BIND – An algorithm for loss-less compression of nucleotide sequence data

TUNGADRI BOSE, MONZOORUL HAQUE MOHAMMED, ANIRBAN DUTTA and SHARMILA S MANDE*

Bio-Sciences R&D Division, TCS Innovation Labs, 54B Hadapsar Industrial Estate, Tata Consultancy Services Limited, Hadapsar, Pune 411 013, India

**Corresponding author (Fax, +91-20-66086399; Email, sharmila@atc.tcs.com; sharmila.mande@tcs.com)*

Recent advances in DNA sequencing technologies have enabled the current generation of life science researchers to probe deeper into the genomic blueprint. The amount of data generated by these technologies has been increasing exponentially since the last decade. Storage, archival and dissemination of such huge data sets require efficient solutions, both from the hardware as well as software perspective. The present paper describes BIND – an algorithm specialized for compressing nucleotide sequence data. By adopting a unique 'block-length' encoding for representing binary data (as a key step), BIND achieves significant compression gains as compared to the widely used general purpose compression algorithms (gzip, bzip2 and lzma). Moreover, in contrast to implementations of existing specialized genomic compression approaches, the implementation of BIND is enabled to handle non-ATGC and lowercase characters. This makes BIND a loss-less compression approach that is suitable for practical use. More importantly, validation results of BIND (with real-world data sets) indicate reasonable speeds of compression and decompression that can be achieved with minimal processor/memory usage. BIND is available for download at *http://metagenomics.atc.tcs.com/compression/BIND*. No license is required for academic or non-profit use.

[Bose T, Mohammed MH, Dutta A and Mande SS 2012 BIND – An algorithm for loss-less compression of nucleotide sequence data. *J. Biosci.* **37** 785-789] **DOI** 10.1007/s12038-012-9230-6

1. Introduction

By drastically reducing the cost of DNA sequencing, Next Generation Sequencing (NGS) platforms have revolutionized the way in which life sciences research is being performed. Data generated using NGS platforms have been speculated to contain the scientific breakthroughs of the future (Metzker 2010; Zhang *et al*. 2011). Unlike sequence data sets in the 1990s, whose maximum size rarely exceeded a few megabytes, the size of present day sequence data sets (generated using NGS platforms) is in the range of gigabytes (GB). A typical research laboratory, using an NGS platform to its optimal capacity, is expected to generate a minimum of 100–150 terabytes (TB) of raw sequence data per year. This figure excludes interim processed data, which by itself is several times the size of raw data. Raw sequence data is generally pre-processed and then assembled into contigs. These steps result in generation of reasonable length contigs and/or complete genome sequences. Sequencing data (in raw/assembled format) from a project are either archived by the respective laboratory and/or submitted to major public sequence repositories such as GenBank (NCBI), EMBL, DDBJ, etc. These repositories are currently witnessing exponential growth with respect to the number of sequences being submitted each year. For instance, during the year 2010, the total number of bases (in these sequence repositories) had shown a threefold increase and the number of assembled sequences increased to 195 million (Cochrane *et al*. 2011). It is significant to note that the number of genomes (both prokaryotic and eukaryotic) available in the public domain

Keywords. Block-length encoding; data compression; delta-encoding; genome compression; genome sequences; unary-coding

Supplementary materials pertaining to this article are available on the *Journal of Biosciences* Website at *http://www.ias.ac.in/jbiosci/sep2012/supp/bose.pdf*

Published online: 13 August 2012

is also growing rapidly. This indicates the need for development of algorithms/approaches for efficient compression, storage, retrieval and dissemination of genomic sequence data.

