A New approximate matching compression algorithm for DNA sequences

a,b J.M. Lázaro-Guevara*, c K.M. Garrido
jose.lazaroguevara@utah.edu, dra.kmgarrido@gmail.com

a Department of Human Genetics, University of Utah, Salt Lake City, USA
b Medical Sciences, University of San Carlos, Guatemala, Guatemala
c Paediatrics Department, Guatemalan Social Secure, Guatemala, Guatemala

* Corresponding Author.

1. Abstract: Undeveloped countries like Guatemala, where access to high-speed internet connections is limited, downloading and sharing Biological information of thousands of Mega Bits is a huge problem for the beginning and development of Bioinformatics. Based on that information is an urgent necessity to find a better way to share this biological data. There is when the compression algorithms become relevant. With all this information in mind, born the idea of creating a new algorithm using redundancy and approximate selection. Methods: Using the probability given by the transition matrix of the three-word tuple and relative frequencies. Calculating the relative and total frequencies given by the permutation formula (nr) and compressing 6 bits of information into 1 implementing the ASCII table code (0...255 characters, 28), using clusters of 102 DNA bases compacted into 17 string sequences. For decompressing, the inverse process must be done, except that the triplets must be selected randomly (or use a matrix dictionary, 4102). Conclusion: The compression algorithm has a better compression ratio than LZW and Huffman’s algorithm. However, the time needed for decompressing makes this algorithm incompatible for massive data. The functionality as MD5sum need more research but is a promising helpful tool for DNA checking.

Keywords— DNA compression, DNA analysis, compression algorithm.
2. Background:

Undeveloped countries like Guatemala, where the access to high speed internet connections is limited downloading and sharing Biological information of thousands of Mega Bits is a huge problem for the beginning and development of Bioinformatics.

Based on that information is an urgent necessity to find a better way to share this biological data. There are two different ways to solve this problem, the first one improving the velocity at which the data is transferred (improving the internet connection), however this is not affordable in a country like Guatemala, the second way is to decrease the amount of data transferred.[1][2]

To accomplish this goal on reducing the data, the most common form is compressing the information, and for compressing the information there are several ways to do it.

But this compression forms depends on the kind of information or data to be compressed, the most reliable methods like Huffman Algorithm, cannot be used in the DNA compression because this methods based on relative frequencies can involve some data lost (~3%)[1]. A second way is using a more reliable method like LZW (Lempel-Ziv-Welch) where there is no lost on data but need the use of license to generate the compressed code based on patents, and involves the use of a matrix dictionary for each code compressed[2][3].

With all this information in mind, born the idea of creating a new algorithm based on redundancy and approximate probability given by the transition matrix of the three word tuple and relative frequencies [3][4](figure 1).
3. Methods:  
3.1 Compressing Method:  

First Step: Get a DNA sequence, and split it in subsequence of 102 bases

```
g A T a c A
```

Second Step: transform the bases to a numeric correspondent (a=1, c=2, g=3, t=4).

```
3 1 4 1 2 1
```

Third Step: It is necessary to divide in 102 bases because is needed a number than can be divided by 6 and 3, and also superior a 100 to improve the compression ratio, based and that 1 character will represent a 6 DNA bases.

- \( n \times n \times \ldots (r \text{ times}) = n^r \), \( r \) times, we have 4 possibilities (a,c,g,t), and selecting 3 each time, based on the process of adding the first one and subtracting to that sum the third value (eg. \((3+1)-4=0\)), we only have 7 possible values that will be from \(-2\) to \(7\). This model of adding and subtracting was used to obtain an array of 10 elements in a normal distribution \((1/64,3/64,6/64,10/64,12/64)\) (table 1).

- It is necessary to group 64 possible combination of values in only 10 combinations to use the ASCII mode (range from 0 to 255), in table I it is possible to observe that exist two negative values (-2,-1), and it is necessary to replace them in the second value by the (8 = -1, and 9 = -2), to use the concatenation of two characters as one number.

- After obtaining 2 characters and concatenating it, it is necessary to add 63 to this value, to avoid characters used by the core system. The range of values after concatenation will go from -29 to 79. So after adding 63 to this values the range in ASCII will be from 34 to 142. Table II.

```
0 2
```

```
63+02=65
Ascii =”A”
```
Example 1:

**DNA code: (102 characters)**

```
ataacctacaagagcaggtataacgagccccgagcttgtaagcgccagctatatatacaatacgtttcactggaacagctaccaggatgttaacaaagttcataagg
```

**Compressed Version: (17 characters)**

```
h[Tk`Xih*l*oi`?]N4
```

e. Checking numbers 1 and 2: This numbers allow to calculate the original 102 bases compacted into 17 characters from 34 numbers. Number 1 is obtained by adding and subtracting the product of the 34 calculated number. (eg. \((3+1-4)-(1+2-1)\)....+....). In case of example 1 is **11**.

Number 2 is obtained by a similar process using multiplication and subtracting process calculated number of the 34 by the formula \(2*(1st*3rd)-(3*2nd)\) (eg. \((2*(3*4)-(3*1))-(2*(1*1)-(3*2))....+....\). **In case of example 1 is -3.**

**Compacted code of example 1:** `h[Tk`Xih*l*oi`?]N4!11!-3`

f. Line 1 header. This is the first line of the compacted code and contains the absolute frequency of each one of the 64 possible combinations for future checking in the decompressing process. (eg. 62-95-94-59-06-34-59-46-27-62-66-60-59-96-43-63-36-16-62-56-43-64-26-44)

**3.2 Decompressing Method:**

a. Read Line 1, introduce the data into a linear vector with relative frequencies of each triplet for further comparison against the decompressed code and the original code, and improving the velocity of the change based on the change in the total amount of probability of each possible term.

b. Obtain the compacted code (eg `h[Tk`Xih*l*oi`?]N4!11!-3`) , split into three different terms using the symbol “!”, as separation indicator. **Term 1**: 17 character code.  

