Fq_delta – Efficient storage of processed versions of fastq files





Andra Veraart[1,2]✉, Henk-Jan van den Ham[2], Maarten A. Bijl[2], Arno C. Andeweg[2], Anita C. Schürch[2]

School of Communication, Media and Information Technology, Rotterdam University, Rotterdam, The Netherlands
[2]Dept. Viroscience, Erasmus Medical Center, Rotterdam, The Netherlands

Received 11 July 2013; **Accepted** 2 December 2013; **Published** 5 March 2014

Veraart A *et al.*. (2014) *EMBnet.journal* **20**, e698. http://dx.doi.org/10.14806/ej.20.0.698

Competing Interests: the authors have declared that no competing interests exist.

Abstract

This technical note describes fq _delta, a python module and shell script that enables the storage of processed versions of fastq files generated by DNA and RNA sequencing technologies. By using Myer's diff algorithm to perform per-character comparisons between the original and processed fastq files, we generate delta files that describe the changes in the processed fastq file relative to the original file. While the delta files are only a fraction of the original size (0.1 – 3%), they allow lossless reconstruction of the processed fastq files. Depending on the number of processing steps, implementation of this module will lead to a significant reduction in storage required for processing sequence data.

Availability:
Fq _delta is available for download at https://github.com/averaart/fq_delta.

Introduction

Advances in sequencing technology have led to an exponential increase in the volume of sequencing data that is generated (Wetterstrand, 2013). The amount of data that is now generated poses a challenge to storage facilities, especially for primary data. Currently, the cost of sequencing is dropping faster than the cost of storage space, and will probably continue to do so in the near future (Komorowski, 2009). Additionally, data processing can require storage of intermediate analysis steps. Data compression reduces the need for storage capacity, and several compression methods have been applied to raw sequence data (Grassi *et al.*, 2012; Bhola *et al.*, 2011; Jones *et al.*, 2012; Howison, 2012; Bonfield & Mahoney, 2013; Hach *et al.*, 2012). The processing of sequence data typically consists of several steps. Runs are often split into separate samples, followed by removing sequence tags or trimming of low quality reads. Examples of popular pre-processing software include cutadapt (Martin, 2011), the FASTX-toolkit (Gordon & Hannon, 2010) or Biopython (Cock *et al.*, 2009). These create one or more versions of the original data file, thus tending to require several times the original storage capacity. To save storage capacity, processed versions are often discarded, but in many cases these files are saved to allow easy access to all intermediate steps without redoing the analysis. Moreover, a growing number of researchers advocate the publication of raw sequence data and code to improve reproducibility of results (Peng, 2011; Stodden, 2010; Barnes, 2010; Baggerly & Berry, 2011), which would be facilitated by having processed intermediate files available. Given that the differences between the original and a processed version of the data are often minor, storage and compression of only the differences between versions would be far more efficient than retaining complete versions.

For saving different versions of the same file, several general-purpose applications are available, but the specific type of manipulation that is performed in sequence data processing pre-

cludes their use. Processing often entails the removal of several bases from each read. Existing general purpose applications typically work on a line-basis or block-basis, *i.e.* a fixed number of bytes (*e.g.*, diff[1] and rdiff[2], respectively). If one base has changed, the complete line or block will be stored instead of only the changed bases. This behavior makes these applications inefficient for storing processing steps in sequence data analysis, and suggests that these data require a high-resolution method to efficiently save processed sequence data files.

Table 1. Size reduction achieved by storing the processed fastq file in a zip archive, by storing an rdiff delta file in a zip archive and by using fq _ delta. File sizes expressed in Megabytes. Percentages are based on the processed fastq file size, as indicated in the first column. The original, unprocessed fastq file was 802.5 MB.

	fastq	**zipped fastq**	**zipped rdiff delta**	**fq_delta**
fastq _ masker -q 10	765.29	225.58 (29%)	212.87 (28%)	7.19 (0.94%)
fastq _ masker -q 25	765.29	228.17 (30%)	227.49 (30%)	17.53 (2.3%)
fastq _ quality _ trimmer	740.10	221.32 (30%)	197.77 (27%)	2.11 (0.28%)
cutadapt	751.53	223.05 (30%)	89.68 (12%)	0.74 (<0.1%)
cutadapt trimmed only	41.59	11.47 (28%)	11.48 (28%)	0.82 (2.0%)
cutadapt untrimmed only	709.94	211.69 (30%)	81.38 (11%)	0.39 (<0.1%)
removed lines	306.58	90.67 (30%)	0.002 (<0.1%)	0.08 (<0.1%)

This paper describes fq_delta, a python module to store differences between versions of fastq files. This module compares strings on a per-character basis and stores differences between them, thereby saving all changes into a file that is a fraction of the processed fastq file. Storage of the original file and delta files therefore enables full reconstruction of processed versions of fastq files.

