Complementary Contextual Models with FM-index for DNA Compression

Wenjing Fan*, Wenrui Dai†, Yong Li*, and Hongkai Xiong*

*Department of Electronic Engineering
Shanghai Jiao Tong Univ.
Shanghai 200240, China
{fanwenjing,marsleely,xionghongkai}@sjtu.edu.cn

†Department of Biomedical Informatics
University of California, San Diego
CA 92093, USA
wed004@ucsd.edu

Abstract

Demanding for efficient compression and storage of DNA sequences has been rising with the rapid growth of DNA sequencing technologies. Existing reference-based algorithms map all patterns to regions found in the reference sequence, which lead to redundancy of incomplete similarity. This paper proposes an efficient reference-based method for DNA sequence compression that integrates FM-index and complementary context models to improve compression performance. The proposed method introduces FM-index to represent the full-text matching for exact repeats between the target and reference sequences. For unmatched symbols, complementary context models are leveraged to make weighted estimation conditioned on variable-order contexts. Reversed reference index is used to guarantee the longest match of variable-length substrings. Experimental results show that the proposed method can achieve a 213-fold compression ratio when tested on the first Korean personal genome sequence data set.

1 Introduction

DNA sequences are large codified messages, from an alphabet of four symbols 'A', 'C', 'G', 'T'. With the development of high-throughput sequencing technologies, DNA sequences have been widely used in the fields of biology, medicine and diagnostics. Rapid growth of massive genome databases has posed a challenge on the storage of sequencing data. However, DNA sequences are highly redundant with the emergence of approximate repeats, especially for those from the same species. Specifically, there are large quantities of repeat patterns between DNA sequences because most of the patterns imply biological significance with regular combinations by symbols 'A','C','G','T'. Therefore, the compression of similar sequences is of great importance for the storage and access of the segments, which leads to the development of reference-based DNA sequence compression.

Reference-based algorithms have been developed to compress DNA sequences from similar species. RLZ[1] used the reference sequence as an index to compress other sequences via the relative Lempel-Ziv algorithm. The following RLZ-based methods selected an appropriate reference for compressing. However, RLZ is restricted for applications with arbitrary alphabets. Later, GRS[2] measured the differential rate

The work was supported in part by the NSFC under Grants 61425011, U1201255, 61271218, 61501294, 61501293, in part by the China Postdoctoral Science Foundation under Grant 2015M581617, and in part by the NSFC under Grant 61529101.

2375-0359/17 $31.00 © 2017 IEEE
DOI 10.1109/DCC.2017.50

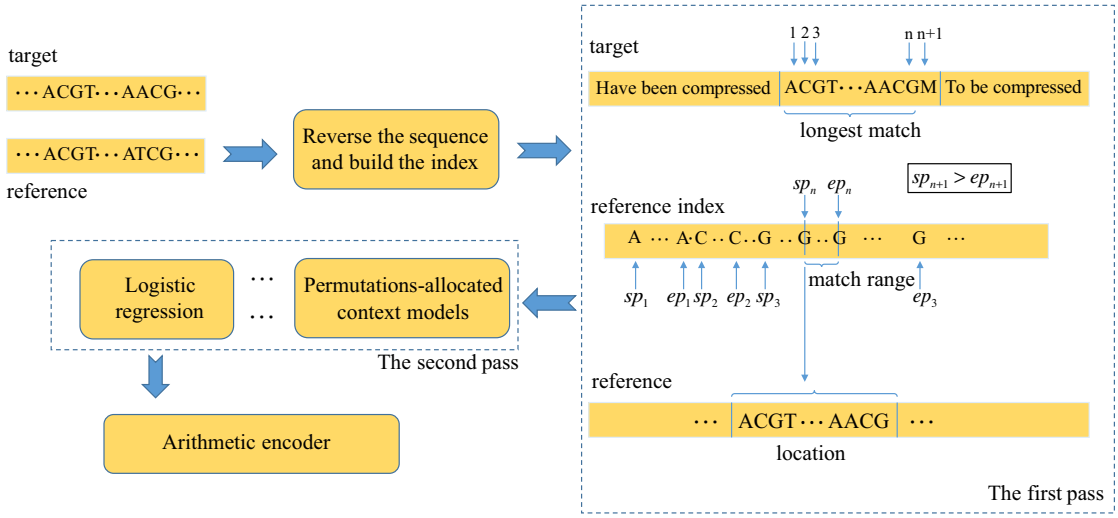

Figure 1: The framework of the proposed method. In the first pass, the variable-length substring is matched and located using the reference index. The remaining unencoded symbols are compressed by the synthesis of complementary contextual models in the second pass.

