Efficient algorithms for the compression of FASTQ files

Subrata Saha and Sanguthevar Rajasekaran
Department of Computer Science and Engineering
University of Connecticut, Storrs
Email: {subrata.saha,rajasek}@engr.uconn.edu

***Abstract*—Since the introduction of the Sanger sequencing technology in 1977 by Frederic Sanger and his colleagues, we observe an explosion of sequence data. The cost of storage, processing, and analyzing the data is getting excessively high. As a result, it is extremely important that we develop efficient data compression and data reduction techniques. But standard data compression tools are not suitable to compress biological data since they contain many repetitive regions. There could exist high similarities among the sequences. In this context we need specialized algorithms to effectively compress biological data. In this paper we propose novel algorithms for compressing FASTQ files. We have done extensive and rigorous experiments that reveal that our proposed algorithm is indeed competitive and performs better than the best known algorithms for this problem.**

I. Introduction

Genomic sequencing technologies have provided significant breakthroughs in our understanding of biological systems at the molecular level and are important for personalized medicine. We now know the major mutational processes involved in the pathogenesis of most cancers and effective biomarkers for diagnosis and prognosis have been developed. However, to further harness the power of biomedical big-data to develop safer and more efficient personalized therapies, we need faster and more effective algorithms to compress and search the big data sets, especially DNA/RNA sequencing data, generated by various genomics projects. For example, at the Jackson Laboratory for Genomic Medicine (JGM) researchers generate large sequencing data sets from different cell lines, patient derived xenograft (PDX) and tumor models, patient tumors and mouse disease models. NCBI houses petabytes of genomic data and biologists around the world are generating 15 petabases of sequence per year (www.simonsfoundation.org/quanta/20131007-our-bodies-our-data/). The size of metagenomic data from multiple samples could be petabytes.

The drop in sequencing cost is outpacing the drop in storage cost. The cost of storage, processing and analyzing these data is getting formidable. As a result, it is extremely important that we develop efficient data compression techniques. For example, the BAM file format uses a lossless compression method. This reduces the storage requirement by 50%. However, the growth of storage and computing resources cannot keep up with the growth of the sequencing projects at many genome centers even with the use of BAM files. Different versions of the data compression problem have been addressed by bioinformaticians. In this paper we offer a novel algorithm for FASTQ files compression. FASTQ format is a text-based format to store both biological sequences (such as nucleotide sequences) and their corresponding quality scores. Our algorithm achieves better compression ratios than the currently best-known algorithms. (By compression ratio we mean the ratio of the uncompressed data size to the compressed data size).

A FASTQ file contains read sequences, quality scores, and metadata information. We deal with the FASTQ file compression problem with three different algorithms, one for compressing each type. These three algorithms are called RFRC (Reference Free Reads Compressor), RFQSC (Reference Free Quality Scores Compressor), and MDC (Metadata Compressor). Collectively, all of these three algorithms are combined into a single algorithm called FQC (FASTQ Compressor).

In RFRC, reads are clustered based on a hashing scheme. Followed by this clustering, a representative string is chosen from each cluster of reads. Compression is independently done for each cluster. In particular, the representative string in any cluster is used as a reference to compress other reads in this cluster. RFQSC also compresses quality score sequences by forming clusters of similar sequences. Here clusters are formed without the employment of hashing. Each sequence in a particular cluster is then compressed with respect to its representative sequence. The representative sequence can be a brand new sequence or one of the quality sequences residing in the cluster. Metadata is compressed by detecting the redundant information. Simulation results show that FQC performs better than some of the best known algorithms existing in the current literature.

The rest of this paper is organized as follows: In Section II we provide a brief survey of compression algorithms that have been proposed in the literature. Details of our novel algorithms are presented in Section III. The performance of our algorithm and experimental results are presented in Section IV. Section V concludes the paper.

II. Related Works

Five different problems of data compression in the context of NGS data have been proposed in the literature. 1) *Genome compression with a reference*, 2) *Reference-free Genome Compression*, 3) *Reference-free Reads Compression*, 4) *Reference-based Reads Compression*, and 5) *Metadata and Quality Scores Compression.* We now survey some of the algorithms that have been proposed to solve problem 3 and problem 5.