   **Term 2**: number 1 and **Term 3**: number 2.

c. Using the reverse method of part d in compression method, convert the ascii code into a two digit value (eg. Ascii = “A”, number from asci = 65), subtracts 63 to that number (eg. 65 - 63=02), using the modulus and the integer division separate into 2 numbers (02 = 0, 2). If number two is 8 or 9 change into the negative portion (eg. \(-29 = -2, -2\)).

d. From the 17 character code, after changing in the 2 digits way, it will be converted into a 34 digits code.
e. Now is necessary to change 34 digits into 102 characters, but it is no possible to use the inverted process on step b on the codifying part because each number has may possibilities from 3 to 12 possible options, only numbers -2 and 7, has one direct transcription, and number -2 and 7, where selected based on the most common triplet of bases repeated in 1000 trials of codifying sequences.

f. Because there are 102 possible DNA bases in this code and there are 4 possible letter for sequence and the order it is important, it is necessary to calculate the permutation \( 4 \text{ P } 102 \), that it is equivalent to count 34 digits and 64 possible values obtained from the triplets, \( 64 \text{ P } 34 \).

But this value is so huge \( (2.57 \times 10^6) \), but it is limited as shown in TABLE III, by the number of probabilities from 3 to 12, in 8 possible numbers.

For that reason is necessary the checking numbers 1 and 2, there is only one sequence of 34 digits (17 characters) for each pair of checking numbers (look at probability table at the bottom of TABLE III, figure 1).

### 3.3 Measuring Accuracy on decompression and data loss:

For obtaining the original sequence of DNA it is necessary to obtain the permutations, and compare it to the checking numbers, the problem is that this take a lot of time so the best solution is to import a common Tables Comparing Dictionary, the disadvantage of this solution is the big size of the dictionary.

The measuring of accuracy is related to the number of cycles permuted, to obtain the original DNA sequence is necessary compare checking numbers 1 and 2, until obtaining the original sequence, more the numbers of permutation higher the probability on obtaining the 102 original DNA bases per permutation. (TABLE IV)

| Original | 1 | 4 | 1 | 1 | 3 | 3 | 1 | 2 | 1 | 2 | 1 | 4 |
|----------|---|---|---|---|---|---|---|---|---|---|---|---|
| Decompressed | 1 | 4 | 1 | 1 | 2 | 2 | 1 | 2 | 1 | 1 | 1 | 3 |
| Match | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |

| Sum | 8 | 67% |

The accuracy will be measured comparing simultaneously the sequences (Compressed = original), and measuring the differences or sections with discrepancy it is obtained the accuracy proportion (e.g.)
4. Results:

After several trials, and running multiple simulations in R (Statistical Programming Language), on how this compression and decompression system works. (Figure 2, figure 3) It is obtained a compression ratio based on data weight in kilobytes (Kb), using a 10,000 bases (A, C, G, T) section of DNA extracted from NC_000913.1 (E.coli) with read.Genbank function.

The data extracted and printed in a text file (.TXT), uses 20,100 bytes, the data compacted in the same archive system uses only 2,567.

The definition of the compression ratio is \( \frac{|O|}{2|I|} \), where \(|I|\) is number of bases in the input DNA sequence and \(|O|\) is the length (number of bits) of the output sequence, the general compression ratio is 8:1 (7.84), the inline code compression ratio is 5:1 (102 DNA bases are equivalents to 17 ASCII codes plus 2 checking numbers). The compression ratios, as well as the multiple permutation system, are presented in Figure 1.

The compressing and decompressing time is related to the number of cycles (~102 permutations), using a quad-core 2.3 Ghz processor as standard, it takes 38 milliseconds to complete one cycle, but because it is using a iteration systems (for and while commands), taking on count the RAM and processor bus transfer speed is necessary to add 15% on delay each 1000 cycles.

The accuracy on decoding is related directly to the number of iterations and permutations (figure 2), whit only 100 to 2,500 accuracy varies from 50 to 70%, but this accuracy is counted only in a line of decoding (102 DNA bases), based on time of processing, after 7,500 cycles accuracy increase in exponential, in range from 80 to 91%, reaching the limit in cycles around 50,000, because Random Access Memory limiting, so it is unknown if the accuracy will reach 100% in the estimated 450,000 cycles (RAM memory insufficient for this calculation on normal PC 8 Gigabytes).

After running a sequence of 1020 DNA bases, using 5000 cycles, the analysis took 9.5 hours, and accuracy obtained was 63%. A combination dictionary for improving the speed and accuracy was created with 120,000 permutations and speed increase by 5%, for creating the complete dictionary with all possible permutations (2.57 x 10E61), certainly will improve accuracy and speed of decoding, but was not possible to create it with the R programming language, based on the RAM size.
5. Discussion:
This algorithm have demonstrated that the general compression ratio obtained (8:1) is better that traditional compression codes (eg LZW, Huffmans, Biocompress, Gencompress, FASTA), but the time needed for decompression makes no worthy the usage of this compression method by using multiple permutations, it will improve using a dictionary of sequences preinstalled in the coding and decoding software.

It was not possible to probe the complete accuracy, assuring no data lose in the decompression system after 50,000 cycles accuracy only reach 86%, adding some modifications like improving the random sampling permutations it was able to obtain 91%. For that reason is necessary to do more changes on the code and try to reach 100% accuracy on selected sampling instead of random sampling on a better equipment with more processor speed and RAM.