between target and reference sequences without requiring any additional information. Huffman coding was adopted to encode their differences. GReEn[3] adapted the probabilistic model for arithmetic coding based on the distribution of symbols. In [4], a two-pass compression framework COMPACT was developed to compress unmatched symbols based on the complementary contextual models. But it would fail when the sequences were not well aligned. Ferragina and Manzini presented the self-index algorithm[5] which leveraged the relationship between the Burrows-Wheeler Transform[6] and the suffix array data structure[7] to achieve efficient substring searching in the space of the stored text. Kuma et al. developed a BWT-based compression algorithm[8] to support short read alignment. FM-index was utilized in [9] for compression but it only worked for small matched patterns.

In this paper, we propose an efficient reference-based compression method that incorporates FM-index into complementary contextual models for improved performance. In the first pass, the proposed method can locate the longest match using the inverse of reference index in an efficient manner. We adopt self-index to compress and store the reverse index of reference as a reference sequence for compression and decompression. For unmatched symbol, the complementary contextual models integrate variable length pattern search to further improve the compression performance in the second pass. Experiments show that the proposed method outperforms the state-of-the-art in the compression ratio.

The rest of the paper is organized as follows. Section 2 overviews the Burrows-Wheeler transform (BWT) and suffix array. In Section 3, we propose a reference-based compression method based on FM-index and complementary contextual models. Section 4 provides the experimental results in terms of compression ratio and time cost. Finally, Section 5 concludes this paper.

2 Background

Given a text T[1...n], which is defined as a string of length n over the alphabet Σ = {A,C,G,T}, T[i,n] denotes the i-th suffix and T[i,j] denotes the substring of T determined by i and j. The suffix array, denoted SA[1...n], is defined as the index number that gives the starting positions of all suffixes of T in lexicographical order. In other words, SA[i] = j if the suffix T[j,n] is lexicographically the i-th smallest suffix among all suffixes of T. T# represents a joint string which appends a character #, lexicographically smaller than any others, at the end of T. The suffix array of a text with n characters requires nlogn bits of memory in addition to the text, which is not suitable for large-scale sequences.

The Burrows-Wheeler transform implements a reversible transform, which is called "block-sorting transform". Firstly, a string, which has n symbols, is permutated by shifting n cycles of T#. Then the resulting rotations sort in lexicographic order and construct a n×n matrix M_T. Taking the last column of M_T, the resulting string T^{bwt}(also marked as L) can "group together" several occurrences of the same characters, although it has no compression compared with the original. DNA sequences has a small alphabet of symbols so that it can get a low entropy string T^{bwt} and turn out to be highly compressible.

Remarkably, the original T can be recovered from T^{bwt} using reverse transform in linear time. The first column of M_T in lexicographic order, written as F, is corresponding to L. To find the mapping relationship of the first and the last column of M_T, we transform the expression L[i] = F[j] to j = LF(i). The LF mapping is computed as follows:

$$LF(i) = C[s] + Occ(s, i) \tag{1}$$

Here the symbol s = $T^{bwt}[i]$, C[s] represents the total number of characters which are alphabetically smaller than s, Occ(s,i) denotes the number of occurrences of character s in the prefix $T^{bwt}[1, i]$. Figure 2 shows an example of Burrows-Wheeler transform for the string T = cattacaggac and LF mapping results. We can get LF(8) = C[#] + Occ(#,8) = 1, thus T[n] = #. T[$n-1$] = $T^{bwt}[LF(8)]$ = c. Consequently T[$1, n$] can be recovered by backward recovery.

