Efficient compression of genomic sequences

Diogo Pratas, Armando J. Pinho, and Paulo J. S. G. Ferreira

IEETA - Institute of Electronics and Informatics Engineering of Aveiro
DETI - Department of Electronics, Telecommunications and Informatics
University of Aveiro, 3810-193 Aveiro, Portugal
{pratas,ap,pjf}@ua.pt

Abstract

The number of genomic sequences is growing substantially. Besides discarding part of the data, the only efficient possibility for coping with this trend is data compression. We present an efficient compressor for genomic sequences, allowing both reference-free and referential compression. This compressor uses a mixture of context models of several orders, according to two model classes: reference and target. A new type of context model, which is capable of tolerating substitution errors, is introduced. For ensuring flexibility regarding hardware specifications, the compressor uses cache-hashes in high order models. The results show additional compression gains over several specific top tools in different levels of redundancy. The implementation is available at http://bioinformatics.ua.pt/software/geco/.

Introduction

Genomic (DNA) sequences are large codified messages, from an alphabet of four symbols $\Theta = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$, describing most of the structure of all known living organisms. A huge amount of genomic data has been generated, with diverse characteristics, rendering the data deluge phenomenon a serious problem in most genomics centers. As such, most of the data are discarded (when possible), while other are compressed using general purpose algorithms, often attaining modest data reduction results.

Several specific algorithms have been proposed for the compression of genomic sequences, such as [1–7], but only some of them have been made available as usable and reliable compression tools, and those have been developed to some specific purpose or data characteristic [8–11].

The dramatic increase of sequenced genomes, given the reduced sequencing costs, and the high redundancy characteristics, led to the development of genomic reference sequence compression. Several compressors of this type have been proposed [12–16], although most of them seem to be less efficient in handling sequences with higher rates of mutation.

In this paper, we describe a statistical compressor that uses arithmetic coding [17], able to function in reference or reference-free mode. It is very flexible and can cope with diverse hardware specifications, due to the use of cache-hashes. It relies on a mixture of finite-context models (FCMs) and extended finite-context models (XFCMs). The XFCMs are a new type of models that we present and use in the compressor, showing promising results. Overall, the compressor shows some improvements over other state-of-the-art compressors.

Method

Extended FCMs

Consider an information source that generates symbols from Θ and that it has already generated the sequence of n symbols $x^n = x_1 x_2 \ldots x_n,\ x_i \in \Theta$ (we currently ignore other symbols). A subsequence of x^n, from position i to j, is denoted as x_i^j.

Finite-Context Models (FCMs) are statistical models assuming the Markov property, that have been used in many applications of data compression, namely associated to genomic sequences. A FCM of an information source assigns probability estimates to the symbols of the alphabet, according to a conditioning context computed over a finite and fixed number, k, of past outcomes (order-k FCM) [18]. At time n, these conditioning outcomes are represented by $x_{n-k+1}^n = x_{n-k+1}, \ldots, x_{n-1}, x_n$. The number of conditioning states of the model is $|\Theta|^k$ (in our case, 4^k).

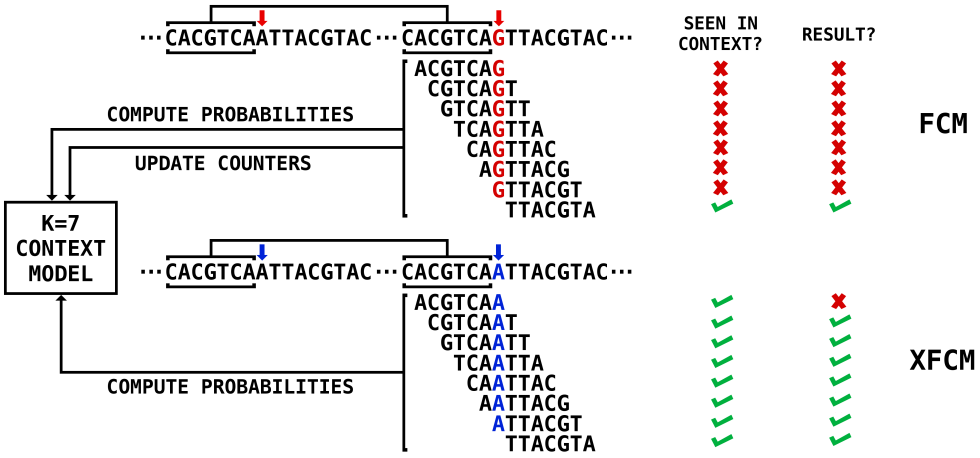

Figure 1: Performance of FCM versus XFCM, for $k = 7$, when a substitution (mutation) occurs. The memory (model) of both is shared. In the example, the XFCM computes the probabilities assuming as context the most probable symbols (according to the model and to a maximum number of allowed corrections). Larger k reveal larger differences between the models.