Besides the time consumption for decompressing, this algorithm can also be used in the compression form as a checksum like the MD5sum system for DNA codes. MD5 algorithm is the mainstream for the cryptographic check and file checkup, the hash function when MD5 is applied returns a unique check number that allow knowing that this file is uncorrupted, this is an important problem in compressing method when data lose exist, and is necessary to be sure of that DNA is not corrupted.

So it is necessary to continue in researching for improving it.

6. Conclusion:

- The compression algorithm has better compression ratio than LZW and huffmans algorithm.

- The time need for decompressing make not useful this algorithm for huge amount of data.

- The functionality as MD5sum need more research, but is a promising useful tool for DNA checking.
7. Reference:

[1]. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. C 1990. Second Edition

[2] Welch, T.A., "A Technique for High-Performance Data Compression," in Computer, vol.17, no.6, pp.8-19, June 1984 doi: 10.1109/MC.1984.1659158

[3] A compression algorithm for DNA sequences DOI: 10.1109/51.940049

[4] Hsi-Yang Fritz, Markus et al. “Efficient Storage of High Throughput DNA Sequencing Data Using Reference-Based Compression.” Genome Research21.5 (2011): 734–740. PMC. Web. 3 Dec. 2015.

[5] Zhao Yong-Xia; Zhen Ge, "MD5 Research," in Multimedia and Information Technology (MMIT), 2010 Second International Conference on, vol.2, no., pp.271-273, 24-25 April 2010 doi: 10.1109/MMIT.2010.186
| Replace | Combination Values | Distribution | Combination Values | Distribution |
|---------|--------------------|--------------|--------------------|--------------|
| 9       | -2 1 1 4           | 1/64         | 3 2 4 3           | 1/64         |
| 8       | -1 1 2 4           | 3/64         | 3 1 4 2           | 3/64         |
| 8       | -1 2 1 4           | 1/3          | 3 3 4 4           | 1/3          |
| 8       | -1 1 1 3           | 1/64         | 3 4 2 3           | 1/64         |
| 0       | 2 2 4             | 1/64         | 3 2 2 1           | 1/64         |
| 0       | 1 2 3             | 3/64         | 3 3 2 2           | 3/64         |
| 0       | 2 1 3             | 1/3          | 3 4 1 2           | 1/3          |
| 0       | 1 1 2             | 3/64         | 3 3 1 1           | 3/64         |
| 0       | 3 1 4             | 1/64         | 3 4 3 4           | 1/64         |
| 0       | 1 3 4             | 3/64         | 3 2 3 2           | 3/64         |
| 1       | 1 4 4             | 1/3          | 3 1 3 1           | 1/3          |
| 1       | 2 2 3             | 3/64         | 3 3 3 3           | 3/64         |
| 1       | 1 2 2             |              | 4 4 4 4           |              |
| 1       | 3 2 4             |              | 4 2 4 2           |              |
| 1       | 4 1 4             |              | 4 1 4 1           |              |
| 1       | 2 1 2             |              | 4 3 4 3           |              |
| 1       | 1 1 1             |              | 4 4 2 2           |              |
| 1       | 3 1 3             |              | 4 3 2 1           |              |
| 1       | 2 3 4             |              | 4 4 1 1           |              |
| 1       | 1 3 3             |              | 4 4 3 3           |              |
| 2       | 2 4 4             |              | 4 2 3 1           |              |
| 2       | 1 4 3             |              | 4 3 3 2           |              |
| 2       | 4 2 4             |              | 5 4 4 3           |              |
| 2       | 2 2 2             |              | 5 2 4 1           |              |
| 2       | 1 2 1             |              | 5 3 4 2           |              |
| 2       | 3 2 3             |              | 5 4 2 1           |              |
| 2       | 4 1 3             |              | 5 4 3 2           |              |
| 2       | 2 1 1             |              | 5 3 3 1           |              |
| 2       | 3 1 2             |              | 6 4 4 2           |              |
| 2       | 2 3 3             |              | 6 3 4 1           |              |
| 2       | 1 3 2             |              | 6 4 3 1           |              |
| 2       | 3 3 4             |              | 7 4 4 1           |              |
TABLE II

ASCII SEQUENCE FOR CHAR AND HEXADECIMAL COMPRESION

A) Primary ASCII code (27)

