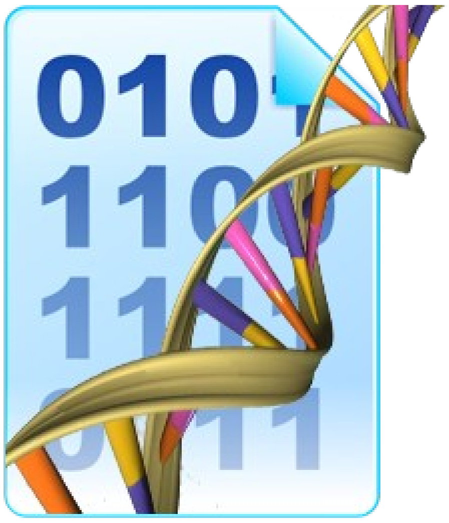
Index



1. INTRODUCTION

1.1 Intention of this document

This document is intent to describe the basic principles of DNA and is focusing on the encoding part. It does not contain information about the DNA algorithms itself or how the algorithms work together, this information including the intellectual rights are property of the author and will not be published.

1.2 About DNA

DNA is a combination of algorithms that makes it possible to calculate for every file type, exe, bin, mp3, mpeg, wav, etc, a key code with a minimum size that is smaller than, or equal to 256 bytes. With the help of a reference table, DNA can restore the original file from this key code. DNA does not use known compression techniques but is a different way of storing and restoring data lossless. The name DNA was chosen because the algorithms, like genetic DNA, uses unique (DNA) sequences to carry the information from which the original data can be restored. DNA is not a replacement of current compression techniques but is an addition to it and they can work very well together.

1.3 History

The idea that data could be stored, archived more efficiently than we do up to now was spinning in my mind for a lot of years starting in the '90s. The intention was to look for a way to store or archive text documents and images on a more efficient way so hard disk storage space can be used optimal.

Use a compression tool you would probably think now!

Nothing against that thinking and it would have been the easiest way but at that point my feeling was asking me, hey, isn't there a better and more efficient way? I had to agree with my feeling and from that point on I started to look for that way. After many attempts without positive results espied of the fact that I learned a lot about compression, finally the DNA algorithms where born. Versions 2 of DNA is designed to demonstrate and to prove the existence of that idea in conjunction with the algorithm, and they did! There is a long way to go but the base is there. DNA is a hobby project and although the amount of research that still is to be done it offers already many possibilities, for example data encryption!

2. BASIC PRINCIPLES

2.1 The idea behind DNA

The main idea behind DNA is to store common used data only once! Its a simple idea but easier said than done. A file is nothing more or less than a sequence of numbers irrespective of its type. By creating unique (DNA) sequences from blocks of data which are equal for every file type, it would be possible to replace this block by a shorter reference. This also opens the possibility to use iterations which can bring the final size back to a minimum, in the case of DNA <= 256 bytes. A few of years ago I stumbled upon a story of Jan Sloot, a dutch inventor that had developed a revolutionary data storage technique. After reading the book “De Broncode”, I recognized a lot in what I was looking for and to be honest, it pushed me into the right direction. There are also some remarkable similarities in the results.

2.2 Encoding / decoding

The basic encoding process is visualized by the flowchart in appendix 5.1. The process starts with opening, reading a file and dividing it into unique data blocks. For every block the DNA algorithms calculates a (DNA) sequence. After this DNA checks if this sequence is already present in the reference table. If not, it will be added to the table and gets an reference, but when it is already in the table the sequence will not be added but only his reference will be used. The block will be replaced by his smaller reference and after processing every block a check on the total final size is done. If this size is still bigger than then requested size, <= 256 bytes is the minimum but every size above that can be made, the output is taken back to process it again until the requested size is reached. When the total process is finished the final result will be stored as the key code. Decoding is in basic the same as encoding only reversed.

