Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Manuscript Category

Subject Section

METHCOMP: A Special Purpose Compression Platform for DNA Methylation Data

Jianhao Peng, Olgica Milenkovic*, and Idoia Ochoa*

Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, 61801, U.S.

*To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: DNA methylation is one of the most important epigenetic mechanisms in cells that exhibits a significant role in controlling gene expressions. Abnormal methylation patterns have been associated with cancer, imprinting disorders, and repeat-instability diseases. As inexpensive bisulfite sequencing approaches have led to significant efforts in acquiring methylation data, problems of data storage and management have become increasingly important. The de facto compression method for methylation data is *gzip*, which is a general purpose compression algorithm that does not cater to the special format of methylation files. We propose METHCOMP, a new compression scheme tailor-made for bedMethyl files.
Results: We tested the METHCOMP algorithm on 24 bedMethyl files retrieved from four randomly selected ENCODE assays. Our findings reveal that METHCOMP offers an average compression ratio improvement over gzip of up to $8\times$. As an example, METHCOMP compresses a 48GB file to only 0.9GB, which corresponds to a 98% reduction in size. The compression times of METHCOMP and gzip are of comparable order amounting to 15MB/s.
Availability: The software METHCOMP is implemented in C++ and freely available at GitHub `https://github.com/jianhao2016/METHCOMP`.
Contact: milenkov@illinois.edu, idoia@illinois.edu
Supplementary information: None.

1 Introduction

DNA methylation is one of the most common mechanisms of epigenetic modification and a key element in controlling vertebrate gene function and cell differentiation [1]. Recent years have seen a surge in the number of projects focused on determining methylation abnormalities in carcinogenesis[2]. DNA methylation metrics were also found to be important in early detection of tumors and in determining the prognosis of the disease. In another direction, targeted DNA methylation has beed used to re-express erroneously silenced genes in cancer cells[3]. The recent survey [4] lists a number of other diseases currently known to be caused by improperly regulated DNA methylation and details the underlying aberration mechanisms.

Given is importance in fundamental biological and medical research, DNA methylation has been the subject of many large-scale projects including MethylomeDB [5] , DiseaseMeth [6] , NGSmethDB [7] and MethBase [8]. In particular, over 800 ENCODE project assays are DNA methylation-related (`https://www.encodeproject.org/matrix/?type=Experiment`), amounting to roughly 10% of the total assays; a total of 200 additional assays involve methylation state data.

Methylation data from these projects is almost exclusively generated from whole-genome shotgun bisulfite sequencing (WGBS)[16] coupled with a reduced representation bisulfite sequencing (RRBS) pipeline. The raw data is converted into BED format (`https://genome.ucsc.edu/FAQ/FAQformat.html#format1`), which in the ENCODE database is referred to as a bedMethyl file (`https://www.encodeproject.org/wgbs/#outputs`). The bedMethyl files keep all methylation states such as CHG, CpG and CHH in order to enable maximal information content, and are readable as plain text file. Consequently, each bedMethyl file contains at least hundreds of millions or even billions of lines, and is of average size of 15GB, and often larger than 40GBs.

As a result, a large volume of DNA methylation data files has to be stored and transferred online for analysis, learning and data mining purposes. For this purpose, until now, only traditional gzip software has been used to reduce the footprint of methylation data. Unfortunately, gzip compressors are universal methods designed to operate on diverse types of redundancy, and are hence not specialized for repetitive patterns encountered in bedMethyl files. This leads to significantly compromised compression performance. To address this problem, we developed a new, specialized compression method for bedMethyl files, termed METHCOMP. METHCOMP relies on a carefully integrated processing structure which encompasses deinterleaving different columns of bedMethyl files to optimize the corresponding differential and runlength coding schemes and it includes a highly efficient collection of arithmetic encoders. As a result, METHCOMP offers almost an order of magnitude improvement in the compression ratio of bedMethyl files when compared to gzip, and roughly compacts the files to not more than 2% of their original size.