| Dec | Ha | Oct | Char  | Dec | Ha | Oct | Char  | Dec | Ha | Oct | Char  |
|-----|----|-----|-------|-----|----|-----|-------|-----|----|-----|-------|
| 0   | 0  | 0   | null  | 32  | 20 | 0   | space | 64  | 40 | 0   | add   |
| 1   | 001 | 011 | start of heading | 33  | 21 | 011 | start of control A | 65  | 41 | 011 | start of control D |
| 2   | 002 | 010 | start of text | 34  | 22 | 010 | start of text | 66  | 42 | 010 | start of text |
| 3   | 003 | 011 | start of text | 35  | 23 | 011 | start of text | 67  | 43 | 010 | start of text |
| 4   | 004 | 100 | end of transmission | 36  | 24 | 100 | end of transmission | 68  | 44 | 100 | end of transmission |
| 5   | 005 | 101 | enq | 37  | 25 | 101 | enq | 69  | 45 | 101 | enq |
| 6   | 006 | ACK | acknowledge | 38  | 26 | 006 |ACK | 70  | 46 | 006 | ACK |
| 7   | 007 | BEL | bell | 39  | 27 | 007 |Bell | 71  | 47 | 007 | Bell |
| 8   | 010 | BS | (backspace) | 40  | 28 | 010 | BS | 72  | 48 | 110 | x170 |
| 9   | 011 | HT | (horizontal tab) | 41  | 29 | 011 | HT | 73  | 49 | 111 | x171 |
| 10  | A  | LF | (line feed, new line) | 42  | 2A | 012 | LF | 74  | 4A | 112 | x172 |
| 11  | B  | VT | (vertical tab) | 43  | 2B | 013 | VT | 75  | 4B | 113 | x173 |
| 12  | C  | FF | (form feed, new page) | 44  | 2C | 014 | FF | 76  | 4C | 114 | x174 |
| 13  | D  | CR | (carriage return) | 45  | 2D | 015 | CR | 77  | 4D | 115 | x175 |
| 14  | E  | SI | (shift in) | 46  | 2E | 016 | SI | 78  | 4E | 116 | x176 |
| 15  | F  | DLE | (data link escape) | 47  | 2F | 017 | DLE | 79  | 4F | 117 | x177 |
| 16  | 020 | DC1 | (device control 1) | 48  | 30 | 020 | DC1 | 80  | 50 | 120 | x200 |
| 17  | 021 | DC2 | (device control 2) | 49  | 31 | 021 | DC2 | 81  | 51 | 121 | x201 |
| 18  | 022 | DC3 | (device control 3) | 50  | 32 | 022 | DC3 | 82  | 52 | 122 | x202 |
| 19  | 023 | DC4 | (device control 4) | 51  | 33 | 023 | DC4 | 83  | 53 | 123 | x203 |
| 20  | 024 | NAK | (negative acknowledge) | 52  | 34 | 024 | NAK | 84  | 54 | 124 | x204 |
| 21  | 025 | SYN | (synchronous idle) | 53  | 35 | 025 | SYN | 85  | 55 | 125 | x205 |
| 22  | 026 | ETX | (end of transmission) | 54  | 36 | 026 | ETX | 86  | 56 | 126 | x206 |
| 23  | 027 | EOT | (end of output) | 55  | 37 | 027 | EOT | 87  | 57 | 127 | x207 |
| 24  | 028 | ENQ | (enquiry) | 56  | 38 | 028 | ENQ | 88  | 58 | 128 | x208 |
| 25  | 029 | ACK | (acknowledge) | 57  | 39 | 029 | ACK | 89  | 59 | 129 | x209 |
| 26  | 030 | BEL | (bell) | 58  | 3A | 030 | BEL | 90  | 5A | 130 | x20A |
| 27  | 031 | BS | (backspace) | 59  | 3B | 031 | BS | 91  | 5B | 131 | x20B |
| 28  | 032 | HT | (horizontal tab) | 60  | 3C | 032 | HT | 92  | 5C | 132 | x20C |
| 29  | 033 | LF | (line feed, new line) | 61  | 3D | 033 | LF | 93  | 5D | 133 | x20D |
| 30  | 034 | VT | (vertical tab) | 62  | 3E | 034 | VT | 94  | 5E | 134 | x20E |
| 31  | 035 | FF | (form feed, new page) | 63  | 3F | 035 | FF | 95  | 5F | 135 | x20F |

B) Extended ASCII Code (28)

| Dec | Ha | Oct | Char  | Dec | Ha | Oct | Char  | Dec | Ha | Oct | Char  |
|-----|----|-----|-------|-----|----|-----|-------|-----|----|-----|-------|
| 128  | C | 144 | é | 160 | á | 176 | 24B | 224 | a | 240 | é |
| 129  | ã | 145 | æ | 161 | ï | 177 | 252 | 225 | ß | 241 | ã |
| 130  | é | 146 | À | 162 | õ | 178 | 259 | 226 | 242 | é |
| 131  | à | 147 | ó | 163 | ë | 179 | 231 | 227 | n | 243 | à |
| 132  | à | 148 | õ | 164 | À | 180 | 212 | 228 | ã | 244 | à |
| 133  | à | 149 | ô | 165 | Ñ | 181 | 213 | 229 | õ | 245 | à |
| 134  | à | 150 | ù | 166 | ì | 182 | 214 | 230 | ì | 246 | à |
| 135  | ã | 151 | ù | 167 | ÿ | 183 | 215 | 231 | ÿ | 247 | ã |
| 136  | à | 152 | ÿ | 168 | Ô | 184 | 200 | 232 | ñ | 248 | à |
| 137  | ë | 153 | Ô | 169 | Ó | 185 | 201 | 233 | Ó | 249 | ë |
| 138  | à | 154 | Ù | 170 | ñ | 186 | 202 | 234 | ñ | 250 | à |
| 139  | à | 155 | Ù | 171 | ñ | 187 | 203 | 235 | ñ | 251 | à |
| 140  | à | 156 | ê | 172 | ê | 188 | 204 | 236 | ë | 252 | à |
| 141  | ë | 157 | ò | 173 | ò | 189 | 205 | 237 | ò | 253 | ê |
| 142  | õ | 158 | Ô | 174 | À | 190 | 206 | 238 | ñ | 254 | õ |
| 143  | Á | 159 | j | 175 | À | 191 | 207 | 239 | À | 255 | Á |
## TABLE III
Sequence of DNA of Example 1, possible results on checking numbers 1 and 2 after permutations.

