TBFQC-A FASTQ Compressor Based on Karp Rabin and Levenshtein Distance

Yingjie Chen

College of Information Science and Technology, Jinan University, Guangzhou 510632, China;

lionandcross@163.com

Abstract

Due to the development of sequencing technology, the volume of gene data grow explosively and post great pressure to storage and transport. To solve this problem, this paper present TBFQC, a lossless and reference-based method, to compress the FASTQ file, which is the most widely used file format for gene data. For sequence part of FASTQ file, TBFQC employ Karb Rabin algorithm to build index table of reference file for mapping the target sequence to reference one. Then the difference between target and reference would be found through Levenshtein Distance and replace the original data. Deflate and RLC would be use to handle the rest part of FASTQ. Finally, TBFQC applies xz for obtaining higher compression ratio. The performance experiment shows that TBFQC is a valid solution to reduce the volume of FASTQ file.

Keywords

Fastq, Reference-based Compression, Karp Rabin Algorithm, Levenshtein Distance.

1. Introduction

Sequencing is a technique that analyzes and measures the base sequence of living thing from its blood or organism. The earliest sequencing technology was the bright chain termination dideoxy chain termination method invented by Sanger[1] in 1977 and the sequencing technique of chemical degradation by Maxam and Gilbert[2]. The sequencing technology represented by this method is called the first generation sequencing technology, and its advantage is high accuracy. The complete genome sequencing of humans completed in 2003 is based on a generation of sequencing technology[3]. However, due to the high cost, low throughput and long time of this kind of technology, it can not meet the growing demand of whole genome sequencing, which affects the large-scale application of sequencing technology. Therefore, the next generation of sequencing technology (NGS) has emerged. NGS is also known as HTS High-throughput sequencing, including mature second-generation sequencing and the development of third- and fourth-generation sequencing technologies. At the expense of accuracy, NGS increases the throughput of sequencing, greatly reducing cost and time, making large-scale sequencing of whole genomes possible. On the other hand, with the application of biological information analysis technology, cancer treatment, precision treatment and other fields have important applications[4].

With the development of sequencing technology and the important application of genes in various fields, genetic data has exploded. According to the growth of past gene sequencing data, genetic data can be doubled every 4-5 months. Even some studies speculate that after 2025[5], the growth rate of genetic data will exceed the video, aviation and twitter data. On the other hand, the cost of sequencing the whole genome has dropped below 1,000 since 2017[6], and will continue to decrease in the future. Thus, the cost of sequencing will be less and less restrictive to the growth of genetic data.

The explosive growth of genetic data has brought enormous storage and transmission pressure to bioinformatics researchers, which has restricted the further research and application of genetic data. Computer is the main carrier for storing and processing genetic data. Although it will continue to develop with the advancement of science and technology, its development rate is far less than the

growth rate of genetic data. Therefore, researchers are more inclined to use compression to deal with the large amount of genetic data.

Compression can effectively reduce the size of computer files, save disk storage space, enable large files to be transmitted faster on the network, and simultaneously solve the storage and transmission problems caused by massive genetic data. However, the general compression method can compress the gene data to the original 1/3-1/4[7] , and the compression effect still has not achieved satisfactory compression effect. Traditional general compression algorithms are not fully applicable to this problem. It is very meaningful to research and develop a compression algorithm for genetic data to solve the problem of massive gene data.

The first compression purposed for gene data is Biocompress[8], which use the Fibonacci code to compress the repeat and plalindrome occur in base sequence. After that, similar algorithm such as DNACompress[9], CDNA[10], XM[11] are presented.

However, gene data are saved as specified format,such as FASTA and FASTQ with additional information. The FASTA purposed method include DELIMINATE[12], MFCompress[13] and LEON[14]. And the FASTQ purposed methods incldue DSRC[15], Fqzcomp[16] and Quip[17].

Beside those ,there are a special way to compress gene data that called reference-based method. This method map the target sequence to known reference genome data and find the difference between them. The different message replace original sequence in order to compress. The reference genome data must be same species. Due homologous species have a genetic similarity of over 99 percent ,this method can theoretically achieve a greatly reduced volume of genetic data.

This paper presents a lossless and reference-based compression method for FASTQ files, called TBFQC, to reduce the volume of gene data efficiently. TBFQC apart the FASTQ file into three part, that include identifier, sequence and quality score and use different scheme to compress them. For sequence part, TBFQC design a reference-based method that applies Karb Rabin[18] algorithm to map the target sequence to reference and Levenshtein Distance[19] to find out the difference between target and reference sequence. Deflate and Run Length Coding[20]（RLC） are used to compress identifer and quality score respectively. Finally, after those three part are encoded, TBFQC employs the general compressor, xz, to compress the encoded result again for further compress. Out experiment demonstrates that TBFQC could get a great compress ratio for FASTQ file.

