Annals of West University of Timişoara, ser. Biology, 2018, vol. 21 (2), pp.149-156

APPLYING HASH TO ZIP DNA DATA

Done STOJANOV
Faculty of Computer Science, University Goce Delcev, Krste Misirkov nn – Štip, Republic of Macedonia
**Corresponding author e-mail: done.stojanov@ugd.edu.mk*
Received 3 September 2018; accepted 10 December 2018

ABSTRACT
Current genomic data compression techniques rely on referent sequence. This means that one sequence is used as a template upon which the differences in other samples are tracked. Since none of these techniques works without reference, we propose a novel methodology which does not depend on reference. Even though that our methodology is reference-independent, compression gain of more than 34 % was obtained.
KEY WORDS: *genomic, data, compression, decompression, hash.*

INTRODUCTION

Thanks to the advances in recent genomic sequencing techniques huge amount of genomic data has been collected. This data is stored and offered through public repositories such as: EMBL and GenBank. None of these repositories employs data compression techniques, i.e. sequences and variations of them are stored in raw data format as array of characters. However current research indicates that genomic data compression is possible and highly recommended. It is possible because there is a high percentage of identity between genomic samples that originate from the same species and it is recommended because of the exponential data growth.

The very beginnings of genomic data compression date from 2009, when Brandon (Brandon, 2009) introduced the concept for tracking the differences between a referent genome and genome that ought to be compressed. The compression gain is maximized by mapping the difference information into binary strings applying entropy coding. However some researchers reported difficulties on applying Brandon's methodology, especially if the reference and the sample that has to be compressed significantly differ. Employing more than one reference sequence seems to be the solution of this problem.

For better compression gain, some methods require additional data. Most often this data is presented in form of single-nucleotide polymorphisms and insertions/deletions history. GRS (Wang *et al.,* 2011) was the first software tool which was able to compress genomic data without this data. GRS was upgraded to GreEn (Pinho *et al.,* 2011) which runs at higher speed and it has better compression rate.

Researchers such as (Christley *et al.,* 2008) and (Pavlichin *et al.,* 2013) took the James-Watson's genome as a reference upon which the compression rate was measured. In (Christley *et al.,* 2008) this sequence was made small enough to be sent by mail and further compression down to 2.5 MB applying entropy coding was obtained in (Pavlichin *et al.,* 2013). In both papers single-nucleotide polymorphism information was exploited.

Applying HUFFMAN (Huffman, 1952) or GOLOMB (Golomb, 1966) code results in additional savings. Huffman code was applied in (Tembe *et al.,* 2010) and it resulted in more than 65% of compression gain, while Deorowicz and Grabowski (Deorowicz *et al.,* 2011) limited the application of the compression pattern only to sequences that come from the same species.

Cited papers have one thing in common and this is the use of reference sequence. The reference sequence is used as a template for recording the positions of difference regards other samples and if this sequence is not used, none of the current research works. Therefore in this paper we address the question of compressing genomic data without have to use reference. This is made possible through genomic data hashing and offset tracking approach which is also suitable for fast data decompression. Our methodology results in more than 36 % of compression gain, but since we do not use reference this approach is more reliable than any other.

MATERIALS AND METHODS

In 2005 Reneker and Shyu (Reneker *et al.,* 2005) introduced the concept of genomic data hashing, applying equations (1) and (2). Numerical translations of nucleotides according equation (1) are used to compute the hash of the read $\mathrm{R}, \mathrm{H}(\mathrm{R})$ in quaternary domain, equation (2). The quaternary system was exploited because the DNA is a chain of four nucleotides: A (adenine), T (thymine), G (guanine) and C (cytosine).

$$f(A) \rightarrow 1, f(T) \rightarrow 2, f(G) \rightarrow 3, f(C) \rightarrow 4 \qquad \textbf{(1)}$$

$$H(R) = f(R{:}\, a_0\, a_1 \ldots a_{n-2} a_{n-1}) = \sum_{i=0}^{n-1} f(a_i) \times 4^i = f(a_0) \times 4^0 + f(a_1) \times 4^1 + \cdots + f(a_{n-2}) \times 4^{n-2} + f(a_{n-1}) \times 4^{n-1} \qquad \textbf{(2)}$$