|   | 1 | 4 | 1 | 2 | -1 | 2 | 1 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 1 | 10 | -2 | 1 | 4 | -1 | 4 | 2 | 3 | 3 | 0 | 0 | 3 | 0 | 1 | 5 | -1 | 1 |
|---|---|---|---|---|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 444 | 144 | 244 | 124 | 244 | 144 | 444 | 444 | 444 | 444 | 144 | 244 | 243 | 243 | 243 | 243 | 144 | 244 | 124 | 444 | 244 | 243 | 243 | 243 | 243 | 144 | 144 | 144 | 144 | 244 | 243 | 243 | 243 | 144 | 144 | 144 |
| 2 | 242 | 223 | 143 | 214 | 143 | 223 | 423 | 423 | 423 | 423 | 143 | 223 | 242 | 242 | 242 | 242 | 143 | 242 | 242 | 423 | 242 | 242 | 242 | 242 | 143 | 143 | 143 | 143 | 242 | 242 | 242 | 242 | 143 | 143 | 143 |
| 3 | 414 | 122 | 424 | 113 | 424 | 122 | 414 | 141 | 141 | 141 | 424 | 141 | 223 | 122 | 122 | 122 | 223 | 122 | 141 | 113 | 141 | 424 | 344 | 344 | 213 | 213 | 344 | 113 | 113 | 213 | 344 | 213 | 213 | 344 | 113 | 113 |
| 4 | 343 | 324 | 222 | 222 | 324 | 434 | 343 | 423 | 423 | 423 | 423 | 343 | 423 | 222 | 343 | 324 | 324 | 112 | 324 | 343 | 343 | 343 | 343 | 343 | 343 | 343 | 343 | 343 | 343 | 343 | 343 | 343 | 343 | 343 | 343 | 343 |
| 5 | 422 | 414 | 121 | 121 | 414 | 422 | 422 | 422 | 422 | 121 | 422 | 414 | 414 | 414 | 414 | 414 | 414 | 414 | 121 | 222 | 222 | 222 | 222 | 222 | 222 | 222 | 222 | 222 | 222 | 222 | 222 | 222 | 222 | 222 | 222 | 222 |
| 6 | 321 | 212 | 323 | 323 | 212 | 321 | 321 | 321 | 321 | 321 | 321 | 321 | 212 | 212 | 212 | 212 | 212 | 212 | 321 | 321 | 321 | 321 | 321 | 321 | 321 | 321 | 321 | 321 | 321 | 321 | 321 | 321 | 321 | 321 | 321 | 321 |
| 7 | 411 | 111 | 413 | 413 | 111 | 411 | 411 | 412 | 412 | 413 | 411 | 413 | 411 | 111 | 111 | 111 | 111 | 111 | 411 | 411 | 413 | 413 | 412 | 412 | 412 | 412 | 412 | 412 | 412 | 412 | 412 | 412 | 412 | 412 | 412 | 412 |
| 8 | 433 | 313 | 211 | 211 | 313 | 433 | 433 | 311 | 311 | 211 | 433 | 211 | 433 | 313 | 313 | 313 | 313 | 313 | 433 | 433 | 433 | 433 | 433 | 433 | 433 | 433 | 433 | 433 | 433 | 433 | 433 | 433 | 433 | 433 | 433 | 433 |
| 9 | 231 | 234 | 312 | 312 | 234 | 231 | 231 | 231 | 231 | 231 | 231 | 231 | 234 | 234 | 234 | 234 | 234 | 234 | 231 | 231 | 231 | 231 | 231 | 231 | 231 | 231 | 231 | 231 | 231 | 231 | 231 | 231 | 231 | 231 | 231 | 231 |
| 10 | 332 | 133 | 233 | 233 | 133 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 | 332 |
| 11 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 |
| 12 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 | 334 |

- **Table Note:** The table above represents the sequence of DNA of Example 1, along with possible results from checking numbers 1 and 2 after permutations.
### TABLE IV

Sequence of DNA of Example 1, checking numbers 1 and 2.

|   | 4 | 1 | 2 | -1 | 2 | 1 | 4 | 4 | 3 | 2 | 5 | 4 | 2 | 4 | 1 | 1 | 0 | -2 | 1 | 4 | -1 | 4 | 2 | 3 | 0 | 0 | 3 | 0 | 1 | 5 | -1 | 1 |
|---|---|---|---|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 20 | -4 | 4 | 2 | 4 | -4 | 20 | 20 | 0 | 0 | 4 | 12 | 20 | 4 | 20 | -4 | -4 | 10 | 5 | -4 | 20 | 2 | 20 | 4 | 0 | 0 | 10 | 10 | 0 | 10 | -4 | 12 | 2 | -4 |
| 2 | -4 | 6 | -6 | 13 | -6 | 6 | -4 | -4 | -8 | -8 | -6 | -8 | -6 | -4 | 6 | 6 | 0 | 6 | -4 | 13 | -4 | -6 | -8 | -8 | 0 | 0 | -8 | 0 | 6 | -8 | 13 | 6 |
| 3 | -10 | -2 | 26 | 3 | 26 | -2 | -10 | -10 | 12 | 12 | 26 | 0 | -10 | 26 | -10 | -2 | -2 | 9 | -2 | -10 | 3 | -10 | 26 | 12 | 12 | 9 | 9 | 12 | 9 | -2 | 0 | 3 | -2 |
| 4 | 6 | 18 | 2 | 2 | 18 | 6 | 18 | 18 | 2 | 2 | 6 | 2 | 6 | 18 | 18 | 1 | 18 | 6 | 6 | 2 | 18 | 18 | 1 | 1 | 18 | 1 | 18 | 2 | 18 |
| 5 | 10 | 29 | -4 | -4 | 29 | 10 | 10 | -2 | -2 | -4 | 7 | 10 | -4 | 10 | 29 | 29 | 21 | 29 | 10 | 10 | -4 | -2 | -2 | 21 | 21 | -2 | 21 | 29 | 7 | 29 |
| 6 | 0 | 5 | 12 | 12 | 5 | 0 | 0 | 6 | 6 | 12 | -3 | 0 | 12 | 0 | 5 | 5 | -1 | 5 | 0 | 0 | 12 | 6 | 6 | -1 | -1 | 6 | -1 | 5 | -3 | 5 |
| 7 | 5 | -1 | 21 | 21 | -1 | 5 | 5 | 13 | 13 | 21 | 5 | 21 | 5 | -1 | -1 | -1 | -1 | 5 | 5 | 21 | 13 | 13 | 13 | -1 | -1 | -1 | 5 | -3 | 5 |
| 8 | 15 | 15 | 1 | 1 | 15 | 15 | 15 | 3 | 3 | 1 | 15 | 1 | 15 | 15 | 15 | 15 | 15 | 15 | 1 | 1 | 3 | 3 | 15 | 15 | 15 | 15 | 3 | 15 | 15 |
| 9 | -5 | 7 | 9 | 9 | 7 | -5 | -5 | 23 | 23 | 9 | -5 | 9 | -5 | 7 | 7 | 7 | -5 | -5 | 9 | 23 | 23 | 23 | 7 | 7 |
| 10 | 3 | -3 | 3 | 3 | -3 | 3 | 3 | -1 | -1 | 3 | 3 | 3 | 3 | -3 | -3 | -3 | -3 | 3 | 3 | -1 | -1 | -1 | -1 | -3 | -3 |
| 11 | -5 | -5 | -7 | -7 | -5 | -5 | -5 | -7 | -7 | -7 | -7 | -7 | -7 | -7 | -7 | -7 | -7 | -7 | -7 | -7 | -7 | -7 | -7 | -7 |
| 12 | 15 | 15 | 9 | 9 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 |