An extended FCM (XFCM) shares the memory with a FCM with the same context order, but it may assign a different probability estimate, because it may assume a different conditioning context. In the case of a XFCM, the conditioning context considers that s has always been the most probable symbol, and hence the estimator

$$P(s|x'^n_{n-k+1}) = \frac{N(s|x'^n_{n-k+1}) + \alpha}{N(x'^n_{n-k+1}) + \alpha|\Theta|} \tag{1}$$

is used, where x' is a copy of x, edited according to

$$x'_{n+1} = \underset{\forall s \in \Theta}{\operatorname{argmax}}\, P(s|x'^n_{n-k+1}). \tag{2}$$

Figure 1 depicts a comparison between a FCM and a XFCM, for $k = 7$. Notice that this strategy enables to modify the context that is considered to be seen, without

modifying and increasing the size of the model memory. Since these models make sense only in low complexity regions, we have created a way to turn them on or off, saving some time in the computation.

We permit t substitutions in the conditioning context of a XFCM without discarding it and, hence, turning it off. For example, consider that $k = 7$ and $c_0 =$ CACGTCA is the current context. Also, consider that the number of past symbol occurrences following c_0 was $\mathrm{A} = 1, \mathrm{C} = 0, \mathrm{G} = 0, \mathrm{T} = 0$. If the symbol that is being compressed is G (contradicting the probabilistic model), a FCM would have as next context $c_1 =$ ACGTCAG. However, the XFCM would use a c'_1 taking into account the most probable outcome and, hence, $c'_1 =$ ACGTCAA. Therefore, the next probabilistic model would be dependent on the past context assumed to be seen and, hence, it assumes that the symbol that was compressed is A.

The XFCM works well when using high orders. This also creates sparse occurrences that can be efficiently supported by a cache-hash memory and, therefore, we implemented the XFCMs using cache-hashes (see next subsection).

The XM algorithm [2] uses a copy-model that gives top compression genomic results, although at the expense of huge amounts of memory and time. Our intention with the XFCM is to approximate the copy-model using low and controllable memory, being as fast as possible.

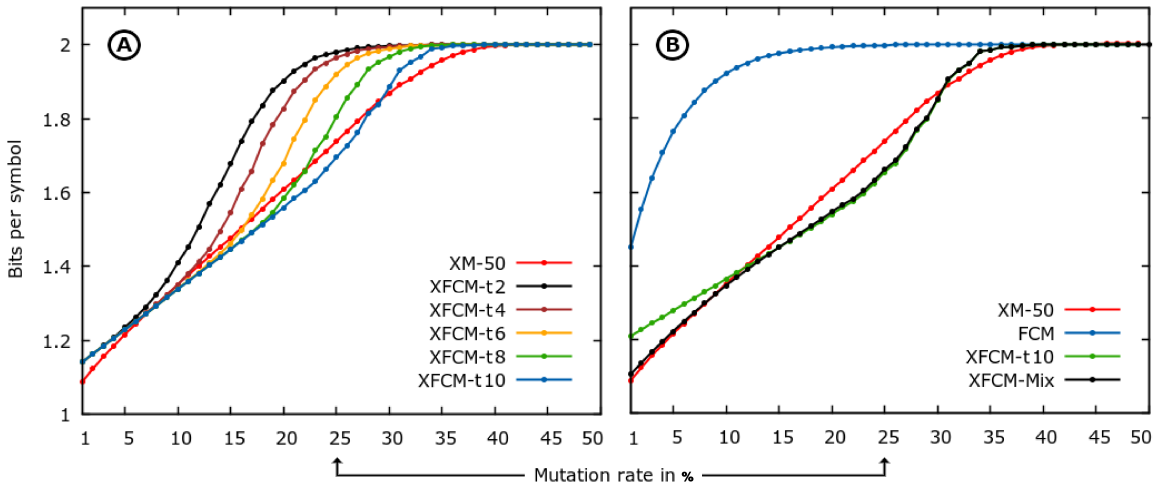

Figure 2: XFCM performance, varying the substitution threshold, in mutated data (target) and using the original as reference. The "t" represents the t permitted substitutions without discarding the context. The "XM-50" stands for the XM compression model using 50 experts. The "FCM-Mix" uses a XFCM mixed with a FCM (the same as having $t = 0$) of the same context order. **A)** shows a comparison of XFCM, for several t, against XM. **B)** shows how a mixture between XFCM and FCM outperforms, on average, XM.