3 Complementary contextual models with FM-index

Compressing the similar sequences with reference sequence efficiently reduces the storage space. The compression process of FM-context needs two steps. First, the reverse of the reference index should be constructed based on Burrows-Wheeler transform. The index is the only auxiliary structure used in the compression and decompression. By using the index for variable-length search, the longest matching substring can be found and located. The algorithm is independent on the alignments between sequences. If the matched length satisfies the minimum benchmark, it is encoded with a triploid. Later, the remaining uncoded symbols are compressed by the synthesis of complementary contextual models, which can encode symbols not in the reference alphabet.

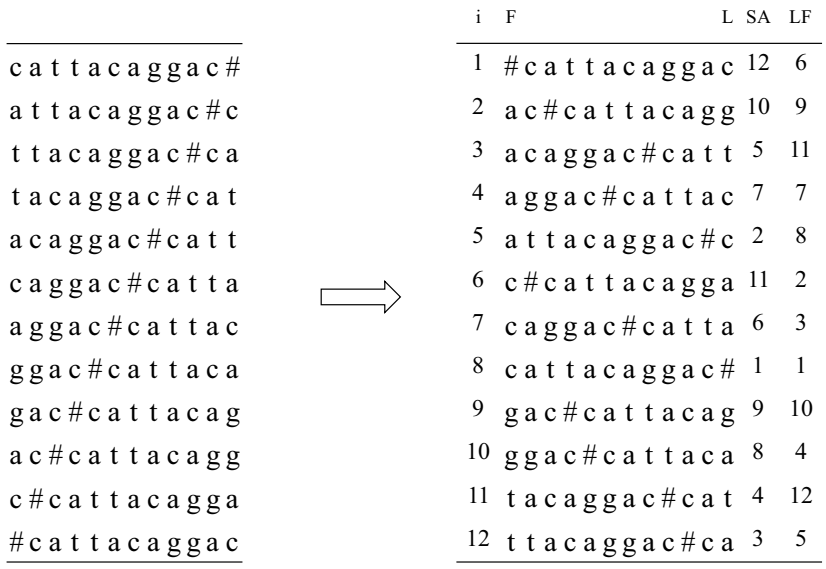

Figure 2: Example of Burrows-Wheeler transform process for the string T = *cattacaggac*. T^{bwt}(L) = *cgtccaa#gata* obtained from the right matrix M_T. F denotes the first column of M_T, sorted by lexicographical order. LF gives the mapping relationship between L and F.

3.1 Variable-length search using FM-index

The FM-index is a compressed self-index based on Burrows-Wheeler transform, because of its efficient searching for occurrences of a specified length pattern P as a substring of the text T. Text T needs to be preprocessed in advance whereas pattern P is provided according to the query requirements. Because of its smaller space occupancy and shorter time for query than other indexes, FM-index is used for variable-length matching between sequences.

Recall that all rows are sorted in lexicographical order in M_T. Given a pattern P, all occurrences of P in T have a corresponding row in M_T within the range (sp, ep), with an initial state (1,n). The FM-index consists of the data structure required to compute C[·], Occ(·), and LF(·) according to Eq.(1). C[·] is directly stored in $|\Sigma|\log n$ bits. To compute Occ(s,i) in a constant time, the FM-index stores a compressed representation of T^{bwt} together with some auxiliary information. LF(i) is computed provided that we have access to $T^{bwt}[\cdot]$. We use the backward searching to build the full-text index to find the repeated pattern between two sequences, where one is preprocessed as reference, and the other provides variable-length patterns in real-time. Algorithm 1 provides the detailed variable-length match procedures. It starts with the reverse index of the reference sequence. The symbol s for variable-length match is obtained in order from the target sequence. The parameter *sp* points to the first row of the BWT matrix M_T prefixed by P[1,i], while *ep* points to the last row of M_T prefixed by P[1,i]. As the algorithm performs last-to-first mapping, the number of pattern occurrences is determined by ep-sp+1. The search stops until finding the longest match. Figure 2 provides an example for the sequence T = *cattacaggac*, which is the reverse of supposed reference sequence S = *caggacattac*. Suppose we have searched for the longest match from the start position of T with predetermined