The remainder of this paper is arranged for this flow. Section 2 introduces the prolegomena used in TBFQC. Section 3 illustrates the framework and implementation of different scheme in TBFQC. Section 4 presents the performance experiments about TBFQC on the HTS data sets from different species and discusses some observations found in the experiment. Finally, Section 5 draws conclusion based the experiment and explains some future work.

2. Materials and Methods

2.1 FASTQ

FASTQ is the most widely used HTS genome file format that saved base sequence and additional information in ASCII coding. A FASTQ file contains many sequence blocks which represent four lines. As depicted in Fig 1, Line 1 begins with character `@' and is followed by the identifier as the identification of this sequence; Line 2 is the raw nucleotide sequence that use character to represent the four different kind of bases and ; Line 3 starts with character `+' and can be optionally followed by the same identifier with Line 1; Line 4 is the quality score, where the number of letters must be equal to the symbols in line 2.

2.2 Karp Rabin Algorithm

The Karp Rabin (KR) algorithm, or Robin-Karp algorithm, is a string search algorithm proposed by Richard M. Karp and Michael O. Rabin in 1987, which is often used for pattern matching problem. The pattern matching problem is that finding the position of a string in text. The string is called pattern string and the text is called main text.

The KR algorithm combines the hash function algorithm to effectively reduce the search time. Before describe the algorithm, there are some notation should be decided. P means the pattern string and M means the main text. The KR algorithm is used to find the position of P in T. The length of P and T is m and n respectively. The hash function used in here is H(x).This specific algorithm is executed as follows:

1)First, calculate the hash value of P and get H(x).

2) Starting from i=1 until i equal n, calculate the hash value of H(M[i...i+m]). Then compare whether H(M[i...i+m]) and H(P) are equal.

3) If the two hash values are the equal, it is considered that M[i...i+m] and P may be the same, so compare M[i...i+m] and P character by character. Output i if it is completely consistent, otherwise set i equal i+1 return step 2.

The example of the KR algorithm is shown in Fig 2. The pattern string P is `GTCCA' and has a length of 5. The main text M is `AAGGTCCAGTCA'. First get the hash value that H(`GTCCA'). Then, starting from the starting position of M, a string of length 5 is taken to calculate its hash value. The first sub-string is the 1st to 5th digits of M, that `AAGGT', whose hash value is H(`AAGGT'). It is not equal to hash (GTCCA).So, take the second sub-string, namely the 2nd to 6th digits of ***M***, `AGGTC', and get H(`AGGTC') . However , they are not equal . Repeat those steps, then find the same hash value in the position 4th ot to 9th. The two strings are compared character by character, and found that they are completely same. It means that ***P*** occur in position 4 of ***T***.

```
@ERR005143.2123189 ID49_20708_20H04AAXX:7:179:804:68/1
CGTATCGCGCAATGGCCGGGTGATCTGACGCGTGAT
+
IIIIIIIIICIIIIIIII%I-I;II+:<8>)G&0
```

Fig 1. It is a sequence block in an example FASTQ file. A block include a nucleotide sequence, its identifier and quality score. A FASTQ file contains many such block.