Genome sequence data is typically stored as flat (text) files, 'fasta' being the most preferred format. Currently, several available General Purpose Compression Algorithms (GPCAs), in the form of user-friendly software implementations (e.g. gzip, bzip2, lzma, etc.), are able to compress a variety of data/text formats, including genomic sequence files. However, it is observed that the compression ratio achieved by GPCAs with genomic sequence data is sub-optimal. For example, gzip, bzip2 and lzma compress a 9.7 MB prokaryotic genome file (*Solibacter usitatus* Ellin6076) to 2.9, 2.8 and 2.6 MB, respectively. This indicates a compression ratio (CR) that is significantly higher than 2 bits/nucleotide base (i.e. CR>25%). In contrast, representing each of the four major nucleotide bases (A, T, G and C) with two bits (00, 01, 10 and 11) would directly generate a file with a better compression ratio as compared to that using GPCAs. Given this observation, several specialized genome compression approaches have been suggested in the last two decades. Some of these methods have been reasonably successful in achieving compression ratios significantly lower than 2 bits/base (Grumbach and Tahi 1994; Rivals *et al*. 1996; Matsumoto *et al*. 2000; Chen *et al*. 2002; Manzini and Rastero 2004; Behzadi and Le Fessant 2005; Cao *et al*. 2007; Korodi and Tabus 2007; Pinho *et al*. 2008, 2011). By capturing the exact/inexact repeat patterns prevalent within biological sequences, these specialized approaches are able to achieve compression ratios better than those achieved using GPCAs.

Although specialized genome compression approaches provide better compression gains (as compared to GPCAs), they are seldom deployed for compressing real-world data sets. The possible reasons behind this observation are as follows. Besides the four standard nucleotide bases (A, T, G and C), present day sequence data sets (generated using NGS technologies) frequently contain non-ATGC characters. Furthermore, repetitive stretches and low-complexity regions are usually indicated by lowercase characters in the sequence data. Gaps of indeterminate length (typically observed in contig sequences) are usually represented using a hyphen (-) character. These additional characters (besides A, T, G and C) are currently not handled by most of the existing specialized genome compression approaches. Consequently, it is practically impossible to ensure a loss-less retrieval of sequence data when such specialized compression approaches are employed. Moreover, the compression/decompression time and the memory requirements of most of these specialized approaches are also very high.

In this article, we present BIND – a new algorithm for compression of genome sequence data sets (in fasta/multi-fasta format). BIND follows a unique 'block-length encoding' methodology and is observed to yield significant gain in compression ratio (less than 2 bits/base) with a reasonable compression/decompression time. Existing genome compression approaches first capture repeat patterns and then encode them using an optimal bit-level encoding scheme. Unlike these approaches, BIND first performs a bit-level encoding of the four most frequent nucleotide bases (A, T, G and C) in the form of two individual data streams. BIND subsequently captures repeat patterns in these data streams by re-representing them in a novel 'block-length encoded' format. The BIND algorithm has been implemented using 'C' programming language, and its executables are available for download. This implementation is enabled to compress and decompress genome sequence data sets of any size in a loss-less fashion. The BIND implementation is also capable of handling non-ATGC as well as lowercase characters. This makes it suitable for practical application to real-world sequencing data sets.

2. Methods

2.1 *Compression steps*

For any given genomic sequence file (in fasta/multi-fasta format), the algorithm compresses the file using the following steps:

(a) BIND first separates all sequence headers and compresses them using 7-Zip, a freely available general purpose compression tool. In case of multi-fasta files, the length of individual sequences is also stored along with the respective headers.

(b) The position and type of non-ATGC characters and lowercase character stretches (if present in the sequence portion) are also stored in a separate file. These characters are subsequently removed and the resultant sequence file (now containing only uppercase A, T, G and C characters) is then processed in the subsequent steps.

(c) BIND uses individual binary codes (00, 01, 10 and 11) to represent the four nucleotides (A, T, G and C), and subsequently splits these codes into two data streams. In the first stream, while the characters A and T are encoded as '0', characters G and C are encoded as '1'. In the second stream, characters A and G are encoded as '0' and characters T and C are encoded as '1'.

(d) Each stream is then individually read in the form of alternating blocks of 1's and 0's, wherein each block represents a consecutive stretch of either 1 or 0. The lengths (ranging from 1 to n) of alternating blocks (in each stream) are computed.

(e) BIND then encodes the computed 'block-lengths' using a unary coding scheme. In this scheme, a number 'n' (in this case the length of a block) is represented by n bits, wherein the first (n−1) bits are represented either as (n−1) 'ones' or (n−1) 'zeros'. The nth bit is represented as 'one' if 'zeros' have been used for representing the first (n−1) bits, and vice versa.