range (2,3) of the rows corresponding to the prefix *ac*. We can now determine the rows in M_T that match *gac* by performing sp = C[*g*] + Occ(*g*,1) = 9 and ep = C[*g*] + Occ(*g*,3) + 1 = 9. Thus the rows prefixed by *gac* are bounded within (9,9). If next symbol is *g*, we can find matches, while other symbols are not matched. The result in the above example determines the number of the substring *cag* in T and finds the longest substring until sp > ep. Thus, the running time of finding matches is dominated by the cost of the 2i computations of the value Occ($\cdot$). It should be noted that every row in M_T is prefixed by some suffixes of T. For example, the ninth row of M_T is prefixed by the sequence suffix T[9,11] = *gac* in Figure 2. When sp $\leq$ ep, only the first matched sequence needs to be specified. When i = sp, Algothrim 1 offers a way to find the starting position of the substring prefixed in the i-th row of M_T. We use the backward searching for text T and find the substring position by LF mapping until the searching row is marked, as described in Section 2.

To improve compression performance, we introduce a parameter *minlen* to indicate minimum matched length to encode. How to choose the value of this parameter will be discussed in Figure 3. We represent each match with a triploid $\langle f,P,L\rangle$ where f is one-bit matching flag indicating whether the matched length can obtain lower bound for compression, L is the matching length and P is the matching offset. (L - *minlen*) is encoded instead of L to get a shorter bit code. Since the estimated probabilities for L and P decrease with the growth of their values, matched length L and the offset P are encoded using Gamma coding[10]. Gamma coding uses $(2\lfloor log_2 \mathrm{n} \rfloor+1)$ bits for input n as described in Algorithm 1.

3.2 Complementary contextual models

The matched triploids from the first pass can be encoded with binary coding. Subsequently, the remaining uncoded symbols are encoded with complementary contextual models in the second pass. Complementary contextual models are able to accurately estimate a weight probability for prediction conditioned on certain prior knowledge of the symbol/bit to be encoded.

To exploiting varying correlations within DNA sequences, a variety of contextual models are combined, including non-sequential models, sequential models and other available models. We take third order sequential and non-sequential models as examples. A third order sequential contextual model for a binary sequence uses the previous three bits as contexts, while third order non-sequential contextual model utilizes the bit y_{t-3} and y_{t-1} to generate the context for y_t. In some cases, however, non-sequential models can give a higher probability and result in a better compression performance. Thus, we adaptively combine these context models for complementary. In practice, maximum order needs to be specified for finite contexts.

Let us denote M_c the number of context models adopted for prediction. Logistic regression model is leveraged to estimate the most probable probability $\mathrm{P}(y_j|t_1^{M_c})$ to encode the next symbol y_j. The estimated probability $\mathrm{P}(y_j|t_1^{M_c})$ can be obtained by weighting all the context models. Let w_i represent the weight of the i-th model, we get

Algorithm 1 Variable-length searching and match encoding

Input: To-be-compressed string S and *reverse index* of reference sequence R
Output: encoding triploid $\langle f,P,L\rangle$; encoded binary bits

```
sp ← 0; ep ← len_R; i ← 0;
while 0 ≤ sp ≤ ep do
    s ← S[i]
    sp ← C[s] + Occ(s,sp-1);
    ep ← C[s] + Occ(s,ep) + 1;
    i ← i+1;
end while
L ← i
if L > L_m then
    t ← 0; k ← sp;
    while row k is not marked do
        k ← LF[k];
        t ← t+1;
    end while
    P = Pos(k) + t;
end if
write(flag,1), encode L and P using gamma coding
if n = 1 then
    write(0,1);
end if
k ← ⌊log2(n)⌋;
write(1,k); write(0,1); Encode n-2^k using binary code
```

$$P(y_j|t_1^{M_c}) = g(\sum_i w_i t_i), \quad g(x) = \frac{1}{1+e^{-x}} \tag{2}$$

As a result, each symbol will be encoded by an arithmetic encoder.

4 Experiment

The compression method is supposed to be given the same index of the reverse reference sequence between the encoder and decoder. The compression ratio is closely related to the similarity of the the reference and target sequences. We compare the proposed method FM-context with three state-of-the-art DNA compression methods: GRS[5], GReEn[3] and Compact[4] over arbitrary alphabets. The experiments are conducted on a computer with Intel(R) Core(TM) 3.2GHz CPU and 32GB of RAM, with C++ performed.

