.gz
```

### *H. sapiens* 2 - ERR194146

```
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR194/ERR194146/ERR194146.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR194/ERR194146/ERR194146_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR194/ERR194146/ERR194146_2.fastq.gz
```

This dataset is part of the SAM datasets in the MPEG benchmarking dataset. Thus, the reads were shuffled to obtain random ordering in the FASTQ file.

***H. sapiens* 2 - ERP001775_1**

```
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/ERR174326/ERR174326_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174327/ERR174327_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174328/ERR174328_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174329/ERR174329_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174330/ERR174330_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174331/ERR174331_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174332/ERR174332_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174333/ERR174333_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174334/ERR174334_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174335/ERR174335_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174336/ERR174336_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174337/ERR174337_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174338/ERR174338_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174339/ERR174339_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174340/ERR174340_1.fastq.gz
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174341/ERR174341_1.fastq.gz
```

## Generating Simulated reads

To generate the simulated reads used in the paper, first download the human genome from

```
ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/human_g1k_v37.fasta.gz
```

Then separate the particular chromosome (e.g. Chromosome 22 to chrom22.fasta).
Change folder to
`readcompression/util/gen_fastq_noRC`
Then run `make`. To generate 35000000 noiseless reads of length 100 from chrom22.fasta without any reverse complementation, run
`./gen_fastq_noRC 35000000 100 PATH/chrom22.fasta PATH/chrom22_reads.fastq`
For reads with 1% uniform noise, add a `-e` at the end.

## Downloading and installing tools

### Proposed Algorithm

```
git clone https://github.com/shubhamchandak94/readcompression.git
cd readcompression
./install.sh
```

### Orcom

Boost should already be installed.

```
git clone https://github.com/lrog/orcom.git
cd orcom
make boost
```

### Leon

```
git clone --recursive https://github.com/GATB/leon.git
cd leon
sh install
```

## Running compression algorithms

### Proposed algorithm

Change directory to readcompression/

```
Compression - compresses FASTQ reads. Output written to .tar file
./run_default.sh -c PATH_TO_FASTQ [-p] [-t NUM_THREADS]
-p = Preserve order of reads (compression ratio 2-4x worse if order preserved)
-t = NUM_THREADS - default 8

Decompression - decompresses reads. Output written to .dna.d file
./run_default.sh -d PATH_TO_TAR [-p] [-t NUM_THREADS]
-p = Get reads in original order (slower). Only applicable if -p was used during
compression.
-t = NUM_THREADS - default 8

Help (this message)
./run_default.sh -h
```

### Orcom

Change directory to orcom/bin/
Compression:

```
./orcom_bin e -iPATH/file.fastq -oPATH/file.bin
./orcom_pack e -iPATH/file.bin -oPATH/file.orcom
```

Decompression:

```
./orcom_pack d -iPATH/file.orcom -oPATH/file_orcom.dna
```

### Leon

Change directory to leon. Compression (sequence only mode):

```
./leon -file PATH/file.fastq -seq-only -c
```

Decompression:

```
./leon -file PATH/file.leon -d
```

## Additional results

We present here some additional results and discussions:

### Ordering according to genome position

If there was a way to reorder reads according to their position in the genome, then we could achieve good compression by exploiting the redundancy between neighboring reads. We can represent a read in terms of its previous read, by storing the length of the match with the previous read and the new symbols in the read separately. Note that the sequence formed by the new symbols exactly corresponds to the genome, and can be stored using $L_{FASTA}$ bits. Storing the match lengths can be done in $H(Z|X)$ space. The offset of the current read from the previous read is distributed as a Geometric random variable with mean $m/n$, whose entropy is close to the upper bound on the $H(Z|X)$ term. Recall that $H(Z|X) = \log_2 \binom{n+m-1}{m} \approx (m+n)\log_2(m+n) - m\log_2 m - n\log_2 n$ by Stirling's approximation. The entropy of geometric random variable with mean $\mu$ is $(\mu+1)\log_2(\mu+1) - \mu\log_2\mu$. Substituting $\mu = m/n$ and multiplying by the number of reads $m$, we recover the expression for $H(Z|X)$. We can use an entropy achieving prefix code for geometric random variables to achieve this entropy. Thus, we observe that if the reads are reordered according to their position in the genome sequence, then the upper bound on the entropy as expressed in equation 2 in the paper can be achieved.

### Handling variable-length reads

The current implementation of the algorithms does not support variable-length reads. To handle these reads, certain changes need to be made in the algorithm. In the Hamming distance computation step, the distance should be computed after truncating the longer read so that the read lengths match. The dictionary indices can be chosen in the middle of the read for all the read lengths - thus the exact indices will differ from read to read. Also, the read lengths need to be stored separately in the encoding stage as well.

Another way to handle such datasets is to run the algorithm separately on the portions having the different read lengths. However this is not optimal because the effective coverage for each of the files is lower than the actual coverage.

## Analysis of minimizer-based techniques

Existing reordering compressors like Orcom and Mince cluster the reads into bins based on some shared substring (e.g., the minimizer) and then the bins are encoded and compressed independently. Recall that the minimizer of length $l$ of a read is the lexicographically smallest $l$-length substring in the read. We assume that the reads within each bucket are reordered in an optimal fashion. For example, Orcom orders the reads in a minimizer bin by lexicographically sorting according to portion of read after the minimizer concatenated with portion of read before the minimizer. This heuristic allows the reads overlapping in the genome to be together.

Let $len$ be the length of a read and $n$ be length of genome. To analyze such schemes we assume that as we slide along the genome, the minimizer (of length $l \approx 10$) of $len$-length substrings changes after every $len/2$ positions (based on results in “Reducing storage requirements for biological sequence comparison”). This is reasonable because the reads starting at 1 and $len+1$ are non-overlapping and hence are likely to have different minimizers. Thus the minimizer has to change at some point between 1 and $len + 1$. This effectively means that the reads starting at positions between 1 and $len/2$ go to one bucket and those starting at positions between $len/2 + 1$ and $len$ go to another bucket. Note that reads from different portions of the genome can have the same minimizer. However for simplicity and noting that $\Sigma^{len} \gg n$, we assume that the overlap between such reads is insignificant.

Now we compute the overhead caused by the binning process. Consider the portion of the genome from $len + 1$ to $3 \times len/2$. This portion is present in a read starting at position (i) $len/2 - 1$, (ii) $len/2 + 1$ and (iii) $len + 1$. As per our assumption above, these three reads have different minimizers and hence go to different bins. Thus the portion of the genome from $len + 1$ to $3 \times len/2$ is encoded three times. Thus instead of spending $n \log_2(\Sigma)$ on $H(X)$ we are effectively spending $3n \log_2(\Sigma)$. It can be shown that there is no overhead spent with respect to the $H(Z|X)$ term in this idealized setting. This suggests that the gap to entropy for Orcom should stay constant with coverage, and this is approximately true for simulated noiseless datasets (Table 5 in paper).

## Analysis of proposed algorithm for noiseless reads

If the genome has no repeats of length $k$, then the proposed algorithm will find the read closest to the current read on the genome with high probability (as long as read is not shifted by more than $maxshift$. The first read can be from any position in the genome and hence the algorithm will find matches till it reaches the end of genome. After that it will pick some read from the part of the genome before the first read and so on. It can be shown that on average the number of unmatched reads will be $O(\log_2(n))$ where $n$ is the length of the genome.