For simplicity, we have generated a uniform pseudo-random synthetic sequence (using XS [19]) and have mutated the sequence with a defined substitution rate. Our intention is to simulate genomic sequences given several degrees of mutations (also removing the self-redundancy of the sequence) and compress it using the original as reference. In Fig. 2 we show the performance of the XFCMs, compared with the XM

model, using several values for parameter t. It can be seen that for a high value of t the model adjusts better to the nature of the data.

In Fig. 2 it is possible to see that a FCM is not able to deal with mutations as XFCMs and XM does. This was a disadvantage in our past models. When we mix FCM and XFCM ("FCM-Mix"), we are able to get results even better that XM. Moreover, we do not have two memory models, since they share the same cache-hash (because they have the same context order). In fact, for the same statistics, we have different predictors that cooperate according to a mixture.

Mixture of classes

The mixture is based on two model classes, those belonging to what we call the reference set, $\mathcal{R}$, and those in the target set, $\mathcal{T}$. The reference set contains the FCMs or XFCMs responsible for modeling the conditioning string, i.e., the y of $C(x|y)$, whereas the target set of FCMs or XFCMs is used to represent x, when required. The $C(x|y)$ represents the number of bits when compressing sequence x given y and, thus, it can be seen as conditional compression.

Basically, the probability of the next symbol, x_{n+1}, is given by

$$P(x_{n+1}) = \sum_{k \in \mathcal{R}} P_r(x_{n+1}|x_{n-k+1}^n)\ w_{k,n}^r + \sum_{k \in \mathcal{T}} P_t(x_{n+1}|x_{n-k+1}^n)\ w_{k,n}^t, \tag{3}$$

where $P_r(x_{n+1}|x_{n-k+1}^n)$ and $P_t(x_{n+1}|x_{n-k+1}^n)$ are, respectively, the probability assigned to the next symbol by a model from the reference set and from the target set, and where $w_{k,n}^r$ and $w_{k,n}^t$ denote the corresponding weighting factors, with

$$w_{k,n}^r \propto (w_{k,n-1}^r)^\gamma P_r(x_n|x_{n-k}^{n-1}) \quad \text{and} \quad w_{k,n}^t \propto (w_{k,n-1}^t)^\gamma P_t(x_n|x_{n-k}^{n-1}) \tag{4}$$

where $\gamma \in [0, 1)$ acts as a forgetting factor. Moreover, both are constrained to

$$\sum_{k \in \mathcal{R}} w_{k,n}^r + \sum_{k \in \mathcal{T}} w_{k,n}^t = 1. \tag{5}$$

To compute $C(x)$ only the models of the $\mathcal{T}$ class are used, while to compute $C(x|y)$ the compression is performed in two phases. In the first phase, the $\mathcal{R}$ class of models accumulates the counts regarding the y sequence. After the entire y sequence was processed, the models are kept frozen and, hence, the second phase starts. At this point, the x sequence starts to be compressed using the $\mathcal{R}$ models computed during the first phase, in cooperation with the set of models of the $\mathcal{T}$ class, that dynamically accumulate the counts only from x.

Cache-hash

In order to ensure a flexible compressor in the sense of memory optimization for any computer hardware specification, we have developed the cache-hash. The cache-hash approach keeps only the last hashed entries in memory, rendering a flexible and predictable quantification of the memory necessary to run in any sequence (memory does not blow with the size of the sequence).

