G-SQZ: compact encoding of genomic sequence and quality data

Waibhav Tembe*, James Lowey and Edward Suh

Translational Genomics Research Institute, 445 N 5th Street, Phoenix, AZ 85004, USA

Associate Editor: Joaquin Dopazo

ABSTRACT

SUMMARY: Large volumes of data generated by high-throughput sequencing instruments present non-trivial challenges in data storage, content access and transfer. We present G-SQZ, a Huffman coding-based sequencing-reads-specific representation scheme that compresses data without altering the relative order. G-SQZ has achieved from 65% to 81% compression on benchmark datasets, and it allows selective access without scanning and decoding from start. This article focuses on describing the underlying encoding scheme and its software implementation, and a more theoretical problem of optimal compression is out of scope. The immediate practical benefits include reduced infrastructure and informatics costs in managing and analyzing large sequencing data.

Availability: http://public.tgen.org/sqz. Academic/non-profit: Source: available at no cost under a non-open-source license by requesting from the web-site; Binary: available for direct download at no cost. For-Profit: Submit request for for-profit license from the web-site.

Contact: wtembe@tgen.org

1 INTRODUCTION

High-throughput sequencing methods and instruments (Ansorge, 2009; Mardis, 2008; Shendure and Ji, 2008) generate hundreds-of- millions to billions of short reads. Storage, management and transfer of such large volumes of data necessitate an order of magnitude more infrastructure capabilities. Algorithms and software are required to analyze terabytes of data within practical constraints of resources, such as computer processors, memory, storage, time, etc., and have to cope with the new challenge of handling very large files containing dense data. With constantly increasing throughput, monetary costs of processing and managing sequencing data are becoming a larger factor in research planning. To that end, this article proposes a lossless, order preserving and compact-encoding scheme called Genomic SQueeZ (G-SQZ) for sequence read data and provides a software implementation.

Typically, genomic sequencing data consists of annotated genomic bases in their relative order. Annotation is required due to reasons such as (i) nucleotide base calling algorithms typically report error probability, generally called quality score, to account for intrinsic uncertainties in sequence identification processes; (ii) additional information needs to be reported when sequence identification is inconclusive, incomplete or erroneous; and (iii) for various downstream analyses, assigning unique

identifiers to sequencing reads is desirable. This overhead of storing annotation information dramatically increases file sizes and any storage reduction approach requires compressing both base and annotation data.

2 METHODS

Huffman coding (Huffman, 1952) is a lossless encoding algorithm that generates variable length codes for symbols used to represent information. By encoding high frequency symbols with shorter codes and low frequency symbols with longer codes, the original information is stored as much smaller encoded data. The codes are constructed in such a way that no code is a prefix of any other code, a property that enables unambiguous decoding. Within the context of genomic sequencing data, let $B = \{b_1, b_2, \ldots, b_m\}$ denote a finite set of m symbols representing all possible distinct genomic bases (nucleotide, color-space, error calls, no calls, etc.) and let $Q = \{q_1, q_2, \ldots, q_p\}$ be a finite set of symbols representing all distinct quality scores or other annotation information available for each base. Since not all symbols from B and Q might be present in a given data, let $B' \subseteq B$ and $Q' \subseteq Q$ denote the actual symbols used, such that $|B'| = m'$; $1 \leq m' \leq m$ and $|Q| = p'$; $1 \leq p' \leq p$. We construct a pair-wise symbol set $S = B' \times Q'$ of size $n = m' \cdot p'$ that consists of all base-quality pairs $< b_i, \ q_j >$; $b_i \in B'$, $q_j \in Q'$ in the data.

As shown in Figure 1, the first scan generates a unique Huffman code for each $<base, quality>$ pair from the frequency distribution of all pairs in S. In the second scan, encoded pairs are written to a binary file, along with a header containing meta-information, such as Huffman encoding for each pair, platform, number of reads, etc. Additionally, the meta-characters ('@', '+', '>', '_', ':', etc.) are stored only once and the differences between successive read identifiers are noted, taking care that all original information can be reproduced. Thus, the encoded file (Fig. 1) consists of a fixed-length header followed by sequence of blocks, one block per read. Information in the header can be retrieved via simple queries to retrieve number of reads, base/quality statistics, sample name, etc., and save time in parsing large files.

