IT468 Natural Computing Spring 2017
Prof. Manish K Gupta
DA-IICT

DNA Storage using Akhmetov Encoding

3rd May, 2017
Rushikesh Nalla 201401106
Omkar Damle 201401114

1 Introduction

Data is growing at an alarming rate and to keep up with it we need a better storage technology and this is where DNA comes into picture. DNA has extremely long half-life and unlike digital media it is not prone to degradation. Recently scientists were able to store 215 petabytes per gram of DNA. We have implemented the paper "A highly parallel strategy for storage of digital information in living cells" by Azat Akhmetov, Andrew D.Ellington and Edward M.Marcotte. This paper discussed a new encoding strategy which helps in detecting and correcting errors. We have coded in python and used Lempel Ziv Markov chain algorithm (LZMA) compression and edit-distance libraries.

2 Algorithm

2.1 Compression

- Initially the input file is checked whether it is a text file or an image so that it can encoded accordingly and prepared for compression.
- For a text file we have used utf-8 encoding and for images we used base64 encoding which outputs a byte object.
- The resulting data is compressed using Lempel Ziv Markov chain algorithm (LZMA) compression algorithm.

2.2 Codebook generation

- First all possible DNA sequences containing A,C,G,T of length 4 are created.
- All sequences with high repetitiveness are removed.
- All sequences with Levenshtein distance less than 3 are removed so as to allow recovery from upto one mutation (substitution) in each block (DNA sequence of length 4).
- The above process is repeated until we get 8 codewords for our codebook.
- As the process is random in nature we get a different codebook each time we generate it.

2.3 Mapping and Storing

- The byte stream (base 256) after compression is then converted to base 8 because the number of codewords in the codebook are 8.
- The base 8 string is then mapped with the generated codebook and we get the corresponding DNA sequence.
- The DNA sequence is stored in a text file illustrating storage in cells.

2.4 Error Introduction and Correction

- A letter from A,C,G,T (replacing letter) is chosen randomly along with a random location (one in each block) where the error is introduced.
- After the above step each block has a substitution error which needs to be corrected.
- Each block is checked against the codewords in the codebook and their corresponding Levenshtein distance is stored.
- The codeword with minimum distance is deemed correct and is replaced by that codeword.
- The obtained DNA sequence would then be free from errors.

2.5 Mapping and Conversion

- The DNA sequence is read one block at a time and codebook is used to obtain the integer value corresponding to that block.
- The integer sequence is in base 8 and is converted back to byte stream (base 256).

2.6 Decompression

- The data (byte stream) is decompressed using LZMA compression algorithm.
- Depending on whether the input was a text file or an image it has to be decoded accordingly.
- For text file it is decoded using utf-8 and for images using base64.

3 Example

We have tested our code with text files of varying sizes and also images in different formats like jpg, png etc. As an example we will use a text file named example.txt containing the word DAIICT and look at the working of the code and output at each step.

3.1 Compression

As the input is a text file it is encoded using utf-8. It returns a byte object.

Output: b'DAIICT'

The byte object is compressed using LZMA compression algorithm and stored in example.txt.xz file. The contents of the file are:

Output: b'\xfd7zXZ\x00\x00\x04\xe6\xd6\xb4F\x02\x00!\x01\x16\x00\x00\x00t/\xe5\xa3\x01\x00\
x05DAIICT\x00\x00\x00r\xb0\xbe\xcd\xb6M\x9b\xaa\x00\x01\x1e\x06\xc1/\xa4\x1d\x1f\xb6\xf3}
\x01\x00\x00\x00\x00\x04YZ'

3.2 Codebook

After following the procedure as mentioned in the algorithm we obtained the following codebook:

Codeword	Value
AGCG	0
CTAT	1
ATGA	2
TAGC	3
GATA	4
CGTC	5
TCTG	6
GCAC	7

Table 1: Codebook

We get a different codebook each time we run the code. We will be using this codebook for the example.

3.3 Mapping and Storage

After base conversion of byte stream from base 256 to base 8 we get the following octal sequence.

Output: 0o3751567513026400000011633265504300200020401054000000007205771321401000025042024451
120652000000001625413731555446633524000010740330113722035077333633720040000000000001054532

This octal sequence is mapped to DNA sequence using the generated codebook.

Output: TAGCGCACCGTCCTATCGTCTCTGGCACCGTCCTATTAGCAGCGATGATCTG
GATAAGCGAGCGAGCGAGCGAGCGAGCGCTATCTATTCTGTAGCTAGCATGATCTGCGTCCG
TCAGCGGATATAGCAGCGAGCGATGAAGCGAGCGAGCGATGAAGCGGATAAGCGCTATAGCG
CGTCGATAAGCGAGCGAGCGAGCGAGCGAGCGAGCGAGCGGCACATGAAGCGCGTCGCACGC
ACCTATTAGCATGACTATGATAAGCGCTATAGCGAGCGAGCGAGCGATGACGTCAGCGGATA
ATGAAGCGATGAGATAGATACGTCCTATCTATATGAAGCGTCTGCGTCATGAAGCGAGCGAG

CGAGCGAGCGAGCGAGCGAGCGCTATTCTGATGACGTCGATACTATTAGCGCACTAGCCTAT
CGTCCGTCCGTCGATAGATATCTGTCTGTAGCTAGCCGTCATGAGATAAGCGAGCGAGCGAG
CGCTATAGCGGCACGATAAGCGTAGCTAGCAGCGCTATCTATTAGCGCACATGAATGAAGCG
TAGCCGTCAGCGGCACGCACTAGCTAGCTAGCTCTGTAGCTAGCGCACATGAAGCGAGCGGA
TAAGCGAGCGAGCGAGCGAGCGAGCGAGCGAGCGAGCGAGCGAGCGCTATAGCGCGTCGATA
CGTCTAGCATGA