1) Reference-free Reads Compression: Reads compression methods can be categorized into reference-based and non reference-based, similar to genome compression techniques. In a reference-based technique we assume the presence of a reference genome sequence. Each read is mapped to this reference and only the difference between the read and its mapped substring in the reference is stored. G-SQZ method has been proposed by Tembe *et al.* [1] where the frequency of each unique tuple <base, quality> is computed. Each tuple is then encoded by generating a Huffman code such that if a tuple is more frequent then the number of bits needed to encode this tuple will be less. After this, the encoded tuples along with a header containing meta-information like the platform and the number of reads are written to a binary file. The DSRC algorithm [2] divides the input into blocks of 32 records and every 512 blocks are clustered to make a superblock. Each of the superblocks is then indexed and compressed independently using LZ77 coding scheme. Quip [3] uses an arithmetic encoding scheme based on high order Markov chains. Fqzcomp and Fastqz are proposed by Bonfield and Mahoney [4]. In these methods, the sequences are compressed by exploiting an order-N context model and an arithmetic coder. BEETL is an algorithm of Coax *et al.* [5]. In this algorithm, repeats among the reads are identified using the Burros-Wheeler Transform (BWT) data structure [6]. The transformed data are then compressed exploiting general purpose compression algorithms like gzip, bzip2, or 7zip. A similar algorithm is SCALCE [7] where the consistent parsing algorithm [8] is used to find an identical longest 'core' substring from the clustered reads. The reads within a cluster are then compressed using other standard compression algorithms.

2) Metadata and Quality Scores Compression: Standard NGS sequencing datasets can be thought of as sets of records where each record contains a read ID, a read sequence, and a quality sequence. In addition to these records, a dataset will also contain metadata such as the read platform used, project ID, etc. Each dataset is stored in FASTQ or SAM/BAM format. Metadata are typically compressed using general purpose text compression algorithms such as Delta, Huffman, arithmetic, run-length, and LZ encoding. Some read compression algorithms discard parts of the metadata if such discards do not affect the downstream analysis [7] [3]. As the alphabet size is large (typically 40), general purpose text compression is used to compress the quality score sequences. Some algorithms such as G-SQZ, SlimGene, and DSRC use different coding techniques to compress quality score sequences as described in previous sections. The reference based method CRAM stores quality score variations with respect to reference quality score sequences and a user-specified percent of quality scores identical to the reference. Quip employs third-order Markov-chain based arithmetic encoding whereas QScores-Archiver [13] employs three lossy transformations based on coarse binning.

III. Our Algorithms

A. Reference-free Reads Compression

In this section we provide details of our reference-free reads compression algorithm namely RFRC. There are 3 steps in our novel algorithm. The algorithm clusters the reads based on overlaps and similarity scores. For each cluster a consensus sequence is created. All the reads in the cluster are compressed using the consensus as the reference. I.e., for each read we only store its difference with the consensus. Two reads are said to be *neighbors* of each other if they have a large overlap (with a small Hamming distance in the overlapping region). We find the neighbors of each read in steps 1 and 2 and use this neighborhood information to cluster the reads and perform compression. In step 1 we find the potential neighbors of each read and in step 2 we find the neighbors each read. More details follow.

In the first step we generate k-mers of each read and hash the reads based on these k-mers. Hence equal k-mers fall into the same bucket. Let R be the set of reads. If $r \in R$ is any read, then any other read falling into at least one of the buckets that r falls into will be treated as a potential neighbor of r. We thus create a list $P(r)$ of potential neighbors for every read $r \in R$. If the size of a bucket is larger than a user defined threshold τ, we retain only a subset of the bucket in this potential neighbors identification process. In the second step we align every read in $P(r)$ with r to find the neighbors of r. Let r' be any read in $P(r)$. If r and r' overlap sufficiently, i.e., the Hamming distance between r and r' in the overlapping region is $\leq d$ (d being a user defined threshold), we treat r' as a neighbor of r. For every read $r \in R$ we construct a list $L(r)$ of neighbors of r. In the third step we greedily align r' with r for every $r' \in L(r)$. The consensus string r_c is then built. Subsequently, r along with reads in $L(r)$ are compressed using r_c as a reference. This is done for every read $r \in R$. Note that in the above step, a read may be present in more than one clusters. In this case, the read is compressed in that cluster where its compressed length will be the least. At the end the compressed reads are further compressed using delta encoding and the PPMD compression algorithm which is a variation of the Prediction by partial matching (PPM) algorithm. PPMD is an adaptive statistical data compression technique based on context modeling and prediction. It is freely available in 7-zip compression tools package.