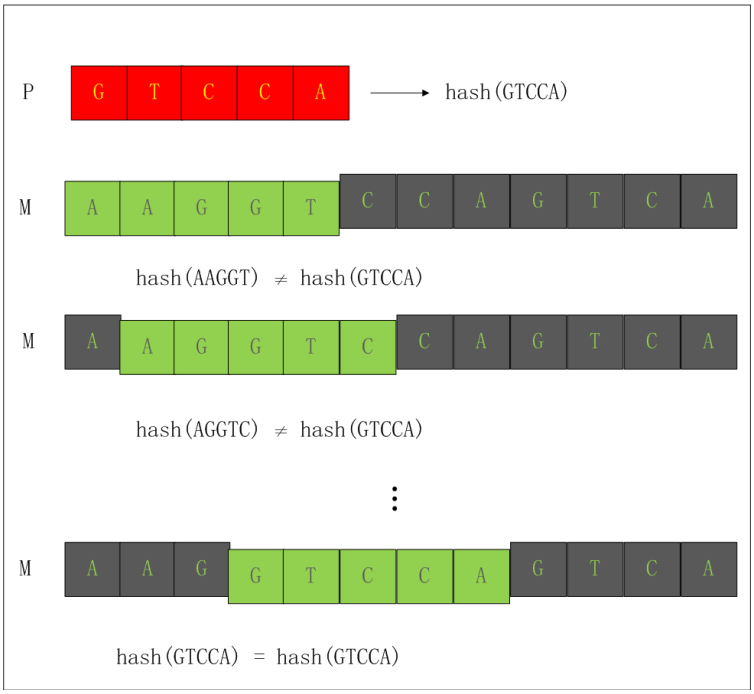

Fig 2. The example of KR algorithm. P is pattern string and M is the main string.

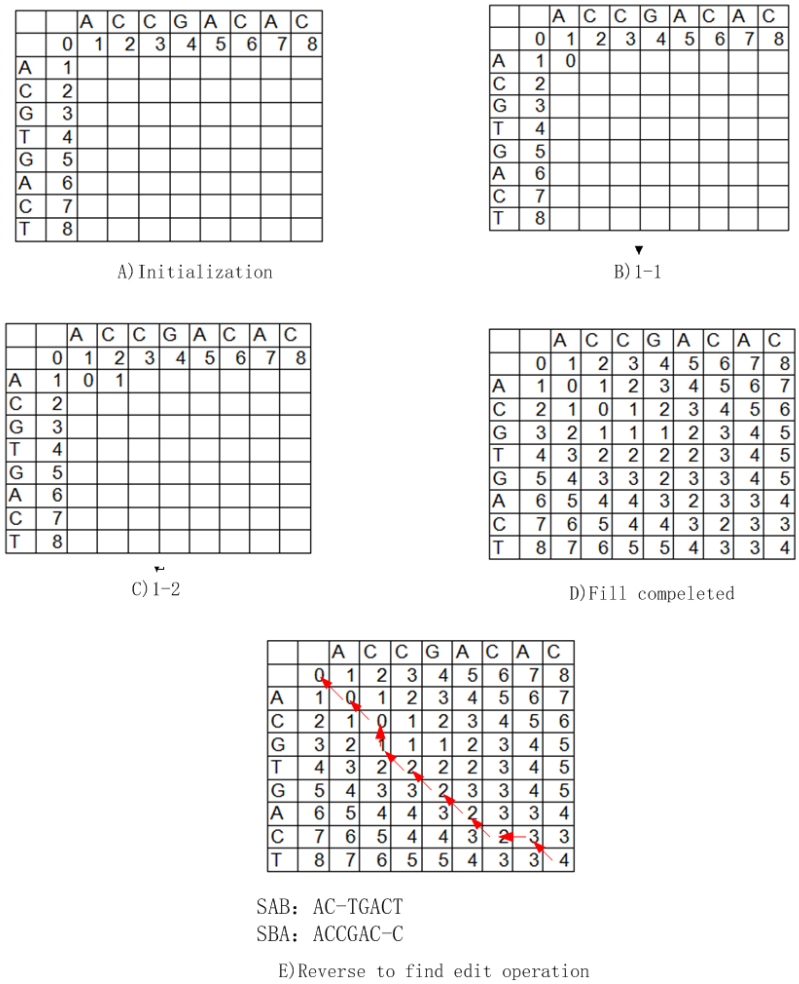

Fig 3. The example of LD algorithm. P is pattern string and M is the main string. Figure a to d is the flow about the construction of LD matrix. And the figure e show how to get edit operation

2.3 Levenshtein Distance Algorithm

The Levenshtein Distance, also called edit Distance, refer to the minimum time of basic modify operation to change a string another.

The basic modify operation, include three type that are insertion, deletion and substitution of one character. For example, the Levenshtein Distance of `ACTG' and `NCG' is 2,

There is Levenshtein Distance(LD) algorithm that can calculate the edit distance as well as find the edit operation, namely basic modify operation, between two strings. Consider that there are string ***A*** and ***B*** whose length is ***m*** and ***n***. And there are two string ***SAB*** and ***SBA*** which is null at the begin. The specific process is as follows:

Create a matrix M with ***m+1*** lines and ***n+1*** columns. Fill this matrix as the formula below. ***temp*** is 1 if the A[i] equal B[j], or 0 if otherwise.

$$M[i][j] = \begin{cases} i, & \text{if j} = 0 \\ j, & \text{if i} = 0 \\ min\left(M[i-1][j]+1, M[i][j-1]+1, M[i-1][j-1]+temp\right), & 0 < i < m+1, 0 < j < n+1 \end{cases} \quad (1)$$

2) Output the M[m+1][n+1] as edit distance.