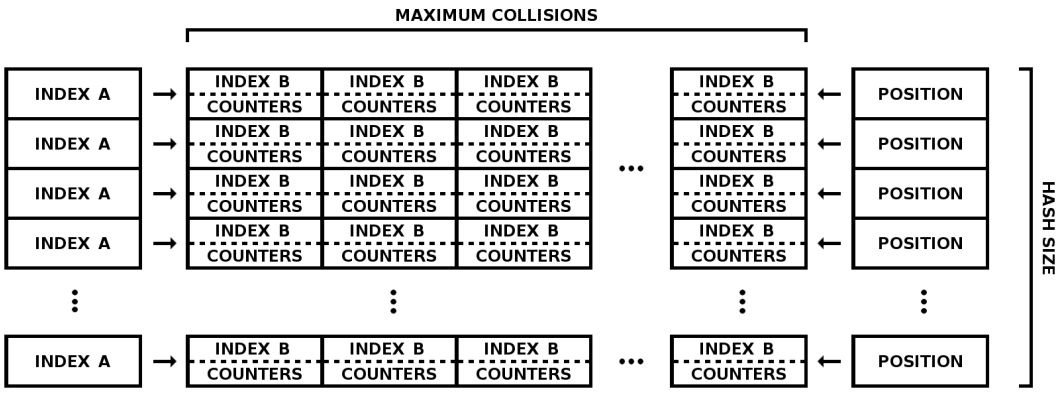

Figure 3: Cache-hash scheme. The "POSITION" indicates the position of the last edited block relatively to the hash index. Each hash index is shaped by both "INDEX A" and "INDEX B". The "COUNTERS" store the counts for each base. These are packed using a 2 bits per base approach that on overflow normalize all by half.

Depicted in Fig. 3, the cache-hash is able to store each index primarily by exploring precision splitting. For example, if the context order size can be up to 20 and the hash table has size 24 bits (specifically, the next prime after 2^{24}), each hashed entry uses 24 bits for the "INDEX A" and 16 bits for the "INDEX B". This entry will be added or updated (only if already exists) according to the "POSITION" (circular buffer having the "MAXIMUM COLLISIONS" as size). An advantage is that we only need to store "INDEX B", since the full index can be disambiguated by the position where it is inserted or updated. For each block according to each "POSITION", we have the "INDEX B" (already described) and the "COUNTERS". The latter stores the counters with 2 bits precision in a unsigned char (4 symbols and 2 bits per symbol gives the 8 bits). Each time a counter reaches the value 3, all the counters (for the 4 symbols) are normalized, hence divided by two (only integer division). This would be problematic if the size of the context was small. A context of 20 is considered large, resulting in a sparse table, if represented in that sense. As such, the cache-hash in fact simulates a structure that can be seen as a middle point between a probabilistic and dictionary model. The cache-hash uses a fixed hash function that, on average, has a low probability of collisions.

For searching in the cache-hash we need to compare each key, formed by "INDEX A" and "INDEX B", with the actual key, given by the context. This is a costly task, although needed. To minimize the processing time, we start the search from the previous position relatively to the current position, given by "POSITION", and search from newest to oldest entries. This approach relies on the characteristics of the genomic sequences, for which similar regions tend to be grouped or near.

Results

The tool (GeCo), written in C language, is available at `http://bioinformatics.ua.pt/software/geco`, under GPL-2, and can be applied to any genomic sequence. The experiments have been performed on a Linux server running Ubuntu with 16 Intel(R)

Xeon(R) CPU E7320 at 2.13 GHz and with 256 GB of RAM. We have used seven datasets. Three datasets were used from the recent sequenced birds project [1], namely duck, peregrine falcon and cuckoo. All of the birds datasets were not assembled (in contig state), totaling more than 3.2 GB multi-FASTA derived format. The fourth dataset was the assembled human genome GRC-b38 [2] (with 24 chromosomes) totaling an approximated size of 2.9 GB. The fifth dataset contains a FASTQ file used in [10] [3], while the sixth is a frequently used one [4]. For fair comparison, the datasets have been filtered and transformed into equivalent files in order to compare only genomic sequences, using the Goose framework [5]. The final dataset, for reference compression, has been downloaded from the NCBI, including the assembled genomic sequences of human, chimpanzee, gorilla and orangutan (chromosomes 5,11 and 18).

As it can be seen in Table 1, Gzip was unable to compress the human genome below 2 bits per base. On the other hand, the specific methods were able to compress it with success. DNACompact [7] used a very small amount of memory, but at the expense of more computational time. The proposed algorithm (GeCo) provides a substantial compression improvement to our previous algorithm (DNAEnc3 [4]), using much less resources. In fact, GeCo is able to compress the human genome in less than 550 MB, using memory equivalent to a laptop computer, and much faster than the previous approaches. Moreover, memory will not explode with the size of the sequence, unlike DNAEnc3 and XM [2]. XM was unable to compress two of the birds datasets, due to a processing error.