B. Quality Scores Compression

Every read output by any sequencing machine also has an associated quality sequence. Specifically, there is a quality score for each base. This score can be thought of as the probability that this base is correct. As the alphabet size for the quality sequences is large (typically 40), a general purpose text compression algorithm is typically used to compress the quality score sequences. In this section we describe our novel algorithm to compress quality score sequences. This algorithm achieves a better compression ratio than the compression algorithms currently used to compress quality sequences. The algorithm has two steps. At first we count the frequency of each character in the quality score sequences and identify the top m characters based on frequencies (where the value of m is chosen to get the best compression). Let the set of these characters be $\{c_1, c_2, \ldots, c_m\}$. We then build m representative quality score sequences $s_1, s_2, \ldots, s_m$, where s_i consists of only c_i's (for $1 \leq i \leq m$). For example, s_1 will be $c_1, c_1, \ldots, c_1$. For each of these m sequences we create a list of 'neighbors'. Let $L(s_i)$ denote the list of neighbors of s_i, $1 \leq i \leq m$. For each quality sequence s in the input we compute the Hamming distance between s and each of the sequences $s_1, s_2, \ldots, s_m$. If the Hamming distance between s and s_i is the least and if this distance is less than a threshold, we store s in the neighbor list of s_i. The quality sequences in $L(s_i)$ are then compressed using s_i as the reference (for $1 \leq i \leq m$). Specifically, if $s \in L(s_i)$ for some specific i, then we only store the difference between s and s_i.

Clearly, some of the quality scores sequences will not be neighbors to any of the representative sequences $s_1, s_2, \ldots, s_m$. We use a different algorithm to compress these. Let the set of sequences without any neighbors be S''. We divide S'' into q parts of equal size. Let the collection of these parts be P. For each part $p \in P$ we cluster $s \in p$ into buckets and choose a representative in each bucket. Let the collection of these buckets be B_p. Quality sequences in any bucket will be compressed using the representative as the reference. Bucketing of quality sequences is done as follows. For each of the sequences s from p we insert it into a bucket $b \in B_p$ where the Hamming distance between s and the representative sequence of b is minimum and below some threshold d. If no such bucket exists, a brand new bucket is created where s will be the representative sequence. At the end compressed sequences are further compressed using delta encoding and the PPMD compression algorithm as stated above.

C. Metadata Compression

Metadata contain unique read/quality sequence identifiers, name of the platform used, project id, read length, etc. Some of the information is identical for all the reads and the quality score sequences such as the name of the platform, project id and read length. This redundant information is saved only once. Read/quality score sequence identifiers are typically in increasing order in the FASTQ file. Hence we have compressed this information using delta encoding. Compressed metadata are further compressed by using the PPMD compression algorithm.

We have combined RFQSC (for quality scores compression), RFRC (for reads compression), and the above technique (for metadata compression) to obtain an algorithm for compressing FASTQ files. We call this algorithm FQC (FASTQ Compressor).

IV. Results and Discussions

We have compared our algorithm FQC with the best known algorithms existing in the literature for FASTQ data compression. In this section we summarize the results.

A. Simulation Environment

All the experiments were done on an Intel Westmere compute node with 12 Intel Xeon X5650 Westmere cores and 48 GB of RAM. The operating system running was Red Hat Enterprise Linux Server release 5.7 (Tikanga). RFRC compression and decompression algorithms are written in C++ and standard Java programming language, respectively. Both of RFQSC and MDC compression and decompression algorithms are written in standard Java programming language. To compile the C++ source code we used g++ compiler (gcc version 4.6.1) with the -O3 option. Java source code is compiled and run by Java Virtual Machine (JVM) 1.6.0.

B. Datasets

We have employed real datasets in our evaluation. Real datasets used are Illumina-generated short reads of various lengths. The eight experimental datasets listed in Table I have been taken from Sequence and Read Archive (SRA) at NCBI. Reference genomes are Sanger assembled bacterial genomes of various kinds. Please, note that G_R, $|G|$, $|R|$, and $|r|$ refer to accession number of the reference genome, genome length, total number of reads/quality sequences, and read/quality sequence length, respectively.

C. Simulation Results

We have compared our algorithm with 3 other well-known algorithms using real reads and their associated quality scores. Quip is known to be one of the most efficient FASTQ files compression algorithms in the current literature. We have done extensive and rigorous experiments to realize that FQC is indeed an effective and competitive FASTQ files compression algorithm. Real sequencing data are taken from Sequence Read Archive (SRA) as described above. The results for the datasets taken from NCBI can be found in Table II. It is evident from the results shown in Table II that FQC performs better than Gzip, Bzip2, and Quip in all the datasets. It compresses these datasets 1.2 to 2.7 times better than Quip which is one of the best-known and state-of-the-art algorithms in the domain of FASTQ compression.