3) Backtracking from M[m+1][n+1] to the M[0][0]. Find the minimum one from the neighbour of M[i][j], namely M[i-1][j],M[i][j-1] and M[i-1][j-1]. If there are two or more minimum one, then choose as diagonal, left ,right order. Take the minimum one as the next element to compare and set the ***SAB*** and ***SBA*** as follow.

4) Reverse the ***SAB*** and SBA. And the edit operation can be got from string A, B, SAB and SBA

The example process of using the LD algorithm to find the edit distance and edit operation is shown in Fig 3. The strings A and B are `ACGTGACT' and `ACCGACAC' respectively. Fig 3.A is the initialization of matrix M. Fig 3.B calculates the value of M[1][1]. As A(1) = B(1) = A, temp is 0, then the minimum of its neighborhood is 0. Thus M[1][1] is 0. And so on as Fig 3.B to Fig 3.C. Fig 3.D is the result of filling. Since M[8][8] is 4, the edit distance between A and B is 4. Then start from $M[8][8]$ to get SAB and SBA. As shown in Fig 3.E, the red arrow is the backtracking path. The three elements near M[8][8] are equal, so choose the element in the diagonal direction, which is M[7][7] as the next element. Character `T' and `C' are added into ***SAB*** and ***SBA***, respectively. Then, backtracking from M[7][7] , the elements of M[7][6] are the smallest one, so choose it as next one and put the elements `C' and `-' to the ***SAB*** and the ***SBA***, respectively. Repeat the steps above until M[0][0] and reverse SAB and ***SBA*** and archive `AC-TGACT' and `ACCGAC-C', respectively. Then A can transform into B by deleting the third character , replacing the fourth bit by 'C', inserting 'A' into seventh digit and replacing the eighth bit by' C'.

3. TBFQC

The FASTQ file contains multiple gene data blocks, each of which can be divided into identifier, sequence, and quality score. Since the data of those three parts have their own feature, the implementing of this scheme is the following steps. First The FASTQ file is divided into three parts: identifier, base sequence and quality score . Secondly, different coding methods, Deflate, KR algorithm combined with LD algorithm and RLC, are adopted for the three parts. Finally, for further compressed, those encoding results are compressed by using xz, a general compressor. The program framework illustrated in Fig 4.

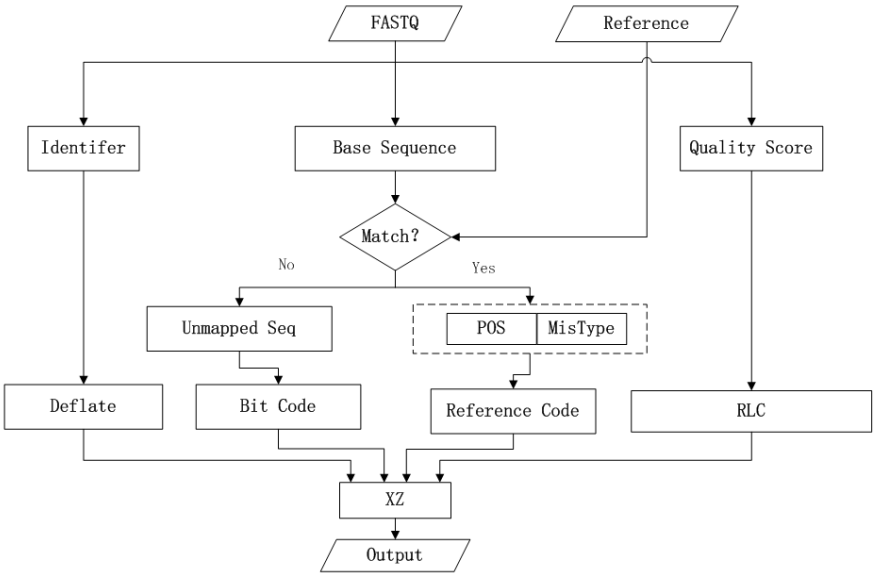

Fig 4. The framework of TBFQC compressor

3.1 Deflate compression of identifier

The identifier includes the first and third rows of each gene block in the FASTQ file. Since the identifer of third line is optional and same with first line, compression of this part only consider the feature of the first line. The identifer is used to uniquely identify the short read sequence. It starting with the character '@', followed by the sequencing instrument model, the flow cell number used for sequencing, the lane number, the sample number, whether it is single-ended, sequence read length, etc. As all the sequences in a FASTQ file are obtained in one sequencing, the identifer of all sequences in a FASTQ file are highly similar.