(f) The resulting unary-coded block-length files (from the two data streams) are further compressed using the lzma algorithm. The lzma algorithm is freely available as an implementation within the 7-Zip compression package.

(g) Files generated in the preceding steps thus represent (i) compressed form of sequence headers, (ii) position/type information of non-ATGC characters and stretches of lowercase characters and (iii) unary coded 'block lengths' of the two data streams. These files are finally archived by BIND (with the help of 7-Zip archiver) to generate a single compressed output file. The final output file thus contains all information required for a loss-less reconstruction of the original input file.

2.2 *Decompression steps*

BIND decompresses a compressed genomic sequence file in the following manner:

(a) From the compressed file, data corresponding to headers, position/type information of non-ATGC characters as well as stretches of lowercase characters, and 'block-lengths' of the two data streams, are extracted.

(b) Block-length information (of both streams) is utilized to decipher the sequence (i.e. order) of four major nucleotides (A, T, G and C).

(c) Positional information of lowercase stretches and non-ATGC characters is then used for converting/inserting them at respective positions in the sequence file obtained in the previous step.

(d) Sequence length information appended within the headers file is then employed for splitting the sequence file obtained in step (c). Header information is finally inserted in the sequence file at appropriate position(s) for reconstructing the original (uncompressed) genomic sequence file.

2.3 *Validation procedure*

The compression efficiency of BIND, in terms of compression ratio and compression/decompression speeds, was evaluated using four real-world data sets. These data sets comprised files having variation with respect to size (kilobytes to gigabytes) and format (i.e. single-fasta and multi-fasta). While the first data set (FNA data set) comprised of 2679 files (each in single fasta format) corresponding to completely sequenced prokaryotic genomes, the second data set (FFN data set) had an equal number of files (in multi-fasta format) containing gene sequences corresponding to these prokaryotes. The third data set (Eukaryotic data set) contained files corresponding to 25 human chromosomes (chromosomes 1-22, Mitochondrial, X and Y). The fourth data set comprised of 14 files (ranging in size from a few MB to several GB) containing raw sequencing data generated using NGS platforms. Sequences in this data set were observed to contain several non-standard nucleotide character(s). Files of all four data sets were downloaded from prominent sequence repositories viz., NCBI, UCSC and CAMERA.

Results obtained with BIND were compared with those obtained using bzip2 (v.1.0.5), default gzip (v.1.4), gzip with −9 option (i.e. best compression mode) and lzma (v.9.20). A Linux desktop having a dual core 2.33 GHz processor with 2 GB RAM was used for all evaluation experiments. Results were analysed in terms of the following three parameters:

(1) PCR i.e. percentage compression ratio

$$PCR = (\textit{compressed file size}/\textit{original file size}) \times 100$$

(2) PICR i.e. percentage improvement in compression ratio (CR) of BIND as compared to CR of other compression algorithms

$$PICR = [1 - (\textit{CR of BIND}/\textit{CR of compared algorithm})] \times 100$$

(3) Time (in minutes) taken for compressing and decompressing all files in a given data set.

3. Results

Figure 1 illustrates a comparison of (a) compression ratios and (b) total processing time for compression and decompression of four validation data sets by BIND and the GPCAs. In this plot, the horizontal axis denotes the PCR and the vertical axis represents the total time taken for compression and decompression. An efficient compression algorithm should ideally attain a low value of PCR in the shortest possible time. Accordingly, the results with such an algorithm would be plotted closest to the origin. On the other hand, an algorithm inefficient in terms of PCR and/or in terms of processing time will have its results plotted farther from the origin. It is evident from figure 1 that for all types of