Despite that Reneker and Shyu directed their research towards efficient search of massive genomic database against short DNA query; it is also applicable for data compression purpose. Compressing genomic data applying equations (1) and (2) can be done without any problem, but when it comes to data decompression we may face difficulties and errors in data decryption because one of the nucleotides C (cytosine) is mapped to the same value as the radix in equation (2) which equals 4.

To overcome this problem we propose equation (3) which equals equation (2) but instead of radix 4 we use radix 5. Single nucleotide translations remain the same as given in equation (1), but now, none of translations equals the radix, that will prevent errors in data decryption. Radix change allows the straightforward application of equation (4) for random nucleotide decryption, given $\mathrm{H}(\mathrm{R})$ and k as input. Note that $\mathrm{H}(\mathrm{R})$ stands for the hash of the read, k is the position of decryption inside the read, while mod in equation (4) stands for the operator for computing the remainder.

$$H(R) = f(R{:}\, a_0\, a_1 \ldots a_{n-2} a_{n-1}) = \sum_{i=0}^{n-1} f(a_i) \times 5^i = f(a_0) \times 5^0 + f(a_1) \times 5^1 + \cdots + f(a_{n-2}) \times 5^{n-2} + f(a_{n-1}) \times 5^{n-1} \qquad \textbf{(3)}$$

$$f(a_k) = \left({}^{H(R)}\!/_{5^k} \right) \mathrm{mod} 5 \qquad \textbf{(4)}$$

In order to prove equation (4) we can rewrite equation (3) into equation (5) and equation (6). Equation (6) equals equation (3), but it is written in terms of 5^k as a common factor.

$$H(R) = f(a_0) \times 5^0 + f(a_1) \times 5^1 + \cdots + f(a_k) \times 5^k + f(a_{k+1}) \times 5^{k+1} + \cdots + f(a_{n-1}) \times 5^{n-1}$$

(5)

$$H(R) = f(a_0) \times 5^0 + f(a_1) \times 5^1 + \cdots + 5^k \times (f(a_k) + f(a_{k+1}) \times 5^1 + \cdots + f(a_{n-1}) \times 5^{n-k-1})$$

(6)

Following equation (6) we get that ${}^{H(R)}\!/_{5^k} = f(a_k) + f(a_{k+1}) \times 5^1 + \cdots + f(a_{n-1}) \times 5^{n-k-1}$, equation (7).

$${}^{H(R)}\!/_{5^k} = f(a_k) + f(a_{k+1}) \times 5^1 + \cdots + f(a_{n-1}) \times 5^{n-k-1} \quad \textbf{(7)}$$

Note that all terms in equation (7), without the first, have 5 as a common factor. Therefore we can rewrite equation (7) into equation (8), wherefrom we get that $\left({}^{H(R)}\!/_{5^k} \right) \mathrm{mod} 5 = f(a_k) \mathrm{mod} 5 = f(a_k)$.

$${}^{H(R)}\!/_{5^k} = f(a_k) + 5 \times (f(a_{k+1}) + \cdots + f(a_{n-1}) \times 5^{n-k-2}) \quad \textbf{(8)}$$

This means that if we know the hash of the read $H(R)$ and the position of decryption k inside the read we can decrypt a_k applying equation (4). Since $\left({}^{H(R)}\!/_{5^k} \right) \mathrm{mod} 5 = f(a_k) \in \{1,2,3,4\}$ we get that if 1 then $a_k = A$ (adenine), if 2 then $a_k = T$ (thymine), if 3 then $a_k = G$ (guanine) and if 4 then $a_k = C$ (cytosine).