Thus, TBFQC use deflate algorithm, a kind of LZ77 style algorithm, to handle identifer. This k After that, the identifier will be further compressed using xz.

Algorithm 1

Require: T target sequence, R Reference sequence, hl hash length,sl search length
Ensure:

```
function CreateIndex(R, hl)
    index ← malloc(N)
    i ← 0
    while i < R.length do
        str ← R[i...i + hl]
        ptr ← hash(str)
        index[ptr] ← i
        i ← i + hl
    end while
    return index
end function

function SearchMap(T, R, hl, sl)
    Word
    index ←+CreateIndex(R, hl)
    for i = 0 → T.length do
        key ← T[i...i + hl]
        ptr ← hash(key)
        if index[ptr] then
            j ← 0
            A ← T[i + j...i + j + sl]
            B ← R[ptr...ptr + sl]
            temp ← LD(A, B)
            while temp and i + j < T.lenght do
                j ← j + sl
                A ← T[i + j...i + j + sl − 1]
                B ← R[ptr...ptr + sl − 1]
                temp ← LD(A, B) Word ← Word + temp
            end while
        end if
    end for
    return Word
end function
```

3.2 Reference-based compression of base sequence

The base sequence only contains five characters `ACTGN'. Since the base sequences among homologous organisms are highly similar, this part of the data is compressed by reference-based algorithm. In general, the reference-based algorithm execute as follow. First, find the position of the target gene sequence on the reference genome. Second, obtain the difference information between the two strings. And finally, encode the difference information and replace the original sequence.

The first step is a pattern matching problem actually. The reference is the main string, and the target sequences are a large amount of pattern strings. Thus, TBFQC try to use the KR algorithm to do it.

Before the search operation,a hash table, or a key-value dictionary of reference genome is constructed . The reference genome sequence, which is always too long, is divided into a plurality of small fragments according to ***k*** bases, and each fragment is calculated its hash value by a hash function. Hash value as key and the offset of base fragment as value would be combined as a pair and put into the dictionary. The hash collision is inevitable, so the linear probing is used to solve this problem.

Then find the matching position of each target sequence in reference genome through the hash table. TBFQC use a sliding window of size ***k*** to calculate a hash value of ***k*** bases in the target sequence. The hash function used in search must be same as the function used in create hash table. If the value of sub-sequence in hash value is not empty, it means that there are a base fragment in genome might same with the sub-sequence. Then TBFQC move into next step of finding the difference information.

In order to facilitate step of finding the difference information, the character N in both the reference genome and the target sequence will be extracted. The N in the reference genome are discarded, and the N in the target sequence are stored by its position. The location information will be used in decompress.

Finding the difference information between the reference genome and the target sequence can be regarded as calculating edit distance and edit operation between them. Therefore, the LD algorithm

is used to compare the two gene data to find the different between matching position of the reference genome and the target sequence. The length of the target base sequence is generally from 50 to 250 bases. If the LD algorithm take the entire sequence as input , the LD matrix is large, which leads to a long operation time. According to the purpose of greedy algorithm, TBFQC does not directly seek for the globally optimal solution, instead of local optimal solution by dividing the problem into pieces, find the optimal solutions of locally problem and then combine them as the solution of the problem. Starting from the matching position of the reference genome and the target sequence, every ***b*** bases is selected as the input string,that ***A*** and ***B***, of the LD algorithm to get the edit distance and edit operation.

The transform operation of transforming the sub-string A of the reference genome to the target sequence string B is taken as a local optimal solution. The sub-string of the next ***b***-length of the reference genome and the target sequence is then used as the new A, B string, and the edit distance and editing operation are again obtained by the LD algorithm, and then merged with the known result. The above steps are repeated until the target sequence is traversed or the edit distance of the current solution exceeds the threshold ***t***. The selection of the threshold t is related to the encoding.