**Note:** The table shows the sequence of DNA with numbers 1 and 2 checked.
FIGURE 1

| M | A | L | T | K | A | E | M | S | E | Y | L | F | ...
|---|---|---|---|---|---|---|---|---|---|---|---|---|---
| ATG | GCG | CTT | ACA | AAA | GCT | GAA | ATG | TCA | GAA | TAT | CTG | TTT | ...

|   | 1.000 | 0.469 | 0.018 | 0.451 | 0.798 | 0.469 | 0.794 | 1.000 | 0.428 | 0.794 | 0.193 | 0.018 | 0.228 |
|---|------|-------|-------|-------|-------|-------|-------|------|-------|-------|------|-------|-------|
|   | 0.057 | 0.018 | 0.038 | 0.035 | 0.202 | 0.057 | 0.206 | 0.319 | 0.206 | 0.794 | 0.807 | 0.018 | 0.046 |
|   | 0.275 | 0.007 | 0.033 | 0.046 | 0.199 | 0.199 | 0.018 | 0.033 | 0.037 | 0.176 | 0.033 | 0.088 | 0.033 |
|   | 0.199 | 0.088 | 0.038 | 0.007 | 0.046 | 0.033 | 0.037 | 0.037 | 0.176 | 0.033 | 0.007 | 0.088 | 0.033 |

Imaged copied from Slides on Statistical Methods in Genetics and Bioinformatics MAS84 at WPI
FIGURE 2