2.3 The algorithms

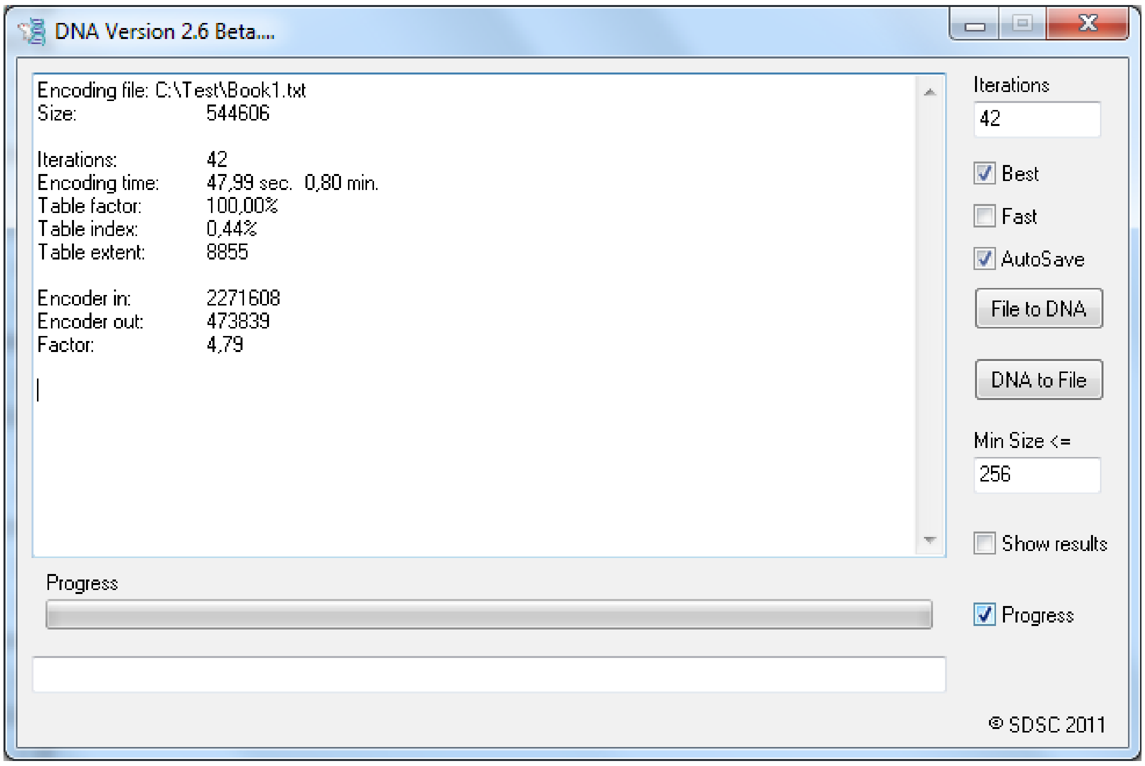
Version 2.x of DNA uses 5 algorithms (Jan Sloot?). They work together and the complete processing is done by DNA and this all is mathematical proofed. The first intention of DNA was to optimize data storage on a local device and that’s the reason for the dynamic reference table. Don’t store more than necessary! If key codes where mend to be exchangeable between systems then this test version of DNA would need a table with a size of approximately 9,3652456509479766941447916893795e+145 bytes. This was not relevant for testing with this version and beside of that a table of such size is impossible to handle. Besides that, billions of sequences in the table will probably never be used.

3. TEST RESULTS

3.1 The first test results

The test results presented here are from DNA version 2.6. Work on version 3 is started but in between there where some new ideas and they could be easier implemented and tested in version 2.4 and up.

Lets start with an empty reference table and a screenshot from version 2.6 with the results of encoding a file.



```
Encoding file: C:\Test\Book1.txt
Size:               544606

Iterations:         42
Encoding time:      47,99 sec.  0,80 min.
Table factor:       100,00%
Table index:        0,44%
Table extent:       8855

Encoder in:         2271608
Encoder out:        473839
Factor:             4,79
```

Screenshot 1

In this case the input file has a size of 544606 bytes and the final size for the key code is set to <= 256 bytes. As you can see, the number of iterations to get there is 42. Important are the 'encoder in' and 'encoder out' values! Because of the use of iterations the total amount of data processed by the encoder is a lot more than the original file size. This is depending on the number of iterations that are necessary to reach the requested final size of the key code. The 'encoder in' value is the amount of bytes that the encoder processed and the 'encoder out' value is the total amount of bytes added to the the reference table.

I can hear you think now!

Encoder out = 473839 and file size = 544606, that's a data reduction of 12.99%, any compression tool would do a better job here!

But lets be honest, it would be unfair to judge DNA on this simple calculation because there is more than that.

Compression tools would do a better job on this single file because they are specially designed to know about file types and will compress a single file optimal to that knowledge. DNA instead does not look at the file type and processes every file as equal, just a sequence of numbers! That's also one of the reason why iterations are possible. Try to compress a compressed file again and you will see that it will not compress any more because all the redundancy was removed at the first run. The algorithms reached the end of there capacity and this results that the file grows again. (negative compression!)