2 Methods

We seek to compress DNA methylation data stored in the bedMethyl format. The bedMethyl format is a “modified” BED format consisting of 11 columns, with the first nine following the same format as BED files, and the last two columns being methylation specific. In particular, each column (c) contains the following information:

- c1: **chrom**, nreference chromosome or scaffold
- c2: **chromStart**, the start position in chromosome
- c3: **chromEnd**, the end position in chromosome
- c4: **name**, the name of item
- c5: **score**, the score of item
- c6: **strand**, the strand of item
- c7: **thickStart**, the start position where the feature is drawn thickly
- c8: **thickEnd**, the end of position of thick displayed feature
- c9: **itemRgb**, the color mapping value of item
- c10: **coverage**, the number of reads
- c11: **percentage**, the percentage of methylated reads

The columns are a combination of strings and integers, and each of them encodes different information. Thus, we use different compression strategies for different columns. The columns containing string values include *chrom* (c1), *name* (c4), and *strand* (c6). The columns in the bedMethyl format are sorted by reference chromosome (c1), and within a given chromosome, by the starting position (c2). In addition, a given reference chromosome may contain millions of items. With these assumptions, it is reasonable to use run-lengh coding for column 1. The value of column 4 (*name*) is the same for consecutive records, and thus for the same reasons as for column 1, we use run-length encoding. The *strand* column indicates if methylation occurs on the sense or antisense strand (it can take values in $\{+, -, .\}$, where “.” corresponds to “unknown origin”). The strand value is most likely independent of the value of any other columns, and thus we use an adaptive arithmetic coding with a dictionary of size 3 to compress this column. Some of the remaining columns, which contain integers, are highly correlated. Hence, we can discard the values in column 3 *chromEnd* as these values are always equal to the values of column 2 *chromStart* plus one. In addition, in the WGBS pipeline, which is the one we use, columns 7 (*thickStart*) and 8 (*thickEnd*) are identical to *chromStart* and *chromEnd*, respectively, and thus they can also be discarded. Columns 5 (*score*) and 10 (*coverage*) are also highly correlated, with the score being equal to the min value between the coverage and 1000. Thus we encode only the *coverage* column. Finally, column 9 *itemRgb* is a deterministic color mapping that depends on the *percentage* value (c11), and hence can be easily retrieved from the column *percentage*. In summary, from the integer columns, we only need to encode columns 2 (*chromStart*), 10 (*coverage*), and 11 (*percentage*).

The values of column 2 (*chromStart*) are sorted within each reference chromosome (c1), and we first apply differential coding as a preprocessing step to potentially reduce the alphabet size and make the distribution skewed towards smaller values. The resulting values are then encoded by means of adaptive arithmetic encoding. The values of column 10 (*coverage*) are also encoded by means of adaptive arithmetic encoding. Finally, column 11 (*percentage*) takes values between 0 and 100, and thus we encode it with an adaptive arithmetic encoder of alphabet size 101.

The strategies employed by METHCOMP to compress the different columns that constitute the bedMethyl file are summarized below (all other columns are discarded during compression):

- c1: **chrom**, run-length
- c2: **chromStart**, differential encoding followed by adaptive arithmetic
- c4: **name**, run-length
- c6: **strand**, adaptive arithmetic
- c10: **coverage**, adaptive arithmetic
- c11: **percentage**, adaptive arithmetic

2.1 Implementation Details

We implemented the METHCOMP encoder and decoder in C++, and the software may be retrieved from `https://github.com/jianhao2016/METHCOMP`. Instructions on how to run the code are available at the same site. Notice that in order to compile the source, the standard version C++11 [9] of C++ is required. This version has a number of versatile features amenable for use in large scale data compression.