3 IMPLEMENTATION AND RESULTS

A C++ command-line implementation of G-SQZ is available at http://public.tgen.org/sqz. Since more than 4 billion ($\sim 2^{32}$) bases per run are common, G-SQZ has been designed as a 64-bit application. Table 1 shows comparisons with gzip v1.3.5 and bzip2 v1.0.5 using best compression option (-9) for both on publicly available data from the 1000 Genomes Project (www.1000genomes.org). Tests have shown similar storage reduction on several other data sets (data not shown), and we invite readers to test G-SQZ on additional data. While the encoding step for base-quality data stays the same, read identifier format specific steps have been implemented for SRA (NCBI), CSFASTA/QUAL (LifeTechnologies) and FASTQ (Cock *et al.*, 2009) formats. Due to non-standardization in read-identifier formats, capturing all possible formats is a work in progress.

---

*To whom correspondence should be addressed.
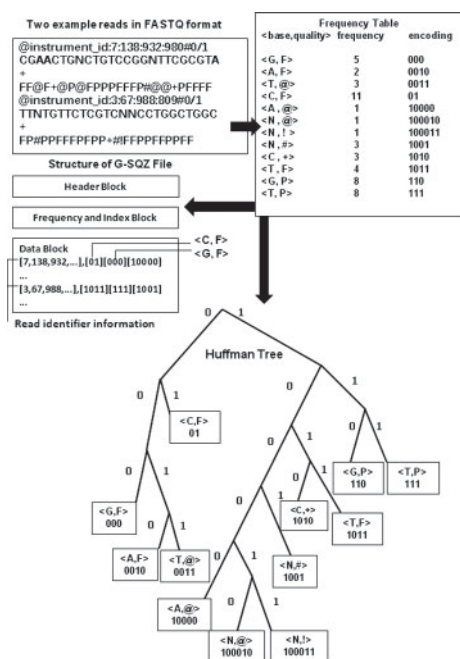
Fig. 1. G-SQZ encoding: first scan calculates frequency for each <base, quality> pair and constructs Huffman tree to generate pair-specific codes. High-frequency pairs have shorter codes. Second scan writes header and encoded read blocks to an output binary file. Each read block consists of read identifier fields followed by encoded base-quality data.

Table 1. Comparison of encoded data sizes (GB)

Data set (FASTQ/SRA format)	Platform	Original size	Encoded size (Percentage of reduction)		
			gzip -9	G-SQZ	bzip2 -9
SRR013951_2.filt	SOLEXA	3.19	1.32 (59)	1.12 (65)	1.13 (65)
SRR027520_1.filt	SOLEXA	4.80	1.67 (65)	1.54 (68)	1.40 (71)
SRR027520_2.filt	SOLEXA	4.80	1.71 (64)	1.54 (68)	1.44 (70)
SRR007215_1.filt	SOLiD	0.69	0.16 (77)	0.13 (81)	0.13 (81)
SRR010637.filt	SOLiD	2.08	0.59 (72)	0.49 (76)	0.49 (76)
SRR014961_2.filt	SOLiD	40.9	11.5 (72)	9.64 (76)	9.35 (77)

4 DISCUSSION

The salient features of G-SQZ are: (i) Specifically designed for sequencing reads in known format(s) and not for arbitrary data; (ii) implicit pairing and simultaneous encoding of base-quality data; (iii) other than counting frequency of <base, quality> pairs, no other string matching is employed; (iv) order of <base, quality> data is unchanged; and (v) data can be retrieved selectively without serial scans and decoding from the beginning, a feature that can be utilized by multi-threaded parallel computing applications.