Lets have a look at some test results.

First of all, the 'encoder out' versus the amount of data processed by the encoder. The input file size is 544606 bytes but to reach the final requested size of the key code it takes some iterations the get there. These iterations make the amount data flowing thru the encoder a lot more than the size of the input file. In this case the encoder had to process 2271608 bytes and its final output was 473839 bytes. In fact the real data reduction percentage has to be calculated from these values and is (2271608 – 473839) / (2271608 / 100) = 79,14%, started with a empty reference table. This was already the first evidence that the algorithms work and they did there job as expected.

Why?

Forget about the input file for the moment and have a look a the total bytes processed by the encoder, 2271608 bytes in and 473839 bytes out. Remember the basic idea of DNA, creating unique (DNA) sequences from blocks of data which are equal for every file type. Another important value here is the 'Table extent' (see Screenshot 1) which is 8855. This value indicates the number of (DNA) sequences added to the table. We started with an empty reference table, which also is necessary to get a good impression of what the algorithms are doing and generating. From the total stream of 2271608 bytes the algorithms calculated 8855 references with a total size of 473839 bytes. This indicates that the algorithms already found matching sequences and can rebuild the total stream of 2271608 bytes out of these 8855 references with the size of 473839 bytes!

Bear in mind that this was the first file processed by the algorithms and a table with no reference at all. When more files are processed by DNA the number of matching sequences will increase along the way and of course resulting in less sequences added to the table. Notice that the total number of bytes added to the reference table is smaller then the size of this input file.

3.2 Reference table size versus input

The following diagram shows us how the size of the reference table grows when the number of files processed by DNA increases.

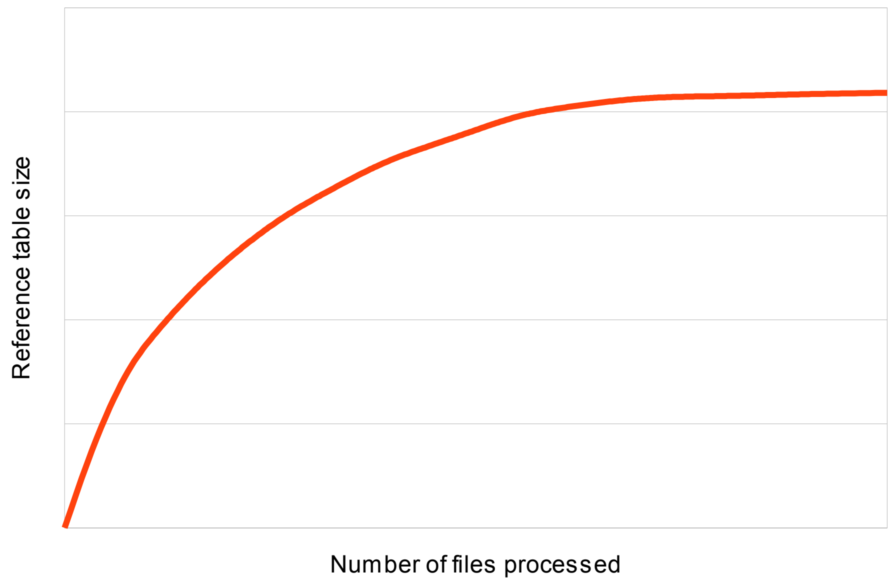


Diagram 1

This was exactly as expected and you can see, the more files are processed the less data is added to the table and the more effective DNA becomes. This diagram also shows that DNA with a dynamic table is not a replacement of compression tools. DNA in this case needs more files/data to get effective (it has to learn ;-)). For single files the usage of a compression tool would still be a better option.

3.3 Key code size versus iterations

Another important aspect was how the size of the key code was behaving by each iteration. As told before DNA does not use know compression algorithms like run time length encoding, Huffman trees, etc, etc. But beside that, I noticed something remarkable in the first iteration. I will come back on this later but first have a look at diagram 2 which shows the key code size versus the number of iterations. For this diagram I took a small file of 2489 bytes but bigger files showed the same behavior. This file took 18 iterations to get to a key code size of <= 256 bytes. In appendix 5.2 you find the key code diagram for the (bigger) file used in screenshot 1 and 2.