Finally, the matching result is encoded and used to replace the original target base sequence. Two encoding methods are used depending on the matching result. If the match is successful, the matching position and the editing operation that transforming the reference genome into the target sequence is saved to replace the original sequence. The matching position is stored in 4 bytes. There are four types of editing operations, namely matching, replacing, inserting, and deleting. It can be encoded using only the ASCII character set that (A,C,G,T,F,H,L,Y,D) and the numbers from 0 to 9. The number indicates the number of consecutively matched bases. The character `A',`C',`G' and `T' indicate the substitution of the corresponding base. `F',`H',`L' and `Y' indicate the insertion of four bases that `A',`C',`G' and `T', respectively. D is used to indicate the deletion. Not all the sequence can find the matching position in the reference genome. For those unmapped target sequences and the unmapped sub-sequence of the matching target sequence, they are encoded by bit code that A=00, C=01, G=10, T=11. The original ASCII code used one byte, or 8 bits, to represent one base, and under bit coding, one byte can represent four bases. Therefore, the edit distance ***d*** should be less than one fourth of the ***b*** .

The encoded base sequences are further compressed by using the xz universal compression program.

3.3 RLC of Quality Score

The quality score is generated during the sequencing process. Each quality score that corresponds to one base is used to indicate the accuracy of sequencing. The quality score is expressed in ASCII code but in different the coding scheme according to different sequencers. Usually, a FASTQ file contains up to 40 characters for representing the quality score in pseudo-random distribution, so the lossless compression of this part of the data is very difficult.

In fact, as development of sequencing technology, the quality score shows the continuous appearance of the same character. As shown in Figure 1, the short read sequence SRR0016666.1 contains 22 consecutive characters `I'. Based on this feature, the quality score is suitable for using Run-Length-Coding(RLC) to encode.

Consider that ***n*** is the consecutive number of a character, which is called the run length. For the every quality score ***q***, when ***n***>=2 , the consecutive occurrence of q is represented by 2 bytes. The first byte is the ASCII code of q, and the second byte is the run length n. When ***n***>256, consecutive ***q*** will be split into multiple sub-continuous quality score part to encode. When ***n***=1, the byte is represented by 1 byte which is set the highest bit position to 1. The purpose of such encoding is to distinguish the discontinuous characters from the continuous characters in order to facilitate decoding.

For example, the quality score of a sequence is `FFFBHHHHH' and will be encoded as ‘F3’ B_s H5. F and H are ASCII coded characters and is followed by its run length 3 and 5 respectively. B_s is the

result of the highest position after ASCII encoding is 1. The result saved in hexadecimal is that 0x46 0x03 0x42 0x48 0x05.

The quality scores processed by the RLC are further compressed by using the xz universal compression program.

4. Experiments

4.1 Experiment configuration

To evaluate the performance of TBFQC, a machine with 32-core 1.70GHz cores Intel Xeon CPU E5-2609 and 64GB memory in 64-bit CentOS Linux release 7.5.1804(core)was used in the experiments. The compressor written by C++ takes a target FASTQ file and a reference FASTA file as input ,and outputs the compressed file. To obtain the original FASTQ file, the compressed file and the same reference file are required.

The details about experiment data set from different species are shown in Table 1. The FS0024 is provided by a private company. The SRA data that with prefix name of SRR are downloaded from the SRA of NCBI; and the ERR data sets are contributed from The European Bioinformatics Institute(EBI) [21]. All other data sets are paired-end data except ERR231645 and ERR233152. The reference data used in the experiments are downloaded from Ensembl[22].

Since the SRA data have the same identifier in third line of every gene block, they can not be compressed by some related compressor. Those optional identifier in third line of gene block is deleted so that the all compressors can take the data as input.

Table 1 The detail about FASTQ dataset

Dataset	Species	Read Length	Size(GB)	Source	Reference	Ref.Size
FS0024	Sus scrofa	2*135	2.51	Private	Sscrofa11.1	2.4GB
ERR231645	E.coli	51	1.41	EBI	NC_000913	
ERR233152	P.aeruginosa	77	0.72	EBI	Ap014622	
SRR7174087	Homo	2*51	6.59	NCBI	Hg19	
SRR327342	S.cerevisiae	2*63	5.57	EBI	ACFL01	
SRR554369	Pseudomonas	2*100	0.71	EBI	KI517354	
SRR935126	A.thaliana	2*76	9.6	EBI	GCF_000001735.4	
SRR489793	C.elegans	2*101	12.6	EBI	WBcel235	
SRR352384	S.cerevisiae	2*76	9.6	EBI	ACFL01	

4.2 Experiment results

First experiment is used to compare the compress performance of TBFQC with the other state-of-the-art compressor purposed for FASTQ, including DSRC[15], Fqzcomp[16], Quip[17]. The general compressor- GZIP is also included in this experiment and its compress result would be the standard measure in this competition as it is the most widely used compressor to compress FASTQ file. Result of this experiment is shown in Table 2 and it demonstrated that TBFQC can obtain higher compress ratio than the other compressor statistically.