For closer look, we will demonstrate this approach to the short read $R = CATGAT$. Applying equations (1) and (3) this read is hashed to 7309 which is computed as: $H(R) = f(C) \times 5^0 + f(A) \times 5^1 + f(T) \times 5^2 + f(G) \times 5^3 + f(A) \times 5^4 + f(T) \times 5^5 = 4 \times 1 + 1 \times 5 + 2 \times 25 + 3 \times 125 + 1 \times 625 + 2 \times 3125$

.

When it comes to decompression , $H(R)$ and the position of decryption k must be provided as input. Knowing them and applying equation (4) we can decrypt the nucleotide at position $\mathbf{k}, \mathbf{a_k}$. For instance, if we want to decrypt the nucleotide at position $\mathbf{2}, \mathbf{k} = \mathbf{2}$ (the third nucleotide from the beginning) equation (4) must be applied: $\mathbf{f(a_2)} = \left({}^{\mathbf{H(R)}}/_{\mathbf{5^2}}\right)\mathbf{mod5} = \left({}^{7309}/_{25}\right)\mathbf{mod5} = \mathbf{292mod5} = \mathbf{2}.$ Knowning that $\mathbf{f(a_2)} = \mathbf{2}$ we get that $\mathbf{a_2}$ =T (Thymine), since $\mathbf{f(T)} \rightarrow \mathbf{2}\ (\mathbf{f(T)} = \mathbf{2})$.

Compressing (hashing) the short read $\mathbf{R} = \mathbf{CATGAT}$ into single integer 7309 of 4 bytes results into 2 bytes of memory saving. This happens because one byte per nucleotide is required if the data is stored in raw format (as array of characters). The read $\mathbf{R}$ which we consider contains 6 bp what means that 6 bytes are required to be stored as array of letters. Translating the read into single integer, only 4 bytes are required, that results in 2 bytes of memory saving.

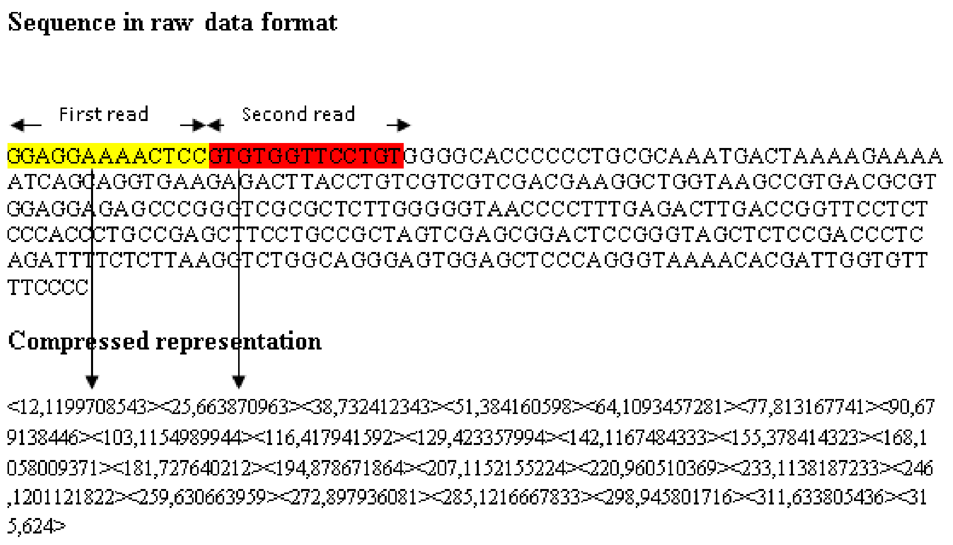
However we should be aware that in computational domain there is always a limitation upon the value which can be stored into single variable. Theoretically 2.147.483.647 is the maximum which can be stored into single variable of integer type, but our tests shown that the maximum value with which we can work safety is 1.220.703.124, which corresponds to the hash of chain of 13 cytosines.