Table 1 shows compression results of GReEn, Compact and FM-context for two different human genome assemblies(KOREF_20090131 and YH) with reference sequence hg18(NCBI36). KOREF_20090131 and YH are two genome databases generated by different organizations. YH is the first diploid genome sequence of a Han Chinese, a representative of Asian population, completed by Beijing Genomics Institude

Table 1: Compression of KOREF2009_0131 and YH using hg18 as reference. 'T' refers to "Total". For ease of comparison, we transforms all characters to lowercase and maps all unknown nucleotides to 'n' before compression. After this transformation, all sequences are composed only of characters from the alphabet{a,c,g,t,n}.

		KOREF_20090131						YH		
Chr	Size	GReEn		Compact		Ours		GReEn	Compact	Ours
	Bytes	MB	Secs	MB	Secs	MB	Secs	MB	MB	MB
1	247 249 719	**0.81**	**14**	1.31	411	1.02	1185	**0.67**	1.09	0.90
2	242 951 149	**0.90**	**13**	1.30	391	1.01	860	**0.74**	1.19	0.93
3	199 501 827	**0.75**	**12**	1.08	304	0.86	767	**0.64**	1.02	0.80
4	191 273 063	**0.84**	**11**	1.24	392	0.92	1071	**0.66**	1.10	0.85
5	180 857 866	**0.69**	**10**	0.96	306	0.73	840	**0.55**	0.85	0.68
6	170 899 992	**0.69**	**10**	1.09	544	0.79	867	**0.58**	0.96	0.75
7	158 821 424	**0.64**	**9**	0.96	302	0.72	753	**0.51**	0.81	0.65
8	146 274 826	**0.57**	**9**	0.88	279	0.64	567	**0.47**	0.78	0.60
9	140 273 252	**0.44**	**8**	0.75	224	0.59	548	**0.36**	0.62	0.50
10	135 374 737	**0.54**	**8**	0.80	228	0.63	642	**0.43**	0.70	0.56
11	134 452 384	**0.52**	**8**	0.83	257	0.62	632	**0.45**	0.71	0.57
12	132 349 534	**0.54**	**7**	0.75	206	0.60	513	**0.43**	0.66	0.54
13	114 142 980	**0.35**	**6**	0.57	168	0.45	437	**0.32**	0.57	0.44
14	106 368 585	**0.31**	**6**	0.50	144	0.40	501	**0.27**	0.45	0.37
15	100 338 915	**0.27**	**6**	0.44	123	0.37	340	**0.24**	0.42	0.34
16	88 827 254	**0.30**	**5**	0.51	169	0.40	418	**0.26**	0.47	0.36
17	78 774 742	**0.29**	**4**	0.44	135	0.34	300	**0.24**	0.38	0.31
18	76 117 153	**0.31**	**4**	0.44	128	0.34	319	**0.25**	0.40	0.32
19	63 811 651	**0.21**	**4**	0.36	161	0.28	223	**0.17**	0.30	0.24
20	62 435 964	**0.23**	**4**	0.35	107	0.26	248	**0.19**	0.32	0.25
21	46 944 323	**0.14**	**3**	0.27	81	0.19	172	**0.12**	0.21	0.16
22	49 691 432	**0.14**	**3**	0.24	93	0.18	193	**0.11**	0.20	0.16
X	154 913 754	0.55	**9**	0.61	257	**0.51**	628	**0.19**	0.28	0.28
Y	57 772 954	**0.07**	**3**	0.19	156	0.10	262	**0.02**	0.03	0.02
T	3 080 419 480	**11.08**	**176**	16.88	5566	12.95	13286	**8.84**	14.53	11.58

at ShenZhen. KOREF_20090131 is the first individual Korean genome released in December 2008. The alphabets of these two databases are different. KOREF_20090131 consists of 21 symbols such as 'A','C','G','T','N','M' and etc, while all bases in YH are confined to 'A','C','G','T','N' by using only 'N' to represent uncertain bases. We evaluated the proposed method with maximum order m = 16 and minimum matched length L_m(KOREF_20090131) = 25, L_m(YH) = 30. Thus, we transformed all characters to lowercase and mapped all unknown nucleotides to 'n' before compression for fair comparison. The reference sequence hg18 is preprocessed to obtain the reverse index. As shown in Table 1, the proposed method significantly improves the compression performance of COMPACT, although GReEn seems to show a superior performance. It is mainly rooted from the fact that GReEn relies on the probability distribution of characters in the target sequence. The compression performance of GReEn is degraded with no elimination of the effect of character case as shown in Table 3.