2.1.1 The Encoder

The encoder reads the bedMethyl file line by line, and then parses each line into a Row object that keeps the raw data retrieved from each line in their default type. For example, *chrom* (c1), *name* (c4) and *strand* (c6) are stored as strings, while all the remaining seven columns except for *itemRgb* (c9) are stored as integers; *itemRgb* (c9) is maintained as a string since it is discarded during subsequent processing and given that there are only 11 different colors in the color map, a saved string is easier to retrieve by an enumerate class than individual numbers. Each Row is sliced into an InputData class, which performs data preprocessing such as mapping and differentiation. The InputData of each row only contains the necessary columns described earlier in this section. The two run-length codes operating on *chrom* and *name* are updated inside the InputData class while lines are being fed to the encoder.

Each InputData is inherited from an ArithmeticInt class, so every column in the integer part of InputData is sent to a separate adaptive arithmetic encoder. The values in *strand* (c6) and *percentage* (c11) are fixed to 3 and 101, respectively. As a result, the arithmetic encoder can use fixed dictionaries of respective size 3 and 101 on them. The values in *chromStart* (c2) and *coverage* (c10) have an unknown range which depends on the results of the experiments themselves. Keeping a dictionary of size 2^{32} to cover all the possible integer values is too expensive and impractical. One solution is to update the dictionary while encoding so as to keep the size of the dictionary at a minimum, but this approach requires keeping additional information that indicates the update and frequent extensions of the dictionary, which consumes significant computational resources. Given that the file size is already very large, a simpler method is preferred. This is why we opted for “slicing” the 32-bit integer into 4 different 8-bit integers and using a fixed dictionary of size 256 for each of them. Since the higher register bits do not change as often as the lower register ones, the frequency counts used in the corresponding arithmetic encoders will concentrate around several small values. This enables adding only a few bits to each integer while keeping the model as simple as possible.

The encoding procedure is depicted in Fig. 1. The input is a bedMethyl file, and after compression, we obtain three files. Two of them are obtained from run-length coding, namely “outfileChrom” and “outfileName”, and are saved as plain text since they are both of size less than 100 KB (for the files tested). The third file is the output bit stream from the arithmetic encoders. The encoder consecutively goes over the Row of these encoders and outputs them into the same bit stream.

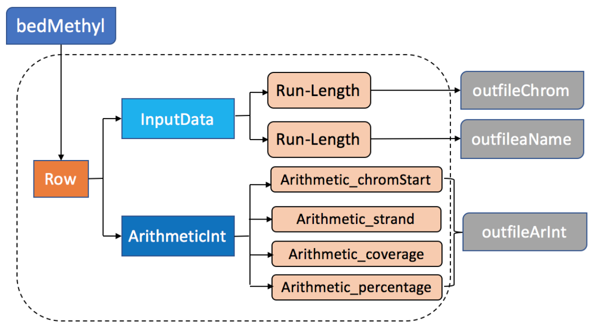

Fig. 1: Schematic of the METHCOMP encoder.

2.1.2 The Decoder

From the data flow perspective, the decoder performs mirror image processing with respect to the encoder: the decoder takes three input files and feeds them to different decoder subunits to generate the components needed for reassembling the final output. The first step is arithmetic decoding of the files bit-by-bit and recovering ArithmeticInt instances. Afterwards, lines from the run-length coded files are used to form the InputData instance. At the third stage, the decoder combines these two instance into a Row instance and writes a row into the final output file.

The decoder terminates once the following three conditions are fulfilled: an EOF symbol is decoded from the arithmetic decoder; the last line in outfileChrom is reached; the last line in outfileName reached.

In the ideal case, those three conditions will be achieved simultaneously, since the encoders also terminate at the same time. If any of those conditions are met before the others, then the decoder will produce an “Error message” informing the user that the input files do not match each other.