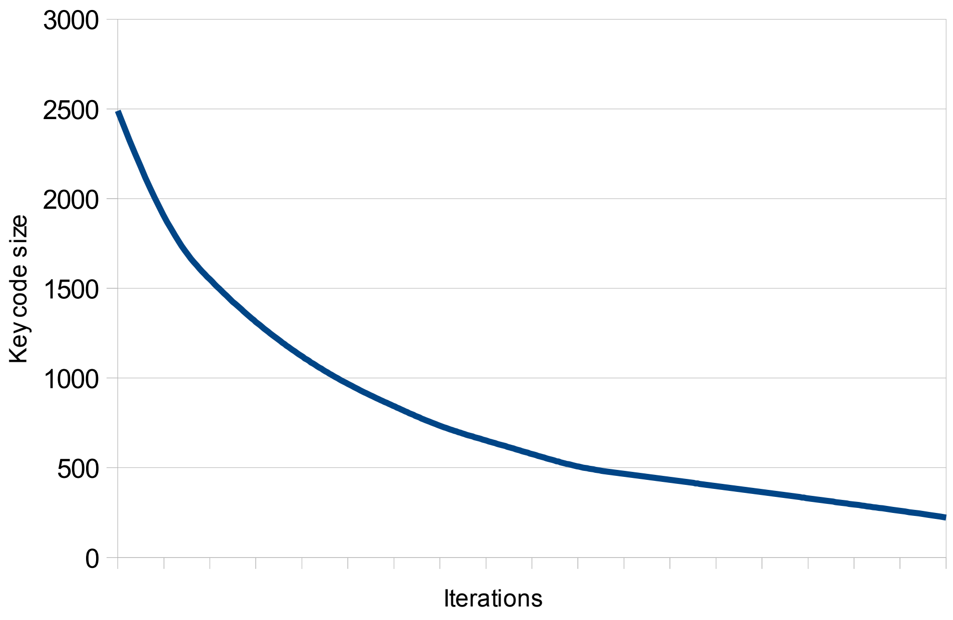
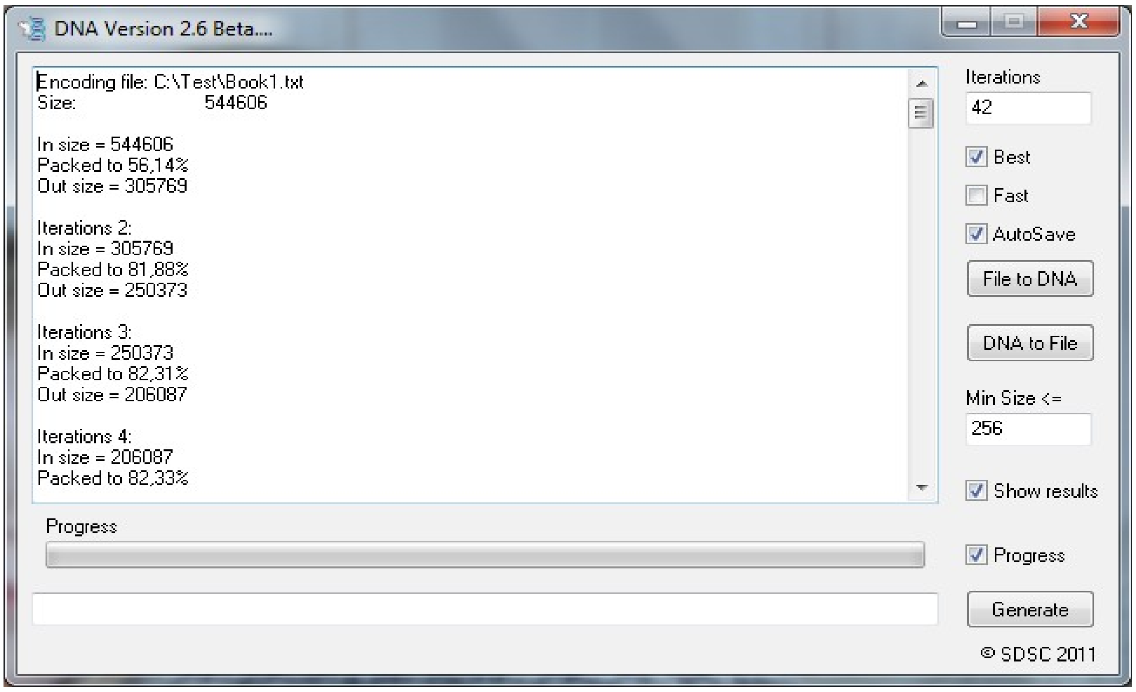