To be able to compress big genomic data, we must split the sample into short reads. Rather than reads of random size, the size of each short read is carefully selected, such as the hash of the read is the one which is closest to 1.220.703.124. Applying this concept, in average, we can zip reads of 13 base pairs into integers of 4 bytes. However, information regards offsets of short reads must be also tracked if we want to be able to decompress data later. This information can be presented in form of end positions of the reads.

To illustrate what happens, we will take Apple dimple fruit viroid (ENA ID: X99487.1) as a sample, Fig. 1. This sequence contains 306 base pairs, i.e. 306 bytes are required to store in raw data format, Fig. 1. Applying our methodology this sequence is compressed down to list of 50 integers, i.e. $\mathbf{50 \times 4 = 200}$ bytes are required that results in 34.6 % (106 bytes) of memory saving, Fig. 1.

Each tuple <> on Fig. 1 stands for concrete read, such as: the first integer represents the end position of the read; the second integer represents its hash. For instance, <25,663870963> represents the second read in the offset 13-25 (Fig. 1) which is hashed in 663.870.963 (Fig. 1). The beginning of the read 13 can be recomputed as the end position of the previous read plus one (12+1=13), Fig. 1.

As explained and shown on Fig. 1, each hash being computed is the one which is closest to 1.220.703.124: 1.199.708.543 for the first read, 663.870.963 for the second….etc. and the reason for this is the our general intention to maximize the overall compression gain.



Sequence in raw data format

```
GGAGGAAAACTCCGTGTGGTTCCTGTGGGGCACCCCCCTGCGCAAATGACTAAAAGAAAA
ATCAGCAGGTGAAGAGACTTACCTGTCGTCGTCGACGAAGGCTGGTAAGCCGTGACGCGT
GGAGGAGAGCCCGGGTCGCGCTCTTGGGGGTAACCCCTTTGAGACTTGACCGGTTCCTCT
CCCACCCTGCCGAGCTTCCTGCCGCTAGTCGAGCGGACTCCGGGTAGCTCTCCGACCCTC
AGATTTTCTCTTAAGGTCTGGCAGGGAGTGGAGCTCCCAGGGTAAAACACGATTGGTGTT
TTCCCC
```

Compressed representation

```
<12,1199708543><25,663870963><38,732412343><51,384160598><64,1093457281><77,813167741><90,67
9138446><103,1154989944><116,417941592><129,423357994><142,1167484333><155,378414323><168,1
058009371><181,727640212><194,878671864><207,1152155224><220,960510369><233,1138187233><246
,1201121822><259,630663959><272,897936081><285,1216667833><298,945801716><311,633805436><31
5,624>
```

FIG. 1. Sample in raw and compressed data format

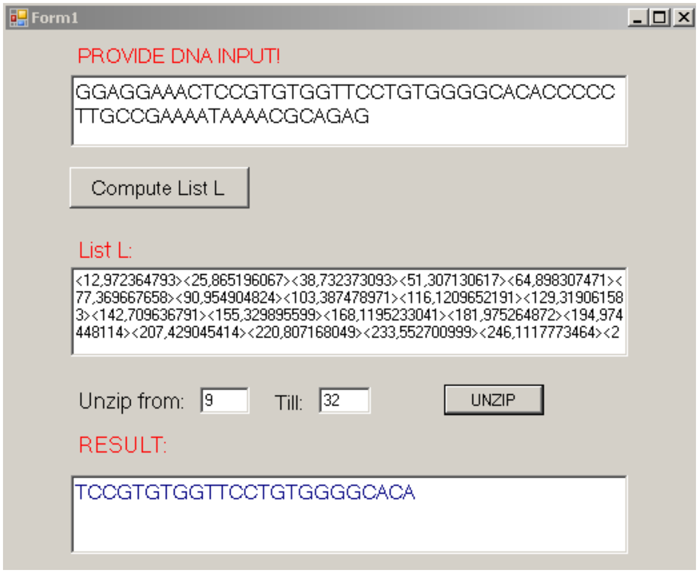
RESULTS AND DISCUSSIONS

Genomic data compression program was developed, Fig. 2. The program was written in Microsoft Visual C # 2008 Express Editions and it compresses data according to the explained methodology. One is expected to provide raw genomic data in the upper text control which is compressed (hashed) and printed as a list of integers in the middle text control, Fig. 2. Given that the range of decompression is provided Fig. 2, data decompression is also possible by using the information form the list. This is done by applying the decompression equation (4) for all positions that are in the range of decompression. The program prints also details about the overall compression gain.