Dataset	Name	Accession	G_R	$\|G\|$	$\|r\|$	$\|R\|$	Coverage
D1	*E. coli*	SRR001665	NC_000913.2	4,639,675	36	10,359,952	78.00
D2	*L. lactis*	SRR088759	NC_013656.1	2,598,144	36	4,370,050	60.55
D3	*E. coli*	SRR396536	NC_000913.2	4,639,675	75	3,454,048	55.83
D4	*E. coli*	SRR022918	NC_000913.1	4,771,872	47	6,717,270	66.00
D5	*E. coli*	SRR396532	NC_000913.2	4,639,675	75	4,341,061	70.17
D6	*T. pallidum*	SRR361468	CP002376.1	1,139,417	35	7,013,188	219.13
D7	*L. interrogans L*	SRR353563	NC_004342.2	4,338,762	100	3,522,192	81.18
D8	*L. interrogans C*	SRR397962	NC_005823.1	4,277,185	100	3,511,477	81.98

TABLE I: Illumina generated reads and quality score sequences from different bacterial genomes. G_R, $|G|$, $|R|$, and $|r|$ refer to accession number of the reference genome, genome length, total number of reads (or quality scores sequences), and read length (or quality scores sequence length), respectively.

Dataset	Size (MB)	Gzip	Bzip2	Quip	FQC
D1	1280	4.3	5.3	6.9	**10.2**
D2	777	5.0	5.8	7.1	**11.8**
D3	914	4.3	4.9	4.0	**10.0**
D4	1403	5.0	5.9	9.1	**11.0**
D5	1147	4.3	5.0	5.3	**11.8**
D6	1300	3.9	4.4	4.7	**7.1**
D7	1096	3.7	4.2	3.0	**7.9**
D8	1096	3.8	4.3	3.1	**8.3**

TABLE II: Compression ratios offered by Gzip, Bzip2, Quip, and the proposed algorithm FQC. The best results are shown in boldface.

V. Conclusions

The exponential growth of high-throughput DNA sequence data poses great challenges in genomic data storage, retrieval, and transmission. Compression is a critical tool to address these challenges, where many methods have been developed to reduce the storage size of the genomes and sequencing data (including reads, quality scores, and metadata). In this article we have proposed an efficient and scalable algorithm namely FQC to compress FASTQ files. Simulation results show that our algorithm is indeed effective, efficient, and competitive with respect to state-of-the-art algorithms existing in the domain of FASTQ file compression.

Acknowledgements

This research has been supported in part by the following grants: NIH R01-LM010101 and NSF 1447711. The authors thank Krishna Karuturi and Joshy George (from Jackson Laboratory) for many fruitful discussions.

References

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

[2] Deorowicz,S. and Grabowski,S. (2011) Compression of DNA sequence reads in FASTQ format. *Bioinformatics*, **27**, 860-862.

[3] Jones,D.C., Ruzzo,W.L., Peng,X., and Katze,M.G. (2012) Compression of nextgeneration sequencing reads aided by highly efficient de novo assembly. *Nucleic Acids Res*, **40**, e171.

[4] Bonfield,J.K., and Mahoney,M.V. (2013) Compression of FASTQ and SAM format sequencing data. *PLoS One*, **8**, e59190.

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

[6] Burrows,M., and Wheeler,D.J. (1994) A block-sorting lossless data compression algorithm. *SRC Research Report*.

[7] Hach,F., Numanagic,I., Alkan,C., and Sahinalp,S.C. (2012) SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics*, **28**, 3051-3057.

[8] Sahinalp,S.C., and Vishkin,U. (1996) Efficient approximate and dynamic matching of patterns using a labeling paradigm. *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pp. 320-328.

[9] Langmead,B., Trapnell,C., Pop,M., and Salzberg,S.L. (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, **10**, R25.

[10] Li,H., and Durbin,R. (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**, 1754-1760.

[11] Fritz,M.H.-Y., Leinonen,R., Cochrane,G., and Birney,E. (2011) Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Res*, **21**, 734-740.

[12] Popitsch,N., and Haeseler,A.V. (2013) NGC: lossless and lossy compression of aligned high-throughput sequencing data. *Nucleic Acids Res*, **41**, e27.

[13] Janin,L., Rosone,G., and Cox,A.J. (2014) Adaptive reference-free compression of sequence quality scores. *Bioinformatics*, **30**, p24.