Design and Implementation

Fq_delta uses the google-diff-match-patch library (Fraser, 2009), which implements Myer's diff algorithm (Myers, 1986). Fq_delta applies this technique to fastq files.

Consider a given sequence line in a fastq file containing the following string:

```
ACACGTAGTATACGGCATGCTACG
```

Assume the first 11 characters comprise a sequence tag that needs to be removed before further analysis takes place, resulting in the following string:

```
ACGGCATGCTACG
```

In the delta file, this line would be listed as:

```
-11 =13
```

This describes that the first 11 characters of the original line are removed and the subsequent 13 characters remain the same. Storing this description uses far fewer bytes than storing the complete processed line. Even when storing a large number of minor changes, this method is more efficient than the standard command line diff application. The processed string can be reconstructed using the first string and the difference between the first and second strings, as documented in the delta file.

Generating a delta file

Fq_delta expects two fastq files as input. The first is assumed to be the original, the second a processed version: *i.e.*, fq_delta computes the delta of the second input file relative to the first. Four lines of each file are read, covering one sequence read. To ensure that related original and processed sequences are matched, the identifier lines (IDs) are first compared, excluding any tab-separated values. If the IDs match, the difference between each line is written to a text file, called the delta file. If the IDs do not match, the read from the original file has evidently been removed from the processed file (fq_delta thereby assumes that the original and processed fastq files have the same read order). This is written into the delta file and the next four lines are read from the original file. This process is repeated until the end of the processed file has been reached. An

1 pubs.opengroup.org/onlinepubs/9699919799/utilities/diff.html
2 linux.die.net/man/1/rdiff

md5 hash of the processed file is calculated and written to a separate checksum file so that data integrity can be verified when reconstructing the file (fq_delta raises an error when these do not match). Finally, the delta file and the checksum file are compressed into one standard zip archive.

Retrieving a processed file

Fq_delta expects two files during retrieval: the original fastq file, and the zip archive containing the delta file that represents the processed version. The delta file and checksum file are both extracted from the zip archive. If the checksum file is not found, the process is aborted immediately. The original file and the delta file are read line by line. The delta line is applied to the original line to reconstruct the processed line. The processed line is either written to a new file or printed to *standard out.* When the end of the delta file is reached, the process is stopped, effectively ignoring the last lines that were in the original but not in the processed versions.

Technical details

Fq_delta is written in Python 2.7 as a module. It provides a class that implements the same functions as typical file-like objects. The class is able to use *standard in* or *standard out* as input or output, respectively. Fastqfiles that are compressed using quip (Jones *et al.*, 2012) are decompressed on-the-fly. Fq_delta assumes an unchanged order of reads from one version to the other to generate the delta file. The Fq_delta module was tested in scenarios where data integrity was deliberately compromised. In all cases, the application detected the error and reportedit to the user.

Fq_delta can also be used as a command-line tool, using two additional shell scripts that are provided with the module. The python module, command line scripts and a test script are available at https://github.com/averaart/fq_delta.

Results & Discussion

The application was tested using fastq_masker and fastq_quality_trimmer from the FASTX-toolkit (Gordon & Hannon, 2010), and cutadapt (Martin, 2011) on the first 2,500,000 reads of a publicly available data-set (Uddenberg *et al.*, 2013) for Norway spruce (Picea abies; the test shell script is available from the codebase). To demonstrate that removed lines are handled correctly, irrespective of their location in the file, an extra test was performed where we removed 500,000 reads from the start, the middle and the end of the example set.

Using fq_delta, the processed files were successfully compressed and accurately reproduced, using the original file as a reference. Table 1 illustrates the file sizes of the processed files and the resulting delta files, showing a reduction in required storage of at least 97 percent.

The sizes of fq_delta files were compared with compressed versions of the processed files, and both uncompressed and compressed versions of diff and rdiff files. In most cases, the uncompressed diff and rdiff files were much larger than the compressed fastq files; only the compressed rdiff file was smaller in all cases (Table 1). The fq_delta files were an order of magnitude smaller in all cases, except the "removed lines" scenario.