3 Experimental results

To test the performance of our compression method, we ran the METHCOMP compression and decompression procedure on four randomly selected assays of the ENCODE project, and all the bedMethyl files within these assays. The codes of the corresponding ENCODE WGBS assays are *ENCSR835OJU* (Mus musculus C57BL/6 heart embryo Biosample), *ENCSR888JFA* (Mus musculus C57BL/6 forebrain embryo), *ENCSR351IPU* (Homo sapiens hepatocyte originated from H9), and *ENCSR656TQD* (Homo sapiens mammary epithelial cell female and female adult). The selected assays contain 6 bedMethyl files each, accounting for a total of 24 bedMethyl files. The bedMethyl files belonging to each assay may be retrieved from: `http://www.encodeproject.org/experiments/` prepended to the project code (e.g., ENCSR835OJU). Due to space constraints, we provide individual results for bedMethyl files belonging to the first assay *ENCSR835OJU* only, and combined results for all files belonging to each of the remaining assays. The original sizes of the tested bedMethyl files are listed in Table 1: they vary in value from 2.6 GBs to 48 GBs.

We tested the performance of our compression method in terms of the compression ratio, defined in Table 1 and with respect to execution time(Table 2). For comparison, in the same tables we listed the same performance results for gzip (Apple version 272). All experiments were performed on an Intel i7 laptop with 16 GB of RAM and 500 GB of disk memory. METHCOMP only uses one thread and occupies less than 2 GB of memory, so it may be run on any other 64 bit machine without anticipated problems.

We used an integer implementation of arithmetic coding, which instead of using a floating point representation for the interval $[0, 1)$ employs a mapping onto n bits, and hence reduces the precision. Therefore, one needs to iteratively output bits and rescale the interval so to make the calculated values fit into the allocated n-bit register. In order to avoid multiplication overflow problems, the number of bits should not be too large, and we settled on a 32-bit representation which is compatible with a 64-bit machine. For a 32-bit machine, this value may have to be decreased in case that arithmetic errors occur.

Table 1. Comparison of compression performance of gzip and METHCOMP. The compression ratio is computed as (original size/compressed size) and the space saving is calculated according to $(1 - \text{compressed size}/\text{original size})$. Individual file names correspond to assay ENCSR835OJU. ENCSR1, ENCSR2 and ENCSR3 stand for the combined results for assay ENCSR888JFA, ENCSR351IPU and ENCSR656TQD respectively.

file name	original size(GB)	compressed size(GB)		compression raito		impro- vement
		gzip	METHCOMP	gzip	METHCOMP	
167OJH	13	2.30	**0.286**	5.65	**45.43**	8.04
327MVH	48	7.30	**0.924**	6.58	**51.96**	7.90
428AXW	2.6	0.47	**0.085**	5.50	**30.60**	5.56
677YTO	13	2.30	**0.287**	5.65	**51.74**	8.01
751DLO	2.6	0.47	**0.085**	5.50	**31.32**	5.56
945JPE	48	7.3	**0.928**	6.58	**45.28**	7.87
ENCSR1	128.2	20.14	**2.600**	6.37	**49.32**	7.75
ENCSR2	139	21.82	**3.068**	6.37	**45.30**	7.11
ENCSR3	138.9	21.93	**3.064**	6.33	**45.33**	7.16
average				6.00	**42.27**	7.05

From the results in Tables 1 and 2, we see that METHCOMP provides storage savings exceeding **97**% for all the files tested. With respect to compression ratio improvements, METHCOMP offers roughly **8**× better results than gzip, with the corresponding numerical values for file 167OJH (which shows the most improvement) being 45.43 and 5.65, respectively. With respect to compression/decompression speed, METHCOMP introduces larger decompression delays compared to gzip due to parsing and reassembling of the file line by line. The compression time of METHCOMP is roughly twice that of gzip, while the decompression time is more than an order of magnitude higher than that of gzip. The average compression and decompression speeds of METHCOMP and gzip are 11.51 and 23.25, and 16.06 and 361.37 MB/s, respectively. It is worth pointing out that compression requires significantly more time than decompression, so that the overall time complexities of the schemes are only a factor of two apart. Furthermore, given that the decompression time is higher for METHCOMP than gzip, amounting to roughly 50 min for a file of size 50 GB, METHCOMP may be best suited for archival storage as it also offers immense file size reductions. The decompression speed of METHCOMP is still roughly **5**× faster than the file download time.