Table 2 Compression result

Data	gzip	dsrc(best)	fqzcomp	quip -r	TBFQC
FS0024	4.17	5.79	7.13	7.65	9.28
ERR231645	2.57	4.33	5.10	4.98	4.53
ERR233152	2.92	4.33	5.92	5.88	5.51
SRR7174087	5.06	9.17	14.44	16.22	15.93
SRR327342	2.92	4.00	4.80	5.01	4.97
SRR554369	2.82	3.88	4.51	4.37	4.93
SRR935126	3.22	4.68	5.64	5.66	5.52
SRR489793	2.85	3.84	4.43	4.43	4.84
SRR352384	4.03	6.63	7.53	8.16	8.38

Avg	3.39	5.18	6.61	6.93	7.10
S.D	0.83	1.77	3.14	3.73	3.71

The compression time of the first experiment is shown in Table 3. The fastest compressor is DSRC and the slower one is TBFQC overall. TBFQC get higher compression at the cost of time.

Table 3 Compression time

Data	gzip	dsrc(best)	fqzcomp	quip -r	TBFQC
FS0024	295	6	40	130	2960
ERR231645	106	4	15	35	435
ERR233152	114	2	24	12	212
SRR7174087	551	12	131	224	3100
SRR327342	711	9	123	151	2716
SRR554369	114	2	20	22	365
SRR935126	1235	10	186	246	4005
SRR489793	1644	12	298	402	7141
SRR352384	1133	10	147	249	3801

Second experiment is used to evaluate the performance of different scheme to handle three components of FASTQ. As illustrated in Table 4, the compression of identifier is best compressed that has the largest average compression ratio 416.88, and the largest standard deviation 826.67. It demonstrated that the scheme for identifier can get higher compression ratio in special case. The scheme of base sequence has second average compression ratio 9.30 but get the smallest standard deviation. It means that compression of sequence is stable and can get a great compression ratio. The smallest compression ratio for quality score means that RLC is not well scheme for quality and should be modified if want to get higher compression ratio for FASTQ compressor

Table 4 Part compression

Dataset	Identifier	Base	Quality
FS0024	7.41	16.98	6.49
ERR231645	7.53	7.85	2.31
ERR233152	8.90	5.67	4.11
SRR7174087	2500.00	8.12	18.18
SRR327342	9.98	8.46	2.32
SRR554369	434.78	8.30	2.87
SRR935126	7.54	9.57	3.33
SRR489793	769.23	7.95	2.79
SRR352384	6.55	10.79	7.56
Avg.	416.88	9.30	5.55
S.D.	826.67	3.19	5.09

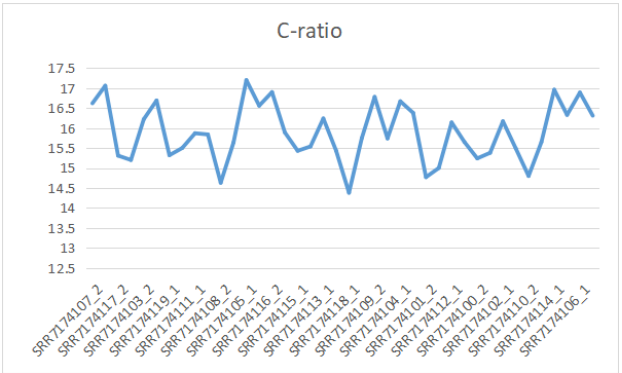

Fig 5. The figure about compression ratio of TBFQC in large scale dataset

Finally, there are 40 FASTQ files to evaluate the overall performance of TBFQC. The range of their size sre from 2.8GB to 4.7GB. Result is shown in Figure \ref{figure:large_compression}. It provided that this method is stable and get a desirable result for large amount of FASTQ files.

4.3 Discussions

The impact of quality score

As shown in Table \ref{table:compression result}, ERR231645,SRR327342,SRR801793,that are microorganism data, have the lower compression ratio than other data set. And we found that they are also get lower compression ratio for the quality score from Table \ref{table:part compression}. The quality score scheme is not suitable for those microorganism data. It is a obstacle for reducing those data volume.