Table 2 depicts the benchmarks of two datasets using several tools, namely two state-of-the-art FASTA dedicated tools: Deliminate [8] and MFCompress [9]. Moreover, two state-of-the-art FASTQ tools are also used: fqz_comp [10] and Orcom [11]. As it can be seen, GeCo provides substantial compression capabilities, although at the expense of more computational time. The tool Orcom, an efficient disk-based tool, was not able to address with much success these files, perhaps because it is more suitable for much larger files.

Table 3 includes several reference-based compressed tools, namely GReEn [14], iDoComp [16] and GeCo (proposed). We have also ran GRS [12], however, as in [15], the sequences have some degree of dissimilarity and therefore the programs are not suitable for this purpose (GRS even suggests not to be used in these cases). Moreover, GDC versions [20, 21] are more suitable for large collections, since in most of the cases they reported a compression value above two bits per base. For more similar sequences, iDoComp seems to attain the best compression results. In more dissimilar sequences, we can see that, on average, GeCo outperforms the specific reference compressors at the expense of some more time and memory. In fact, we have only used the first reference mode (-l 11) and, therefore, the compression factor might increase although at the expense of more space/time resources.

[1] ftp://climb.genomics.cn/pub/10.5524/

[2] ftp://ftp.ncbi.nlm.nih.gov/genomes/H_sapiens/Assembled_chromosomes/seq/

[3] ftp://ftp.ddbj.nig.ac.jp/ddbj_database/dra/fastq/SRA001/SRA001546/SRX000706/

[4] ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR494/SRR494099/SRR494099.fastq.gz

[5] https://github.com/pratas/goose/

Table 1: Compression benchmarks for state-of-the-art pure genomic compression tools. Time is in minutes, while maximum memory peak is in MBytes. With the exception of Gzip, the compressors are symmetric (time/memory compression and decompression are approximately the same). Symbol "*" means that the compressor processed the dataset by parts because of memory, time or testing purposes.

Dataset	Method	Mode	Compression		
			Bps	Time (m)	Mem (MB)
Duck size: 1.0 GB SEQ	Gzip	-9 (best)	2.2010	26	6
	DNACompact	default	1.8998	1,656	1,348
	DNAEnc3	standard best	1.8676	118	10,668
	XM	50 experts	1.8601	1,131	32,879
	XM	150 experts	**1.8505**	1,384	33,634
	GeCo	-l 6	1.8570	52	4,800
	GeCo	-l 7	1.8520	64	5,800
Cuckoo size: 1.1 GB SEQ	Gzip	-9 (best)	2.1789	24	6
	DNACompact	default	1.9051	1,720	1,348
	DNAEnc3	standard best	1.8462	128	11,071
	GeCo	-l 6	1.8250	59	4,800
	GeCo	-l 7	**1.8200**	70	5,800
Peregrine Falcon size: 1.1 GB SEQ	Gzip	-9 (best)	2.2074	25	6
	DNACompact	default	1.9038	1,818	1,455
	DNAEnc3	standard best	1.8889	120	11,322
	GeCo	-l 6	1.8790	58	4,800
	GeCo	-l 7	**1.8740**	71	5,800
Assembled human genome (GRC), size: 2.9 GB SEQ	Gzip	-9 (best)	2.1108	70	6
	DNACompact*	default	1.7173	4,764	1,348
	DNAEnc3*	standard best	1.6597	427	4,379
	DNAEnc3	standard best	1.6216	489	14,839
	XM*	50 experts	1.6044	1,170	20,759
	XM*	150 experts	1.5832	1,594	22,295
	GeCo	-l 6	1.5750	131	4,800
	GeCo	-l 7	1.5710	148	5,800
	GeCo	-l 8	**1.5690**	155	6,400

Conclusions

We have presented a compressor that can be applied efficiently to genomic sequences both for non-referential and referential compression. We used a mixture of context models of several orders given by two model classes (reference and target). For high orders, we have created a cache-hash to ensure flexibility given hardware specifications. Moreover, we have introduced the extended context models (XFCM), which can be seen as very flexible context models (fault tolerant). Finally, we have shown the very good adaptability of the compressor to multiple types and characteristics of genomic sequences.

Table 2: Compression benchmarks for state-of-the-art compression tools derived from FASTQ formats. Time is in minutes, while maximum memory peak in MBytes.