To ensure a completely fair comparison between METHCOMP and gzip, we also ran gzip on what we refer to as the “extracted” files. Extracted files contain only those columns effectively used by METHCOMP, as outlined at the introduction of Section 2. The results for some subset of the tested files are shown in Table 3. Even when one operates with extracted files, METHCOMP still offers **3**× better compression rates than gzip. At the same time, the compression/decompression times become comparable, as the largest fraction of the processing time is spent on extracting repetitive columns and reinserting them during decompression.

Table 2. Comparison of compression speeds of gzip and METHCOMP. The speed is computed according to (original size/time taken by the task)

file name	original size(GB)	compression time(s)		compression speed(MB/s)		decompression time(s)		decompression speed(MB/s)	
		gzip	METHCOMP	gzip	METHCOMP	gzip	METHCOMP	gzip	METHCOMP
167OJH	13	567	1201	**23.48**	11.08	36	824	**369.78**	16.16
327MVH	48	2058	4277	**23.88**	11.49	126	3007	**390.10**	16.35
428AXW	2.6	122	243	**21.82**	10.96	8	176	**332.80**	15.13
677YTO	13	572	1200	**23.27**	11.09	38	825	**350.32**	16.14
751DLO	2.6	128	241	**20.80**	11.05	7	176	**380.34**	15.13
945JPE	48	2060	4405	**23.86**	11.16	127	2870	**387.02**	17.13
ENCSR1	128.2	5752	10921	**22.11**	12.02	379	7662	**331.71**	17.13
ENCSR2	139	6070	12407	**23.67**	11.47	381	5353	**368.10**	17.24
ENCSR3	138.9	6257	12119	**23.02**	11.74	388	8419	**356.51**	16.89
average				**22.91**	11.61			**356.18**	16.06

Table 3. Performance of gzip when applied to the subset of columns effectively used by METHCOMP.

file name	original size(GB)	extracted size(GB)	compressed size(GB)	space savings	compression ratio	compression time(s)	compression speed(MB/s)	decompression time(s)	decompression speed(MB/s)
365XZL	2.6	1.0	0.24	91.89%	12.33	219	12.16	149	17.87
506SUF	13	5.1	0.88	93.19%	14.69	1126	11.82	784	16.98
170PBE	15	5.5	1.00	93.33%	15.00	1259	12.20	843	18.22
487XOB	3.5	1.3	0.27	92.24%	12.89	292	12.27	199	18.01
average				92.66%	13.73		12.11		17.77

4 Discussion

The prevalent trend in compression practice in the field of genomics is to apply universal methods that can compress arbitrary file formats to an acceptable level. Unfortunately, with the surge of Big Data platforms in biological and medical research, it has become imperative to devise significantly more space efficient, specialized algorithms for the underlying data. In this direction, new software suites for FASTQ and VCF files have been developed for genomic data storage [10; 11; 12; 17; 18], along with a number of specialized methods for compressing metagenomic data, RNA-seq and ChIP-seq measurements in lossless and lossy modes [13; 14; 15]. METHCOMP is one more addition to the growing library of high-performance compression suites for *-omics* data that is expected to play a significant role in future cancer genomics and personal medicine research. Given that the inherent characteristics of methylation data are unchangeable (e.g., chromosome name, start and end point of the identified methylation cite, methylation statistics etc), shifts in methylation data acquisition technologies will not impact the utility of the software. The column-by-column compression approach also allows to flexibly incorporate new information columns, or discard unnecessary ones. Furthermore, since the bedMethyl format is a special form of a BED format, it may be easily extended to operate on other BED formats.

5 Conclusion

DNA methylation data carries valuable information for early tumor detection and cancer diagnostics, and has been curated by a number of large scale genomics projects. To address the storage issues associated with growing volumes of bedMethyl files, we developed a new special purpose compression method termed METHCOMP. METHCOMP combines run-length coding, differential coding and adaptive arithmetic coding and outperforms gzip in terms of compression ratio close to an order of magnitude.