Table 2 shows the compression performance for the TAIR9 version of the Ara-

Table 2: Arabidopsis thaliana genome: compression of TAIR9 using TAIR8 as reference. The |C| column indicates the size of the alphabet of the target sequence.

Chr	|C|	Size	GRS		GReEn		Our method	
		Bytes	Bytes	Secs	Bytes	Secs	Bytes	Secs
1	11	30 427 671	715	**7**	1551	10	**591**	142
2	11	19 698 289	385	**4**	937	6	**354**	94
3	10	23 459 830	2989	**6**	1097	7	**499**	86
4	7	18 585 056	1951	**5**	2356	**5**	**1580**	89
5	5	26 975 502	604	**6**	618	8	**276**	109
Total	–	119 146 348	6644	**28**	6559	36	**3300**	520

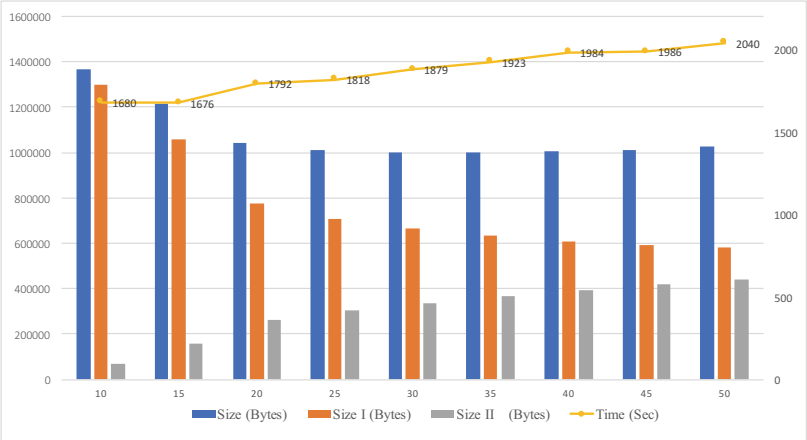

Figure 3: The relationship between the compression performance and the minimum matched length. Size represents the compressed size in total. Size I is from the first pass, and Size II is from the second pass.

bidopsis thaliana genome using the TAIR8 version as reference. The original sequence alphabets have been preserved. In general, FM-context generates a compressed file with 3300 bytes, while GReEn requires 6559 bytes and GRS needs a little more. When compared with GReEn and GRS, FM-context is shown to achieve the best performance in terms of compression ratio for each sequence, with a moderately increased time cost.

Moreover, we discussed the relationship between the compression performance and the parameter of minimum matched length to balance the two passes. Large matched length reduces the number of short substring matches and takes advantage of complementary contextual models. On the contrary, a small one provides more matches for the first pass and makes for better compression ratio and less time consumption. Figure 3 depicts two-pass compression performance for human genome KOREF_20090224 chr1 using KOREF_20090131 chr1 as reference. The time cost increases with the growth of the minimum matched length, while the compression performance can be improved using a proper matched length. In Table 3, we provide compression performance over original sequence alphabets, with the minimum matched length $L_m = 30$. The DNA sequences are reduced from 2938 MB(except chrM) to 13.77 MB, achieving an approximate compression rate of 213 folds.

Table 3: Compression of KOREF_20090224 using KOREF_20090131 as reference. The original sequence alphabets have been preserved.