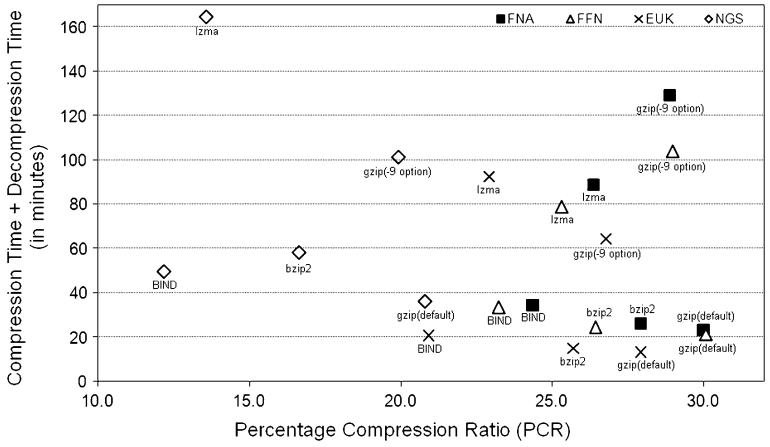

Figure 1. Scatter-plot illustrating a comparison of percentage compression ratios attained (on X-axis) and total processing time required (on Y-axis) by BIND and GPCAs. Different types of data sets are represented by different symbols.

data sets, the compression results of BIND lie closest to the origin as compared to those by other compression techniques. BIND, in all cases, attains the lowest PCR value. Although the results of lzma (in terms of PCR) also appear reasonably close to the vertical axis in some cases, lzma is observed to require a significantly higher processing time as compared to BIND. On the other hand, although the processing time of gzip (default) and bzip2 algorithms are very low, their PCR is significantly higher as compared to both lzma and BIND. It may be noted that the processing time of BIND for all data sets is considerably better than that of lzma and gzip (−9 option), and almost comparable to that of gzip (default) and bzip2. This suggests that the gains in data compression by BIND are not attained at the cost of processing time.

The gains in compression ratio obtained by BIND over the GPCAs are listed as respective PICR values in table 1. The average improvement in CR achieved by BIND over different compression techniques is observed to range between 8% and 27%. Interestingly, the results of BIND indicate a significant improvement over lzma (PICR in the range of 8–10%), an algorithm observed to provide the best CR among the tested GPCAs. The compression gains of BIND are also observed to be dependent on the type of the genomic data. For instance, in the case of NGS data set, the compression gains of BIND over gzip (default) and gzip (with −9 option) compression techniques are as high as 40%.

In summary, the performance of BIND in compressing genomic data files is significantly better than the popular compression techniques like bzip2, gzip and lzma. More importantly, the compression gains are achieved at a reasonable processing speed. Furthermore, BIND can run on systems with minimal hardware configuration, even for very large-sized data sets, thus making it viable for use in practical situations (as compared to implementations of existing specialized genome compression approaches). The detailed validation results of BIND and other compression algorithms (for individual files constituting the different data sets) are provided in supplementary tables 1–5.

Table 1. Percentage Improvement in Compression Ratio (PICR) achieved using BIND over other general purpose compression algorithms

	PICR[a] of BIND with respect to			
Validation data set	bzip2	gzip (default)	gzip (−9 option)	lzma
FNA data set	12.71	18.79	15.59	7.51
FFN data set	12.03	22.73	19.83	8.21
Eukaryotic data set	18.71	25.12	21.93	8.77
NGS data set	26.91	41.49	38.97	10.45
Average	17.59	27.03	24.08	8.74

[a] PICR = [1 − (CR of BIND / CR of compared algorithm)] × 100.

4. Discussion

The reasonably low cost of obtaining sequence data using the NGS technologies has led to the drastic increased rate of sequencing throughput. Such technologies have enabled life science research groups to experimentally study genomes/metagenomes by sequencing/re-sequencing them at increasingly high sequencing depth and coverage. Besides developing novel algorithms/methods for analysis, it is equally important to develop software/hardware solutions for efficiently storing and disseminating the vast amounts of

generated data. Several studies in the recent times have proposed specialized approaches which are able to achieve significant compression gains (with genome sequence data) as compared to general purpose compression approaches. However, it is observed that most (if not all) of these specialized compression approaches have huge memory/time requirements. Therefore, employing/testing them (even with moderate-sized sequence data sets) on a standard workstation becomes practically impossible. Due to this reason, it was not possible (in this study) to compare the compression gains of BIND with those obtained using such specialized genome compression approaches. Given the above observation, it does not seem surprising that none of the major sequence repositories (NCBI, CAMERA, DDBJ, EMBL, etc.) employ them for compressing sequence data set(s). Instead these repositories use GPCAs (mostly gzip) in spite of their low compression efficiency.