Dataset	Method	Mode	Compression		
			Bps	Time (m)	Mem (MB)
SRR003168 361 MB (only bases) FASTQ derived	Gzip	-9 (best)	2.1927	8	6
	fqz_comp	default	1.8029	1	79
	fqz_comp	-e -b -s5+	1.7652	1	199
	fqz_comp	-e -b -s6+	1.7607	1	583
	fqz_comp	-e -b -s7+	1.7602	2	2,070
	fqz_comp	-e -b -s8+	1.7660	2	8,263
	Orcom	-t4 -b256 -p6 -s6	2.1809	1	1,180
	Deliminate	a	1.7381	1	780
	MFCompress	-1	1.7413	3	514
	MFCompress	-2	1.7012	4	514
	MFCompress	-3	1.6405	6	2,322
	GeCo	-l 2	1.5500	17	4,800
	GeCo	-l 4	1.5491	16	3,900
	GeCo	-l 6	1.5344	18	4,800
	GeCo	-l 8	**1.5322**	20	6,400
SRR494099 486 MB (only bases) FASTQ derived	Gzip	-9 (best)	2.2136	11	6
	fqz_comp	default	1.8064	1	79
	fqz_comp	-e -b -s5+	1.7714	2	199
	fqz_comp	-e -b -s6+	1.7496	2	583
	fqz_comp	-e -b -s7+	1.7390	2	2,070
	fqz_comp	-e -b -s8+	1.7418	2	8,263
	Orcom	-t4 -b256 -p6 -s6	1.9495	1	1,252
	Deliminate	a	1.7995	1	780
	MFCompress	-1	1.8810	5	514
	MFCompress	-2	1.8459	5	514
	MFCompress	-3	1.8344	8	2,322
	GeCo	-l 2	1.6670	23	4,800
	GeCo	-l 4	1.6789	22	3,900
	GeCo	-l 6	**1.6662**	25	4,800
	GeCo	-l 8	1.6856	28	6,400

Acknowledgments

This work was partially funded by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 305444 "RD-Connect: An integrated platform connecting registries, biobanks and clinical bioinformatics for rare disease research" and by National Funds through FCT - Foundation for Science and Technology, in the context of the project UID/CEC/00127/2013.

Table 3: Benchmarks for state-of-the-art genomic reference compressors using several references and targets. Time is in minutes, while maximum memory peak in MBytes. The prefix HS, PT, GG, PA, represent, respectively, human, chimpanzee, gorilla and orangutan. The suffix with the numbers represent the chromosome number.

Reference seq	Target seq	Method	Mode	Compression		
				Bps	Time (m)	Mem (MB)
HS18	PT18	GReEn	-	1.2224	2	826
77 MB	71 MB	iDoComp	-	**0.2408**	2	599
SEQ		GeCo	-l 11	0.3176	5	3,938
	GG18	GReEn	-	0.9800	2	826
	72 MB	iDoComp	-	**0.3568**	2	599
		GeCo	-l 11	0.3672	5	3,938
	PA18	GReEn	-	1.7056	2	826
	71 MB	iDoComp	-	0.8224	2	599
		GeCo	-l 11	**0.5992**	5	3,938
PA11	HS11	GReEn	-	1.8784	4	1,112
119 MB	129 MB	iDoComp	-	1.2816	3	1,114
SEQ		GeCo	-l 11	**0.6552**	8	3,938
	PT11	GReEn	-	1.5752	4	1,112
	118 MB	iDoComp	-	1.1352	3	1,114
		GeCo	-l 11	**0.6024**	8	3,938
	GG11	GReEn	-	1.5704	4	1,112
	118 MB	iDoComp	-	1.2784	3	1,114
		GeCo	-l 11	**0.6752**	8	3,938
HS5	PT5	GReEn	-	1.3944	5	1,430
173 MB	167 MB	iDoComp	-	0.9352	4	1,420
SEQ		GeCo	-l 11	**0.3568**	10	3,938
	GG5	GReEn	-	1.9040	5	1,430
	147 MB	iDoComp	-	0.9200	4	1,420
		GeCo	-l 11	**0.8632**	10	3,938
	PA5	GReEn	-	1.4632	5	1,430
	165 MB	iDoComp	-	**0.5640**	4	1,420
		GeCo	-l 11	0.6344	11	3,938

References

[1] G. Korodi and I. Tabus, "Normalized maximum likelihood model of order-1 for the compression of DNA sequences," in *Proc. of the Data Compression Conf., DCC-2007*, Snowbird, Utah, Mar. 2007, pp. 33–42.