Chr	Size	GRS		GReEn		FM-context	
	Bytes	Bytes	Secs	Bytes	Secs	Bytes	Secs
1	247 249 719	1 336 626	181	1 225 767	**24**	**1 000 014**	1906
2	242 951 149	1 354 059	273	1 272 105	**24**	**999 789**	1457
3	199 501 827	1 011 124	166	971 527	**20**	**744 639**	1186
4	191 273 063	1 139 225	73	1 074 357	**19**	**832 449**	1384
5	180 857 866	988 070	84	947 378	**18**	**724 478**	1256
6	170 899 992	906 116	55	865 448	**17**	**664 017**	1258
7	158 821 424	1 096 646	45	998 482	**15**	**804 107**	1100
8	146 274 826	764 313	36	729 362	**14**	**561 320**	885
9	140 273 252	864 222	35	773 716	**13**	**654 282**	1089
10	135 374 737	768 364	32	717 305	**13**	**562 368**	906
11	134 452 384	755 708	34	716 301	**13**	**551 519**	1074
12	132 349 534	702 040	31	668 455	**12**	**513 537**	851
13	114 142 980	520 598	23	490 888	**11**	**373 386**	755
14	106 368 585	484 791	23	451 018	**10**	**352 759**	796
15	100 338 915	496 215	20	453 301	**10**	**365 596**	681
16	88 827 254	567 989	20	510 254	**9**	**421 091**	667
17	78 774 742	505 979	18	464 324	**8**	**373 841**	553
18	76 117 153	408 529	16	378 420	**8**	**292 077**	612
19	63 811 651	399 807	14	369 388	**6**	**296 189**	455
20	62 435 964	282 628	13	266 562	**6**	**205 654**	409
21	46 944 323	226 549	9	203 036	**4**	**161 245**	333
22	49 691 432	262 443	10	230 049	**4**	**192 361**	357
X	154 913 754	3 231 776	74	2 712 153	**15**	**2 373 833**	1632
Y	57 772 954	592 791	32	481 307	**5**	**420 482**	453
Total	3 080 419 480	19 666 608	1317	17 370 903	**298**	**14 441 033**	22 055

5 Conclusions

We have proposed the efficient compression method called FM-context, which is applied to compress DNA sequences using reference without restriction of the alphabet. The proposed method combines FM-index and complementary contextual models to compress DNA sequences efficiently. Variable-length search for longest repeats gives a full-text matching and encodes the matches with the reference index. For the remaining uncoded sequence, we use the mixed contextual models for more compression. FM-context is not limited to a certain length of the searched pattern and it works well for large DNA sequences. Experimental results show that FM-context can achieve good performance in compression ratio of DNA sequences. In future, we would modify the alphabet mapping in the FM-index so that the same symbols with different cases could "group together". This would improve the performance of the proposed method in raw DNA sequences.

References

[1] S. Kuruppu, S. J. Puglisi, and J. Zobel, "Relative Lempel-Ziv Compression of Genomes for Large-Scale Storage and Retrieval," in *International Symposium on String Processing and Information Retrieval*, 2010, pp. 201-206.

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

[3] 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, pp. e27-e27, 2012.

[4] 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, pp. e80377, 2013.

[5] P. Ferragina and R. Venturini, "Indexing compressed text," *Journal of the ACM*, vol. 52, no.4, pp. 552-581, 2005.

[6] M. Burrows and D J. Wheeler, "A block-sorting lossless data compression algorithm," *Technical Report Digital Src Research Report*, vol. 57, no.4, pp. 425, 1995.

[7] U. Manber and G. Myers, "Suffix arrays: a new method for on-line string searches," *SIAM Journal on Computing*, vol. 22, no.5, pp. 935-948, 1993.

[8] S. Kumar, S. Agarwal, and R. Prasad, "Efficient Read Alignment Using Burrows Wheeler Transform and Wavelet Tree," in *Second International Conference on Advances in Computing and Communication Engineering*, 2015, pp. 133-138.

[9] P. Prochazka and J. Holub, "Compressing similar biological sequences using fm-index," in *Proceedings of the IEEE Data Compression Conference (DCC)*, Snowbird, Utah, USA, Mar. 26-28, 2014, pp. 312-321.

[10] P. Elias, "Universal codeword sets and representations of the integers," *IEEE transactions on information theory*, vol. 21, no.2, pp. 194-203, 1975.