G-SQZ is designed for sequencing reads, unlike aligned sequences in SAM/BAM (Li *et al.*, 2009) formats. Since no restriction is imposed on the number of symbols, it is possible to include any number of distinct bases (including other IUPAC symbols, error symbols, etc.) and larger range covering higher

resolution of quality scores. Quality values can be multi-character strings (with known delimiters), not limited by the size of the ASCII character set. Generating *<base, quality>* pair-specific Huffman encoding offers two significant advantages over two-bit encoding (A = 00, C = 01, G = 10, T = 11). For example, two-bits per base will require 20 bits to store a 10-base sequence AAAGTAATAA, while frequency distribution-aware variable length encoding, such as A = 0, G = 10, T = 11, will reduce the number of bits to 13. Second, two bits can encode at most four distinct symbols and, without additional bits, cannot include *no call*, *error call* and quality scores.

G-SQZ offers a compact lossless indexed format to significantly reduce storage needed by plain text files, while allowing selective access to any section of the data without serially scanning and decoding from start. The compression ratio of G-SQZ primarily depends on the relative frequencies of *<base, quality>* pairs. We recognize that G-SQZ might be sub-optimal compared to other compression algorithms employing statistical pattern matching, such as palindromes, string comparisons, repeat detection and data permutation (Adjeroh *et al.*, 2002; Brandon *et al.*, 2009; Chen *et al.*, 2002; Christley *et al.*, 2009; Soliman *et al.*, 2009). For example, for bases AAATTGAAA and qualities FFFFFFFFF, storing <multi-base, multi-quality> pair <AAA,FFF> instead of <A,F> is more efficient, and several such cases can be enumerated. However, investigation of an order preserving, indexed, yet theoretically optimal, encoding and extensive comparisons with all compression methods is out of scope. The goal of this article is to demonstrate the *practical application* of G-SQZ to a critical challenge in bioinformatics, where throughput per run is in the range of hundreds of Giga-bases, but the plain text data format is highly inefficient. As shown in Table 1, G-SQZ has outperformed gzip in all selected data sets. Compared to bzip2 (a series of compression techniques that include the Burrows–Wheeler Transform), the relatively simple approach in G-SQZ did better in one case and came close overall while maintaining the order, allowing selective access and storing meta-information, e.g., number of reads/bases, for quick retrieval.

In cases of data corruption due to bit errors, G-SQZ can detect limited number of cases where, due to corruption, an encoded string is absent from the list of known symbols or values are out of normal ranges. For more reliable checks, we recommend external tools (e.g. md5). A robust error checking scheme and encoding variable length reads (e.g., 454 data) have been left as future work.

REFERENCES

Adjeroh,D. *et al.* (2002) DNA sequence compression using the burrows-wheeler transform. *Proc. IEEE Comput. Soc. Bioinform. Conf.*, **1**, 303–313.

Ansorge,W.J. (2009) Next-generation DNA sequencing techniques. *N. Biotechnol.*, **25**, 195–203.

Brandon,M.C. *et al.* (2009) Data structures and compression algorithms for genomic sequence data. *Bioinformatics*, **25**, 1731–1738.

Chen,X. *et al.* (2002) DNACompress: fast and effective DNA sequence compression. *Bioinformatics*, **18**, 1696–1698.

Christley,S. *et al*. (2009) Human genomes as email attachments. *Bioinformatics*, **25**, 274–275.

Cock,P.J. *et al*. (2009) The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res.*, **38**, 1767–1771.

Huffman,D. (1952) A method for the construction of minimum-redundancy codes. *Proc. IRE*, **40**, 1098–1102.

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

Mardis,E.R. (2008) Next-generation DNA sequencing methods. *Annu. Rev. Genomics Hum. Genet.*, **9**, 387–402.

NCBI (2010) National Center for Biotechnology Info. Available at http://www.ncbi.nlm.nih.gov/sra (last accessed date February 2, 2010).

Shendure,J. and Ji,H. (2008) Next-generation DNA sequencing. *Nat. Biotechnol.*, **26**, 1135–1145.

Soliman,T.H. *et al*. (2009) A lossless compression algorithm for DNA sequences. *Int. J. Bioinform. Res. Appl.*, **5**, 593–602.