A

Accuracy (%) vs. Cycles

B

Accuracy (%) vs. Cycles
Compressing Code in R

```r
#library(ape);
library(stringr);
library(gtools)

#Program: DNA coding.
ThreeWordSeq = function(seq)
{
x=c("A","C","G","T");
valores = permutations(n=4,r=3,v=x,repeats.allowed=T);
seq3=getNumSeq(seq);
tripletas=rep(0,64);
return(tripletas);
}

TwoWordSeq = function(seq)
{
largo=length(seq);
AA=0;CA=0;GA=0;TA=0;AC=0;CC=0;GC=0;TC=0;
AG=0;CG=0;GG=0;TG=0;AT=0;CT=0;GT=0;TT=0;
for (i in 1:largo)
{
    chr=paste0(seq[i],seq[i+1],seq[i+2]);
    chr=toupper(chr);
    if (chr=="AA") {
        AA = AA+1;
    }else if (chr=="AC") {
        AC=AC+1;
    }else if (chr=="AG") {
        AG=AG+1;
    }else if (chr=="AT") {
        AT=AT+1;
    }else if (chr=="CA") {
        CA=CA+1;
    }else if (chr=="CC") {
        CC=CC+1;
    }else if (chr=="CG") {
        CG=CG+1;
    }else if (chr=="CT") {
        CT=CT+1;
    }else if (chr=="GA") {
        GA=GA+1;
    }else if (chr=="GC") {
        GC=GC+1;
    }else if (chr=="GG") {
        GG=GG+1;
    }else if (chr=="GT") {
        GT=GT+1;
    }else if (chr=="TA") {
        TA=TA+1;
    }else if (chr=="TC") {
        TC=TC+1;
    }else if (chr=="TG") {
        TG=TG+1;
    }else if (chr=="TT") {
        TT=TT+1;
    }
    TwoWord=c(AA,AC,AG,AT);
    TwoWord=c(TwoWord,c(CA,CC,CG,CT));
    TwoWord=c(TwoWord,c(GA,GCG,GG,GT));
    TwoWord=c(TwoWord,c(TT,TC,TG,TT));
    return(TwoWord);
}

TwoWordMatrix = TwoWordSeq(seq2);

#simulated sequence using iid model
x=c("a","c","g","t");
seq2 = sample(x, 10000, replace=TRUE);
TwoWordMatrix = TwoWordSeq(seq2);

#Function transfer DNA seq to numerical: a=1; c=2; g=3; t=4;
getNumSeq = function(seq) {
    numericalSeq = array(0, length(seq));
    for (i in 1:length(seq)) {
        if (seq[i]=="a") {
            numericalSeq[i] = 1;
        }else if (seq[i]=="c") {
            numericalSeq[i] = 2;
        }else if (seq[i]=="g") {
            numericalSeq[i] = 3;
        }else if (seq[i]=="t") {
            numericalSeq[i] = 4;
        }
    }
    return(numericalSeq);
}

is.even = function(x) x %% 2 == 0
is.odd = function(x) x %% 2 != 0

codificador = function(sequnecy)
{
    contador=0;
    calibrador=rep(0.34);
}
```

```
sumatoria=rep(0,34);
codificacion=rep("0",21);
codificacion[18]="!";
codificacion[20]="!";
j=1;
while (j<=length(sequency)) {
    contador=contador+1;
    sumatoria[contador]=strtoi(sequency[[j]])+strtoi(sequency[[j+1]])-strtoi(sequency[[j+2]]);
    calibrador[contador]=(2*(strtoi(sequency[[j]])*strtoi(sequency[[j+2]])))+
    (3*strtoi(sequency[[j+1]]));
    calibrador[contador] =calibrador[contador] -
    (3*strtoi(sequency[[j+1]]));
    j=j+3;
}
valor=0;
contador=0;
j=1;
while (j<=length(sumatoria)) {
    x=sumatoria[[j]]
    y=sumatoria[[j+1]]
    if(y==2) {
        y=9;
    } else {
        valor=valor-sumatoria[[i]];
        valor2=valor2+calibrador[[i]];
    }
    codificacion[19]=toString(valor);
    codificacion[21]=toString(valor2);
    return(codificacion);
}
valor=strtoi(paste0(x,y))
valor=valor+63;
contador=contador+1;
codificacion[contador]=chr(valor);
j=j+2;
valor=0;
valor2=0;
for (i in 1:length(sumatoria)) {
    if(is.odd(i))==TRUE) {
        valor=valor+sumatoria[[i]];
        valor2=valor2+calibrador[[i]];
    } else {
        valor=valor-sumatoria[[i]];
        valor2=valor2+calibrador[[i]];
    }
}
codificacion[19]=toString(valor);
codificacion[21]=toString(valor2);
return(codificacion);
#based on the first letter is the missing code (sum letter)
......
#for example if the code is 1000 we only will have 999, but the
#missin letter
#is on the first letter of the code.

data = scan("Datos_check.txt", what=character(1))
seq1=TwoWordSeq(data);
spilit = c(seq1[[1]],"","",seq1[[2]],"","",seq1[[3]],"","",seq1[[4]],"","",seq1[[5]],"","",seq1[[6]],"","",seq1[[7]],"","",seq1[[8]],"","",seq1[[9]],"","",seq1[[10]],"","",seq1[[11]],"","",seq1[[12]],"","",seq1[[13]],"","",seq1[[14]],"","",seq1[[15]],"","",seq1[[16]]);
write(c(data[1],"","",spilit),"compacted.txt",10,sep="");
contador = 0;
line=0;
for (i in 1:length(data)) {
    if (((contador==102)&&(i==length(data)))) {
        line=line+1;
        contador=0;
        seq2=getNumSeq(ExtractoSeq);
        codificado=rep("0",17);
        codificado=codificador(seq2);
    } 
    ExtractoSeq = rep(0,102);
    line=line+1;
    contador=contador+1;
    ExtractoSeq[contador]=datsai[[i]];
    if (((contador==102)&&(i==length(data)))) {
        line=line+1;
        contador=0;
        seq2=getNumSeq(ExtractoSeq);
        codificado=rep("0",17);
        codificado=codificador(seq2);
    } 
    ExtractoSeq = rep(0,102);
    line=line+1;
    contador=contador+1;
    ExtractoSeq[contador]=datsai[[i]];
    if (((contador==102)&&(i==length(data)))) {
        line=line+1;
        contador=0;
        seq2=getNumSeq(ExtractoSeq);
        codificado=rep("0",17);
        codificado=codificador(seq2);
    } 
    ExtractoSeq = rep(0,102);
    line=line+1;
    contador=contador+1;
    ExtractoSeq[contador]=datsai[[i]];
    if (((contador==102)&&(i==length(data)))) {
        line=line+1;
        contador=0;
        seq2=getNumSeq(ExtractoSeq);
        codificado=rep("0",17);
        codificado=codificador(seq2);
    } 
    ExtractoSeq = rep(0,102);
    line=line+1;
}
Decompressing Code in R

```r
#library(ape);
#library(combinat);
library(stringr);
library(gtools)

#Program: DNA coding.
TransMatrix=rep(0,16);
is.even <- function(x) x %% 2 == 0
is.odd <- function(x) x %% 2 != 0

lectura = function(data)
{
    sub1=word(data[2],1,sep="!");
    sub2=word(data[2],2,sep="!");
    sub3=word(data[2],3,sep="!");
    decodificada=c(sub1,sub2,sub3);
    for (i in 3:length(data))
    {
        sub1=word(data[i],1,sep="!");
        sub2=word(data[i],2,sep="!");
        sub3=word(data[i],3,sep="!");
        TransMatrix[i]=strtoi(substring);
        decodificada=c(decodificada,c(sub1,sub2,sub3));
    }
    return(decodificada);
}