Diagram 2

This behavior was a little bit strange to me because my expectation was that the size of the key code would decrease at around the same amount by each iteration but it did not! The reason for that was found in the first iteration. See the next screenshot.



```
Encoding file: C:\Test\Book1.txt
Size:                544606

In size = 544606
Packed to 56,14%
Out size = 305769

Iterations 2:
In size = 305769
Packed to 81,88%
Out size = 250373

Iterations 3:
In size = 250373
Packed to 82,31%
Out size = 206087

Iterations 4:
In size = 206087
Packed to 82,33%
```

Iterations: 42
[✓] Best
[ ] Fast
[✓] AutoSave
Min Size <=: 256
[✓] Show results
[✓] Progress

Screenshot 2

Although the DNA algorithms do not use known compression techniques they seem to be able to remove most of the redundancy in the first iteration! I did a test and compressed the same file as in screenshot 2 using win-zip and the compressed size was 209409 bytes. This is still smaller than the result of the first iteration but the algorithms are doing a very good job here not knowing the file type they are dealing with. If you look at diagram 2 there is another remarkable thing. Up to the fifth iteration the key code decreases strong but after that the effect of the resting iterations gets smaller (thirteen iterations for 750 bytes). Now have a look at the size of the key code at the fifth iteration, approximately 1K (Jan Sloot?). Another nice feature of iterations is that one can set the size of the final output (key code). Although DNA can calculate a key code of <= 256 bytes, this is not necessarily always the best choice. If for example DNA is used to optimize data on a hard disk, the best choice is a final size of <= sector size. This will reduce the number of iterations and hard disk space would be used optimal.

4. SUMMARY

4.1 Further research

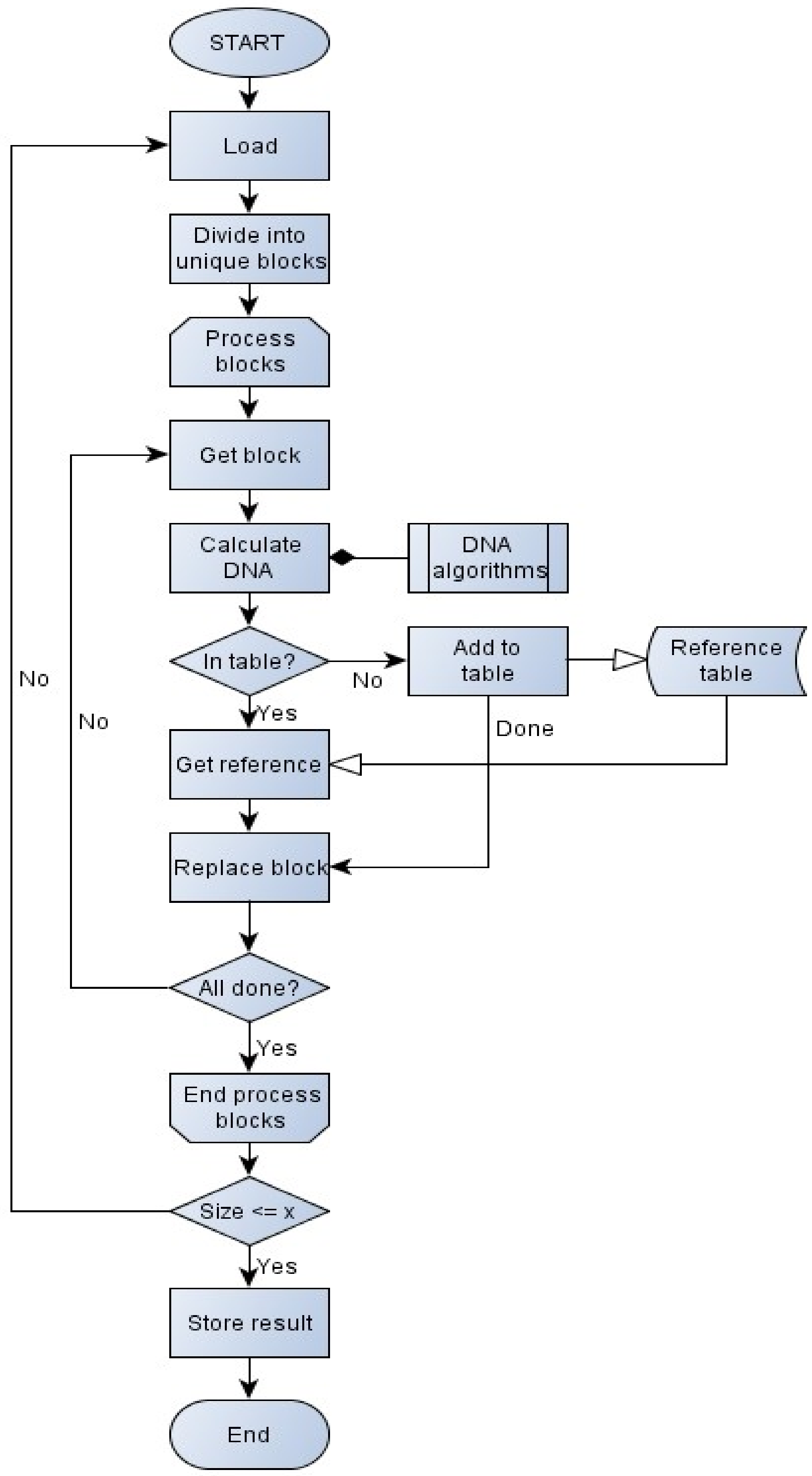
The test results of version 2 showed that the algorithms are working as expected with very positive results, so further research and work on DNA can continue. They showed new room for optimization and ideas but also some points of attention, speed for example. Beside that another very big challenge would be the reference table, first of all getting it smaller but also can we do without it? From test versions to a final application is a long way and there is still a lot of research to be done but the results showed a solid base here.