The large difference between rdiff and fq_delta can be explained by the coarse- and fine-grained resolutions of the respective methods. The rdiff algorithm is too coarse to efficiently register small changes, whereas fq_delta works on a per-character basis. This is illustrated by the 'removed lines' test, where the fastq file was divided into five continuous sections, two of which were saved in the processed file. Only in this coarse-grained scenario did rdiff perform better than fq_delta.

In summary, fq_delta is able to store multiple versions of a fastq file at a fraction of the usual storage costs. None of the tools we are aware of show the same efficiency. There are no requirements to the fastq files, except that the order of the reads should be consistent between original and processed versions. Especially combined with compression of original files, fq_delta drastically reduces the amount of storage necessary when processing sequence data.

Acknowledgements

This study was supported by the Virgo consortium, funded by the Dutch government, project number FES0908, and by the Netherlands Genomics Initiative (NGI), project number 050-060-452.

References

Baggerly KA & Berry DA (2011) Reproducible Research | Amstat News. *AMSTAT News Blog*. http://magazine.amstat.org/blog/2011/01/01/scipolicyjan11/ (accessed 11 June 2013).

Barnes N (2010) Publish your computer code: it is good enough. *Nature* **467**, 753. http://dx.doi.org/10.1038/467753a

Bhola V, Bopardikar AS, Narayanan R, Lee K & Ahn T (2011) No-Reference Compression of Genomic Data Stored in F ASTQ Format IEEE. http://dx.doi.org/10.1109/bibm.2011.110

Bonfield JK & Mahoney M V (2013) Compression of FASTQ and SAM Format Sequencing Data. *PloS one* **8**, e59190. http://dx.doi.org/10.1371/journal.pone.0059190

Cock PJA, Antao T, Chang JT, Chapman BA, Cox CJ et al. (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics (Oxford, England)* **25**, 1422–3. http://dx.doi.org/10.1093/bioinformatics/btp163

Fraser N (2009) google-diff-match-patch - Diff, Match and Patch libraries for Plain Text. http://code.google.com/p/google-diff-match-patch/ (accessed 16 May 2013).

Gordon A & Hannon GJ (2010) FASTX-Toolkit. *FASTQ/A short-reads pre-processing tools*. http://hannonlab.cshl.edu/fastx _ toolkit/ (accessed 31 May 2013).

Grassi E, Gregorio F Di & Molineris I (2012) KungFQ: a simple and powerful approach to compress fastq files. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM* **9**, 1837–42. http://dx.doi.org/10.1109/tcbb.2012.123

Hach F, Numanagic I, Alkan C & Sahinalp SC (2012) SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics (Oxford, England)* **28**, 3051–7. http://dx.doi.org/10.1093/bioinformatics/bts593

Howison M (2012) High-Throughput Compression of FASTQ Data with SeqDB. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM* **10**(1), 213–218. http://dx.doi.org/10.1109/tcbb.2012.160

Jones DC, Ruzzo WL, Peng X & Katze MG (2012) Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic acids research* **40**, e171. http://dx.doi.org/10.1093/nar/gks754

Komorowski M (2009) A History of Storage Cost. http://www.mkomo.com/cost-per-gigabyte (accessed 31 May 2013)

Martin M (2011) Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet.journal* **17**, pp. 10–12. http://dx.doi.org/10.14806/ej.17.1.200

Myers EW (1986) AnO(ND) difference algorithm and its variations. *Algorithmica* **1**, 251–266. http://dx.doi.org/10.1007/bf01840446

Peng RD (2011) Reproducible research in computational science. *Science (New York, N.Y.)* **334**, 1226–7. http://dx.doi.org/10.1126/science.1213847

Stodden V (2010) Reproducible Research. *Computing in Science & Engineering* **12**, 8–13. http://dx.doi.org/10.1109/mcse.2010.113

Uddenberg D, Reimegård J, Clapham D, Almqvist C, Von Arnold S et al. (2013) Early cone setting in Picea abies acrocona is associated with increased transcriptional activity of a MADS box transcription factor. *Plant physiology* **161**, 813–23. http://dx.doi.org/10.1104/pp.112.207746

Wetterstrand KA (2013) DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP). http://www.genome.gov/sequencingcosts/ (accessed 14 May 2013).