numerifica = function(data)
{
    largo=dim(data)[1];
    ancho=dim(data)[2];
    Transision=matrix(0,nrow=largo,ncol=36);
    for (i in 1:largo)
    {
        for (j in 1:17)
        {
            valor=asc(data[i,j]);
            valor1=ifelse(valor>=0,valor%%10,(valor*-
1)%10);
            valor2=ifelse(valor>=0,valor%%10,(valor*-
1)%10);
            Transision[i,((j*2)-1)]=valor1;
            Transision[i,((j*2))]=valor2;
        }
    }
    return(Transision);
}

buscar = function(numero)
{
    x=c(1,2,3,4);
    valores =permutations(n=4,r=3,v=x,repeats.allowed=T);
    tripleta2=rep(0,3);
    for (i in 1:64)
    {
        valores[4]=valores[1]+valores[2]-valores[3];
        valores[5]=((2*(valores[1]*valores[3]))-
3*valores[2]);
    }
    return(c(probability[i,j],"suma":sum(valor)));
}
```
for (i in 1:64) {
    if (numero == valores[i,4]) {
        tripleta1 = c(valores[i,1], valores[i,2], valores[i,3]);
        ## print(tripleta1)
        tripleta2 = rbind(tripleta2, tripleta1);
    }
}
largo = dim(tripleta2)[1];
x3 = sample(2:largo, 1);
cadenab = tripleta2[x3,];
return(cadenab);

####### transcript DNA
convierte_adn = function(data) {
    largo_ADN = sum(TransMatrix) + 1;
    cadena1 = rep(0, 34);
    cadena2 = rep(0, 102);
    cadena3 = rep(0, 34);
    tripleta = rep(0, 3);
    secuencia = rep(0, 1);
    ### Number of cicles for exact match
    ### real length chain "cadena" = dim(data)[1]
    exacta = 0;
    distancia = 100;
    for (i in 1:1) {
        contador = 0;
        cadena1 = data[i, 1:34];
        check1 = data[i, 35];
        check2 = data[i, 36];
        for (repeticion in 1:cicles) {
            cadena2 = buscar(cadena1[1]);
            cadena3[1] = ((2 * cadena2[1] * cadena2[3]) -
                (3 * cadena2[2]));
            for (j in 2:34) {
                tripleta = buscar(cadena1[j]);
                cadena2 = c(cadena2, tripleta);
                cadena3[j] = ((2 * tripleta[1] * tripleta[3]) -
                    (3 * tripleta[2]));
            }
            ### print(cadena1);
            ### print(cadena2);
            ### print(cadena3);
            suma1 = ifelse(is.odd(k) == TRUE, suma1 + cadena1[k],
                suma1 - cadena1[k]);
            suma2 = ifelse(is.odd(k) == TRUE, suma2 + cadena3[k],
                suma2 - cadena3[k]);
        }
        exacta = ifelse(check2 == suma2, 1, 0);
        if (exacta == 1) {
            elegida = cadena2;
            repeticion = cicles;
            exacta = 0;
            ## print(elegida);
            ### print(c(repeticion, suma1, suma2));
        }
    }
    secuencia = c(secuencia, elegida);
    return(secuencia);
}

######## recodifica = function(data) {
    text = substring(data[1], 1:17, 1:17)
    text2 = decodificada[c(2, 3)];
    DNA = c(text, text2);
    i = 4;
    columnas = 1;
    while (i <= length(data)) {
        text = substring(data[i], 1:17, 1:17)
        text2 = decodificada[c(i + 1, i + 2)];
        text3 = c(text, text2);
        DNA = rbind(DNA, text3);
        columnas = columnas + 1;
        i = i + 1;
    }
i=i+3;
}
rownames(DNA)=c(1:columnas);
return(DNA);
}
codificador = function(sequency)
{
  contador=0;
calibrador=rep(0,34);
sumatoria=rep(0,34);
codificacion=rep("0",21);
codificacion[18]="/";
codificacion[20]="=
j=1;
while (j<=length(sequency))
{
  contador=contador+1;
  sumatoria[contador]=strtoi(sequency[j])+strtoi(sequency[j+1])-strtoi(sequency[j+2]);
calibrador[contador]=strtoi(sequency[j])+strtoi(sequency[j+1])+strtoi(sequency[j+2]);
  j=j+3;
}
valor=0;
contador=0;
j=1;
while (j<=length(sumatoria))
{
  valor=strtoi(paste0(sumatoria[j],sumatoria[j+1]));
  valor=valor+33;
  contador=contador+1;
codificacion[contador]=chr(valor);
  j=j+2;
}
for (i in 1:length(sumatoria))
{
  if(is.odd(i)==TRUE)
    { 
      valor=valor+sumatoria[i];
      valor2=valor2+calibrador[i];
    }
  else
    { 
      valor=valor-sumatoria[i];
      valor2=valor2-calibrador[i];
    }
}
codificacion[19]=toString(valor);
codificacion[21]=toString(valor2);
return(codificacion);
}
valor=0;
valor2=0;
for (i in 1:length(sumatoria))
{
  if(is.odd(i)==TRUE)
    { 
      valor=valor+sumatoria[i];
      valor2=valor2+calibrador[i];
    }
  else
    { 
      valor=valor-sumatoria[i];
      valor2=valor2-calibrador[i];
    }
}
codificacion[19]=toString(valor);
codificacion[21]=toString(valor2);
return(codificacion);
}
trials=1
tiempo <- proc.time() ##function that measures time
ADN_seq = convierte_adn(numeros);
###ends time measuring sequence
proc.time()-t2
tiempo=rbind(tiempo,t2);
accuracy=accuracy+10;
cicles = x*100;
tiempo[x+1,4]=cicles;
tiempo[x+1,5]=accuracy;