Acknowledgements

The authors are grateful to Minji Kim and Mikel Hernaez for many fruitful discussions during the software implementation stage.

Funding

This work was supported in part by an NIH BD2K grant for Targeted Software Development.

References

[1]Razin, A., Riggs, A. D. (1980). DNA methylation and gene function, *Science*, **210(4470)**, 604-610.

[2]Das, P. M., Singal, R. (2004). DNA methylation and cancer. *Journal of clinical oncology*, **22(22)**, 4632-4642.

[3]Baylin, S. B. (2005). DNA methylation and gene silencing in cancer. *Nature clinical practice Oncology*, **2**, S4-S11.

[4]Robertson, K. D. (2005). DNA methylation and human disease. *Nature reviews. Genetics*, **6(8)**, 597.

[5]Galperin, M. Y., FernÃ¡ndez-SuÃ¡rez, X. M. (2011). The 2012 nucleic acids research database issue and the online molecular biology database collection. *Nucleic acids research*, **40(D1)**, D1-D8.

[6]Lv, J., Liu, H., Su, J., Wu, X., Liu, H., Li, B., Zhang, Y. (2011). DiseaseMeth: a human disease methylation database. *Nucleic acids research*, **40(D1)**, D1030-D1035.

[7]Hackenberg, M., Barturen, G., Oliver, J. L. (2010). NGSmethDB: a database for next-generation sequencing single-cytosine-resolution DNA methylation data. *Nucleic acids research*, **39(suppl 1)**, D75-D79.

[8]Song, Q., Decato, B., Hong, E. E., Zhou, M., Fang, F., Qu, J., Smith, A. D. (2013). A reference methylome database and analysis pipeline to facilitate integrative and comparative epigenomics. *PloS one*, **8(12)**, e81148.

[9]C++ Standards Committee., (2011). ISO/IEC 14882:2011, Standard for Programming Language C++. *Technical report, 2011*. **http://www.open-std.org/jtc1/sc22/wg21**.

[10]Malysa, G., Hernaez, M., Ochoa, I., Rao, M., Ganesan, K., Weissman, T. (2015). QVZ: lossy compression of quality values. *Bioinformatics*, **31(19)**, 3122-3129.

[11]Long, R., Hernaez, M., Ochoa, I., Weissman, T. (2017, April). GeneComp, a new reference-based compressor for SAM files. *Data Compression Conference (DCC)*, **(pp. 330-339)**. IEEE.

[12]Tatwawadi, K., Hernaez, M., Ochoa, I., Weissman, T. (2016). GTRAC: fast retrieval from compressed collections of genomic variants. *Bioinformatics*, **32(17)**, i479-i486.

[13]Ravanmehr, V., Kim, M., Wang, Z., Milenkovic, O. (2017). ChIPWig: A Random Access-Enabling Lossless And Lossy Compression Method For ChIP-Seq Data. *bioRxiv*, 127464.

[14]Kim, M., Zhang, X., Ligo, J. G., Farnoud, F., Veeravalli, V. V., Milenkovic, O. (2016). MetaCRAM: an integrated pipeline for metagenomic taxonomy identification and compression. *BMC bioinformatics*, **17(1)**, 94.

[15]Wang, Z., Weissman, T., Milenkovic, O. (2015). smallWig: parallel compression of RNA-seq WIG files. *Bioinformatics*, **32(2)**, 173-180.

[16]Yang, A. S., EstÃ©cio, M. R., Doshi, K., Kondo, Y., Tajara, E. H., Issa, J. P. J. (2004). A simple method for estimating global DNA methylation using bisulfite PCR of repetitive DNA elements. *Nucleic acids research*, **32(3)**, e38-e38.

[17]Deorowicz, S., Danek, A., Grabowski, S. (2013). Genome compression: a novel approach for large collections. *Bioinformatics*, **29(20)**, 2572-2578.

[18]Roguski, Ł., Deorowicz, S. (2014). DSRC 2âŁ"Industry-oriented compression of FASTQ files. *Bioinformatics*, **30(15)**, 2213-2215.