[2] M. D. Cao, T. I. Dix, L. Allison, and C. Mears, "A simple statistical algorithm for biological sequence compression," in *Proc. of the Data Compression Conf., DCC-2007*, Snowbird, Utah, Mar. 2007, pp. 43–52.

[3] Z. Zhu, J. Zhou, Z. Ji, and Y. Shi, "DNA sequence compression using adaptive particle swarm optimization-based memetic algorithm," *IEEE Trans. on Evolutionary Computation*, vol. 15, no. 5, pp. 643–658, 2011.

[4] A. J. Pinho, P. J. S. G. Ferreira, A. J. R. Neves, and C. A. C. Bastos, "On the representability of complete genomes by multiple competing finite-context (Markov) models," *PLoS ONE*, vol. 6, no. 6, p. e21588, 2011.

[5] T. Bose, M. H. Mohammed, A. Dutta, and S. S. Mande, "BIND–an algorithm for loss-less compression of nucleotide sequence data," *Journal of Biosciences*, vol. 37, no. 4, pp. 785–789, 2012.

[6] W. Dai, H. Xiong, X. Jiang, and L. Ohno-Machado, "An adaptive difference distribution-based coding with hierarchical tree structure for DNA sequence compression," in *Proc. of the Data Compression Conf., DCC-2013*. IEEE, 2013, pp. 371–380.

[7] P. Li, S. Wang, J. Kim, H. Xiong, L. Ohno-Machado, and X. Jiang, "DNA-COMPACT: DNA compression based on a pattern-aware contextual modeling technique," *PLoS ONE*, vol. 8, no. 11, p. e80377, 2013.

[8] M. H. Mohammed, A. Dutta, T. Bose, S. Chadaram, and S. S. Mande, "DELIMINATE - a fast and efficient method for loss-less compression of genomic sequences," *Bioinformatics*, vol. 28, no. 19, pp. 2527–2529, 2012.

[9] A. J. Pinho and D. Pratas, "MFCompress: a compression tool for fasta and multi-fasta data," *Bioinformatics*, Oct. 2013.

[10] J. K. Bonfield and M. V. Mahoney, "Compression of FASTQ and SAM format sequencing data," *PLoS ONE*, vol. 8, no. 3, p. e59190, Mar. 2013.

[11] S. Grabowski, S. Deorowicz, and Ł. Roguski, "Disk-based compression of data from genome sequencing," *Bioinformatics*, vol. 31, no. 9, pp. 1389–1395, 2015.

[12] C. Wang and D. Zhang, "A novel compression tool for efficient storage of genome resequencing data," *Nucleic Acids Research*, vol. 39, no. 7, p. e45, 2011.

[13] S. Kuruppu, S. J. Puglisi, and J. Zobel, "Optimized relative Lempel-Ziv compression of genomes," in *Proc. of the 34th Australian Computer Science Conference, ACSC-2011*, vol. 11, 2011, pp. 91–98.

[14] A. J. Pinho, D. Pratas, and S. P. Garcia, "GReEn: a tool for efficient compression of genome resequencing data," *Nucleic Acids Research*, vol. 40, no. 4, p. e27, 2012.

[15] S. Wandelt and U. Leser, "FRESCO: referential compression of highly similar sequences," *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, vol. 10, no. 5, pp. 1275–1288, 2013.

[16] I. Ochoa, M. Hernaez, and T. Weissman, "iDoComp: a compression scheme for assembled genomes," *Bioinformatics*, p. btu698, 2014.

[17] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited," *ACM Trans. Inf. Syst.*, vol. 16, no. 3, pp. 256–294, 1998.

[18] K. Sayood, *Introduction to data compression*, 3rd ed. Morgan Kaufmann, 2006.

[19] D. Pratas, A. J. Pinho, and J. M. O. S. Rodrigues, "XS: a FASTQ read simulator," *BMC Research Notes*, vol. 7, no. 1, p. 40, 2014.

[20] S. Deorowicz and S. Grabowski, "Compression of DNA sequence reads in FASTQ format," *Bioinformatics*, vol. 27, no. 6, pp. 860–862, 2011.

[21] S. Deorowicz, A. Danek, and M. Niemiec, "GDC 2: Compression of large collections of genomes," *Scientific Reports*, vol. 5, no. 11565, pp. 1–12, 2015.