4.2 Note from the author

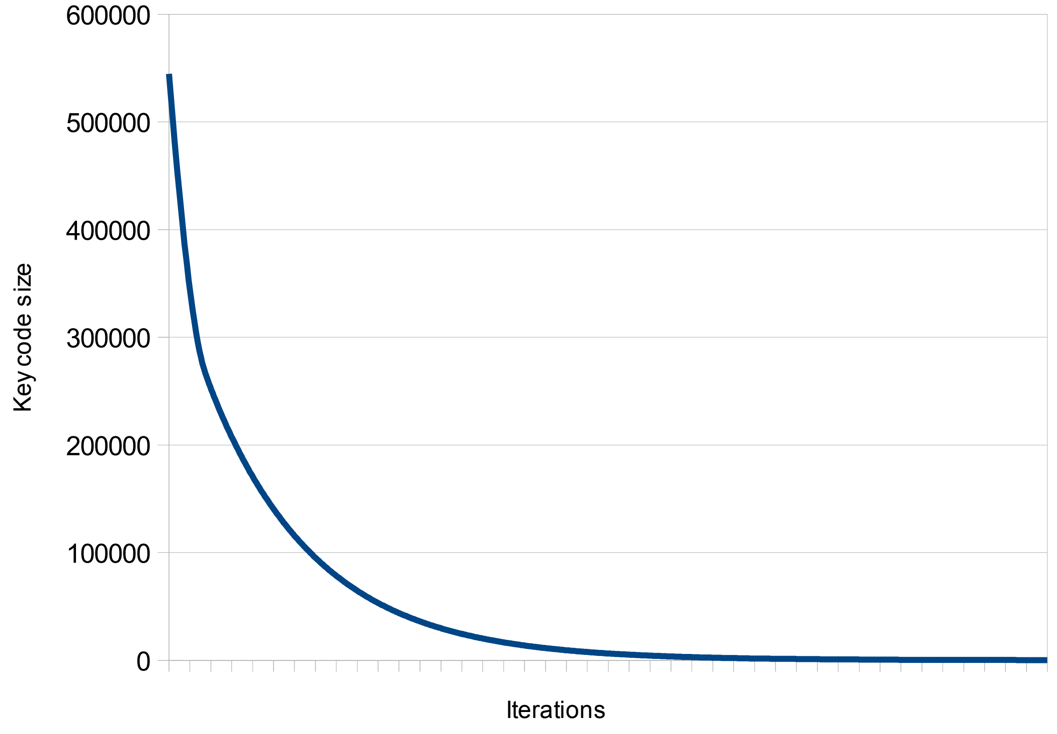
I do not claim that I have found the solution of the missing source code from Jan Sloot or a way to achieve endless compression. DNA is a hobby project and the main goal is the search for alternative ways to store data. The video demo, forum and this white paper are intend to show the status of the project and achievements/results up to now. I really would like to show and tell more about DNA and it algorithms but unfortunately certain circumstances are making this impossible. For more information about, the status of or questions please visit jansloot.telcomsoft.nl.

5. APPENDIX

5.1 Encoding flowchart



5.2 Key code diagram for file in screenshot 1 and 2


Key code size
Iterations