The program was tested on 6 Citrus dwarfing viroid samples which were retrieved from the European Nucleotide Archive, Table 1. Acer Aspire 5507Z computer with Genuine Intel CPU at 1.73 GHz and 2 GB of RAM was used in all tests.

These samples contain 291-295 base pairs and therefore 291-295 bytes are required to store them in raw data format as array of characters, Table 1, Fig. 3. Applying our methodology all these samples were compressed down to lists of 48 integers or 192 bytes were required, Table 1, Fig. 3. Compression gain of more than 34 % was measured in each test Table 1, Fig. 4 and it took only a few milliseconds to decompress any DNA motif applying equation (4).

It may be true that current compression techniques achieve better compression gains, but none of them achieves any compression gain without the use of referent sequence, i.e. the compression gain equals 0% if reference is not exploited. Even though that our methodology does not depend of any kind of reference sequence, we got more than 34% of compression gain, that is the prime advantage of our methodology over the others.



PROVIDE DNA INPUT!

```
GGAGGAAACTCCGTGTGGTTCCTGTGGGGCACACCCCC
TTGCCGAAAATAAAACGCAGAG
```

List L:

```
<12,972364793><25,865196067><38,732373093><51,307130617><64,898307471><
77,369667658><90,954904824><103,387478971><116,1209652191><129,31906158
3><142,709636791><155,329895599><168,1195233041><181,975264872><194,974
448114><207,429045414><220,807168049><233,552700999><246,1117773464><2
```

Unzip from: 9 Till: 32

RESULT:

```
TCCGTGTGGTTCCTGTGGGGCACA
```

FIG. 2. Program screenshot

TABLE 1. Results before and after compression

Sequence	ENA ID	Length (bp)	**Size in native form (B)**	**Size after compression (B)**	**Saving (B)**	**Saving (%)**
Citrus dwarfing viroid, complete genome.	JF521494.1	291	**291**	**192**	**99**	**34,0206185 6**
Citrus dwarfing viroid isolate lot1-1/2-19, complete genome.	KM214215.1	293	**293**	**192**	**101**	**34,4709897 6**
Citrus dwarfing viroid isolate LS-4, complete genome.	JF812069.1	293	**293**	**192**	**101**	**34,4709897 6**
Citrus viroid III isolate 054-4uy, complete genome.	AY514448.1	294	**294**	**192**	**102**	**34,6938775 5**
Citrus dwarfing viroid isolate H16-9, complete genome.	JF970265.1	294	**294**	**192**	**102**	**34,6938775 5**
Citrus dwarfing viroid clone Hb-C1-2, complete genome.	JF742602.1	295	**295**	**192**	**103**	**34,9152542 4**

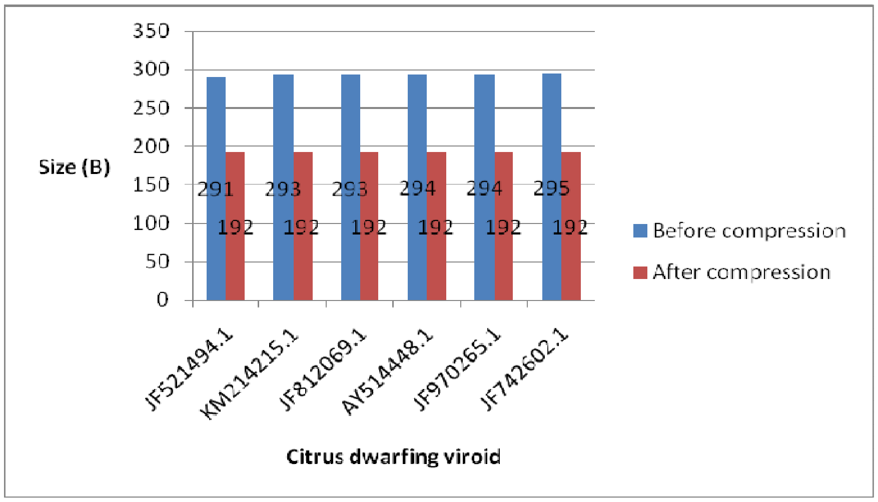


FIG. 3. Comparing sequence size (before and after compression)