The impact of reference size to compression time. According to the Table \ref{table:compression time}, ERR231645, SRR327342, SRR801793, ERR233152 has the lower compression time than FS0024,SRR7174087,SRR7174187. One of the reason is that the size of former is smaller than the latter. The another one is the size of reference file. Those reference files are from 20MB to 40 MB, but the sus_scofa and human reference is 2.4GB and 3.0GB. Because the size of latter reference are far exceed the former, it cause a huge difference in cost time.

5. Conclusion and Future Work

The exponential growth of genetic data poses a huge challenge for storage and transmission of gene data. Use traditional compression tools to solved this problem become past. This paper present a FASTQ file compressor, as TBFQC, in order to compress the gene data in lossless way. TBFQC aprat the FASTQ file into identifier, base sequence, quality score and use deflate, reference-based and RLC to encode them respectively. The reference-based scheme is combined the Karb Rabin and Levenshtein Distance to achieve difference between target and reference. After the encoded operation, TBFQC use xz to compress the FASTQ file further. The experiment about TBFQC shows that this method can compress the FASTQ file effectively at the cost of time. The quality score scheme is not perfect and should be modified for improving the FASTQ method.

References

[1] Langeveld SA, van Mansfeld AD, Baas PD, Jansz HS, van Arkel GA, Weisbeek PJ. Nucleotide sequence of the origin of replication in bacteriophage phiX174 RF DNA. Nature. 1978;271(5644):417{20.

[2] Maxam AM, Gilbert W. A new method for sequencing DNA. Proceedings of the National Academy of Sciences. 1977;74(2):560{564.

[3] Mestan KK, Ilkhanoff L, Mouli S, Lin S. Genomic sequencing in clinical trials. Journal of Translational Medicine,9,1(2011-12-30). 2011;9(1):222{222.

[4] Meldrum C, Doyle MA, Tothill RW. Next-Generation Sequencing for Cancer Diagnostics: a Practical Perspective. Clinical Biochemist Reviews. 2011;32(4):177.

[5] Stephens ZD, Lee SY, Faghri F, Campbell RH, Zhai C, Efron MJ, et al. Big Data: Astronomical or Genomical? Plos Biology. 2015;13(7):e1002195.

[6] KA W. DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP);.https://www.genome.gov/sequencingcostsdata/.

[7] Deorowicz S, Grabowski S. Compression of DNA sequence reads in FASTQ format. Bioinformatics.2011;27(6):860.

[8] Grumbach S, Tahi F. Compression of DNA sequences. In: Data Compression Conference; 1993. p.340{350.

[9] Chen X, Li M, Ma B, Tromp J. DNACompress: fast and effective DNA sequence compression Bioinformatics. 2002;18(12):1696.

[10] Loewenstern D, Yianilos PN. Significantly lower entropy estimates for natural DNA sequences. Journal of Computational Biology. 1999;6(1):125{142.

[11] Cao MD, Dix TI, Allison L, Mears C. A Simple Statistical Algorithm for Biological Sequence Compression. In: Data Compression Conference; 2007. p. 43{52.
[12] Mohammed MH, Dutta A, Bose T, Chadaram S, Mande SS. DELIMINATE{a fast and efficient method for loss-less compression of genomic sequences: sequence analysis. Bioinformatics. 2012;28(19):2527{2529.
[13] Pinho AJ, Pratas D. MFCompress: a compression tool for FASTA and multi-FASTA data. Bioinformatics. 2014;30(1):117.
[14] Benoit G, Lemaitre C, Lavenier D, Drezen E, Dayris T, Uricaru R, et al. Reference-free compression of high throughput sequencing data with a probabilistic de Bruijn graph. Bmc Bioinformatics. 2015;16(1):1{14.
[15] Roguski u, Deorowicz S. DSRC 2|Industry-oriented compression of FASTQ files. Bioinformatics. 2014;30(15):2213.
[16] Bonfield JK, Mahoney MV. Compression of FASTQ and SAM Format Sequencing Data. PLOS ONE. 2013;8(3).
[17] Jones DC, Ruzzo WL, Peng X, Katze MG. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. Nucleic Acids Research. 2012;40(22):e171.
[18] Karp RM, Rabin MO. Efficient randomized pattern-matching algorithms. IBM journal of research and development. 1987;31(2):249{260.
[19] Levenshtein VI. Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet physics doklady. vol. 10; 1966. p. 707{710.
[20] Robinson AH, Cherry C. Results of a prototype television bandwidth compression scheme. Proceedings of the IEEE. 2005;55(3):356{364.
[21] https://www.ebi.ac.uk/.
[22] http://uswest.ensembl.org/info/data/ftp/index.html.