This DNA sequence is stored in a file named example.txt_DNA.txt

3.4 Error Introduction and Correction

The error introduced is random as described in the algorithm and this run letter 'A' was selected at random and the 4th letter of each block is replaced with this letter. DNA sequence with errors is as follows:

Output: TAGAGCAACGTACTAACGTATCTAGCAACGTACTAATAGAAGCAATGATCTA
GATAAGCAAGCAAGCAAGCAAGCAAGCACTAACTAATCTATAGATAGAATGATCTACGTACG
TAAGCAGATATAGAAGCAAGCAATGAAGCAAGCAAGCAATGAAGCAGATAAGCACTAAAGCA
CGTAGATAAGCAAGCAAGCAAGCAAGCAAGCAAGCAAGCAGCAAATGAAGCACGTAGCAAGC
AACTAATAGAATGACTAAGATAAGCACTAAAGCAAGCAAGCAAGCAATGACGTAAGCAGATA
ATGAAGCAATGAGATAGATACGTACTAACTAAATGAAGCATCTACGTAATGAAGCAAGCAAG
CAAGCAAGCAAGCAAGCAAGCACTAATCTAATGACGTAGATACTAATAGAGCAATAGACTAA
CGTACGTACGTAGATAGATATCTATCTATAGATAGACGTAATGAGATAAGCAAGCAAGCAAG
CACTAAAGCAGCAAGATAAGCATAGATAGAAGCACTAACTAATAGAGCAAATGAATGAAGCA
TAGACGTAAGCAGCAAGCAATAGATAGATAGATCTATAGATAGAGCAAATGAAGCAAGCAGA
TAAGCAAGCAAGCAAGCAAGCAAGCAAGCAAGCAAGCAAGCAAGCACTAAAGCACGTAGATA
CGTATAGAATGA

After this the errors are corrected using the procedure mentioned earlier and we get back the original DNA sequence.

3.5 Mapping and Conversion

The DNA sequence is converted to an integer sequence using the mapping present in the codebook.

Output: 3751567513026400000011633265504300200020401054000000007205771321401000025042024451
120652000000001625413731555446633524000010740330113722035077333633720040000000000001054532

It is converted to byte stream which is required for decompression.

Output: b'\xfd7zXZ\x00\x00\x04\xe6\xd6\xb4F\x02\x00!\x01\x16\x00\x00\x00t/\xe5\xa3\x01\x00\
x05DAIICT\x00\x00\x00r\xb0\xbe\xcd\xb6M\x9b\xaa\x00\x01\x1e\x06\xc1/\xa4\x1d\x1f\xb6\xf3}
\x01\x00\x00\x00\x00\x04YZ'

The data above are the contents of the file example.txt_comp.xz

3.6 Decompression

The data stream is decompressed using LZMA compression algorithm.

Output: b'DAIICT'

This text is decoded using utf-8 and written to file example.txt_decoded.txt

Output: DAIICT

4 Results

In all the four files tested by us, we got 100% accuracy in retrieving the information even after introducing errors. We carried out the simulations for the following input files:

Data	Size of file	Size of .xz file	Length of DNA
sym10.png	10,297 bytes	10,688 bytes	114,008
DAIICT.jpg	14,109 bytes	14,424 bytes	153,856
longSameText.txt	10,000 bytes	108 bytes	1152
longRandomText.txt	10,000 bytes	4,472 bytes	47,704

We can observe that the length of the nucleotide sequence depends a lot on the lzma compression i.e. length of .xz file. For image files, the compression is not significant because of the small file size. However, for text files, the compression is quite significant. The flat file consisting of 10,000 zeros is compressed from 10,000 bytes to only 108 bytes! Moreover, the file consisting of random 10,000 characters is compressed to around 4500 bytes.

Dhirubhai Ambani Institute of Information and Communication Technology
DA-IICT

Figure 1: DAIICT.jpg

In the following table, we have presented the percentages of the 4 nucleotide types in the encoded files :

Name of file	Percentage of A	Percentage of C	Percentage of G	Percentage of T
sym10.png	31.3	19.9	21.8	28.0
DAIICT.jpg	25.2	24.8	25.2	24.8
longSameText.txt	22.5	29.3	23.1	25.1
longRandomText.txt	25.0	25.1	25.0	24.9

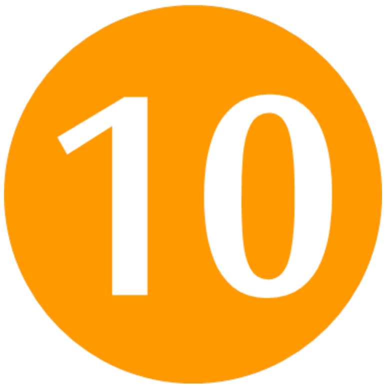

Figure 2: sym10.png

The DAIICT.jpg file and longRandomText.txt file give a uniform distribution of the 4 base pairs. Due to repetitiveness, the longSameText.txt shows some non-uniformity. We didn't find any plausible reason for the sym10.png file to give such non-uniform results. It may be due to the compression technique used.

5 Extensions

- We have introduced only one type of error - by substitution. Other error types like insertion and deletion can be introduced and error correction techniques need to be altered.
- Other file formats like pdf, gif, exe can be tested.

References

[1] Azat Akhmetov, Andrew Ellington, Edward Marcotte, A highly parallel strategy for storage of digital information in living cells, biorxiv.org/content/early/2016/12/26/096792, Dec 2016.

[2] https://img.collegepravesh.com/2014/06/DA-IICT.jpg

[3] https://pbs.twimg.com/profile_images/591032213166301185/jymVKEjM_400x400.png