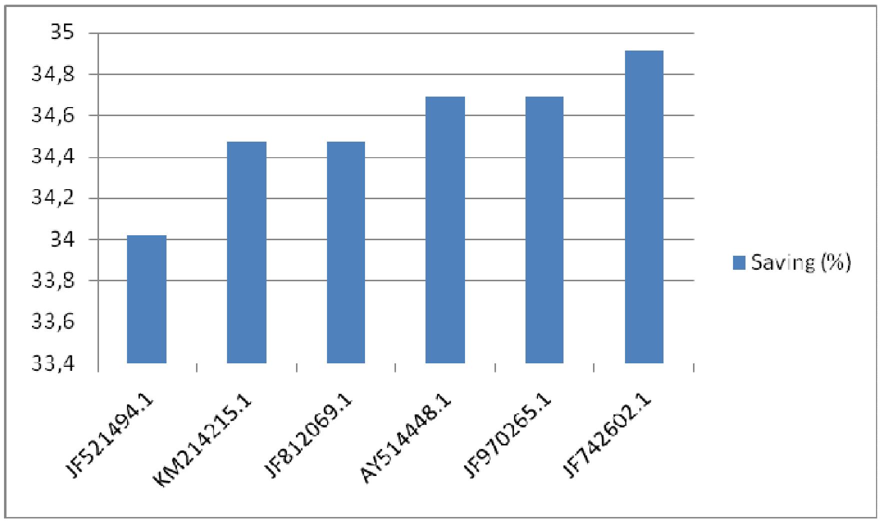


FIG. 4. Percentage of saving

CONCLUSIONS

In this paper we present a new genomic data compression methodology that does not depend on reference sequence. We used data hashing for this purpose, changing the radix in Ranker's formula to 5 that in turns allows us to apply unidirectional formula for data decompression. We hash genomic data into list of integers, using as much as possible of the available capacity of variables. Results indicate that even without reference, more than 34% of compression gain is possible.

REFERENCES

- Brandon MC., Wallace DC., Baldi P. 2009. Data structures and compression algorithms for genomic sequence data. Bioinformatics25(14):1731–1738.
- Christley S., Lu Y., Li C, Xie X. 2008. Human genomes as email attachments. Bioinformatics25(2):274–275.
- Deorowicz S., Grabowski S. 2011. Robust relative compression of genomes with random access. Bioinformatics27(21):2979–2986.
- Golomb S. 1966. Run-length encodings. IEEE transactions on information theory12(3):399–401.
- Huffman DA. 1952. A method for the construction of minimum-redundancy codes, pp. 1098–1101. In: Proceedings of the IRE.
- Pavlichin DS., Weissman T., Yona G. 2013.The human genome contracts again. Bioinformatics29(17):2199–2202.
- Pinho AJ., Pratas D., Garcia SP. 2011. GReEn: a tool for efficient compression of genome resequencing data. Nucleic acids research40(4):e27–e27.
- Reneker J., Shyu CR. 2005.Refined repetitive sequence searches utilizing a fast hash function and cross species information retrievals. BMC bioinformatics6(1):111.
- Tembe W., Lowey J., Suh E. 2010. G-SQZ: Compact encoding of genomic sequence and quality data. Bioinformatics26(17):2192–2194.
- Wang C., Zhang D. 2011. A novel compression tool for efficient storage of genome resequencing data. Nucleic acids research39(7):e45–e45.