The reasonably efficient compression algorithm (BIND) presented in this article uses a unique encoding scheme for achieving compression gains significantly better than the widely used general purpose compression algorithms (GPCAs). After suitably (and separately) encoding headers and the positions of non-ATGC/lowercase character(s) stretches, BIND proceeds to encode each of the four major nucleotides (which constitute the bulk of the data) using a modified two-bit encoding scheme. In this scheme, the BIND implementation first splits the binary data into two independent streams and processes them in parallel. Given that most of the contemporary desktops/workstations support multi-threading and generally posses two or more processing cores, splitting and parallel processing of the data as two streams (as in BIND) is observed to significantly reduce the overall compression time. Furthermore, for such split binary streams, it is observed that the frequencies of block-lengths (i.e. lengths of consecutive stretches of 0 or 1) follow a geometric distribution. Based on this observation, BIND encodes block-lengths using a unary coding scheme (an optimal coding scheme for geometric distributions) and further compresses the same using lzma algorithm. This results in a significantly improved compression ratio.

Validation results of BIND, besides indicating enhanced compression efficiency over GPCAs, indicate an interesting pattern. The compression gains obtained using BIND on the NGS data sets are observed to be significantly higher as compared to that obtained for other three data sets. Given that NGS sequencing data constitutes the bulk of present day sequencing repositories, substituting GPCAs with BIND would logically result in significantly reducing the costs and time associated with storage and dissemination of sequencing data.

References

Behzadi B and Le Fessant F 2005 DNA compression challenge revisited; in *Combinatorial pattern matching: Proceedings of CPM-2005, LNCS, Jeju Island, Korea* (Springer-Verlag)

Cao MD, Dix TI, Allison L and Mears C 2007 A simple statistical algorithm for biological sequence compression; in *Proceedings of the Data Compression Conference, DCC-2007, Snowbird, Utah*

Chen X, Li M, Ma B and Tromp J 2002 DNACompress: fast and effective DNA sequence compression. *Bioinformatics* **18** 1696–1698

Cochrane G, Karsch-Mizrachi I and Nakamura Y 2011 International Nucleotide Sequence Database Collaboration. *Nucleic Acids Res*. **39** D15-8

Grumbach S and Tahi F 1994 A new challenge for compression algorithms: genetic sequences. *Inform. Process. Management* **30** 875–886

Korodi G and Tabus I 2007 Normalized maximum likelihood model of order-1 for the compression of DNA sequences; in *Proceedings of the Data Compression Conference, DCC-2007, Snowbird, Utah*

Manzini G and Rastero M 2004 A simple and fast DNA compressor. *Software—Practice and Experience* **34** 1397–1411

Matsumoto T, Sadakane K and Imai H 2000 Biological sequence compression algorithms; in *Genome informatics: Proceedings of the 11th Workshop, Tokyo, Japan,* (eds) AK Dunker, A Konagaya, S Miyano and T Takagi. pp 43–52

Metzker ML 2010 Sequencing technologies - the next generation. *Nat. Rev. Gene*t. **1** 31–46

Pinho AJ, Neves AJR and Ferreira PJSG 2008 Inverted-repeats-aware finite-context models for DNA coding; in *Proceedings of the 16th European Signal Processing Conference, EUSIPCO-2008, Lausanne, Switzerland*

Pinho AJ, Ferreira PJSG, Neves AJR and Bastos CAC 2011 On the representability of complete genomes by multiple competing finite-context (Markov) models. *PLoS ONE* **6** e21588

Rivals E, Delahaye JP, Dauchet M and Delgrange O 1996 A guaranteed compression scheme for repetitive DNA sequences; in *Proceedings of the Data Compression Conference, DCC-96*, *Snowbird, Utah*, pp 453

Zhang J, Chiodini R, Badr A and Zhang G 2011 The impact of next-generation sequencing on genomics. *J. Genet. Genomics* **38** 95–109

MS received 23 January 2012; accepted 14 May 2012

Corresponding editor: REINER A VEITIA