2016

Compressing Genome Resequencing Data

Aliya Farheen

Compressing Genome Resequencing Data

Aliya Farheen

Thesis submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Donald A. Adjeroh, Ph.D., Chair
Elaine M. Eschen, Ph.D.
YanFang Ye, Ph.D.

Lane Department of Computer Science and Electrical Engineering
West Virginia University

Morgantown, West Virginia
December 2016

Keywords: DNA resequencing, genome compression, common substrings, compression algorithm

Abstract

Compressing Genome Resequencing Data

Aliya Farheen

Recent improvements in high-throughput next generation sequencing (NGS) technologies have led to an exponential increase in the number, size and diversity of available complete genome sequences. This poses major problems in storage, transmission and analysis of such genomic sequence data. Thus, a substantial effort has been made to develop effective data compression techniques to reduce the storage requirements, improve the transmission speed, and analyze the compressed sequences for possible information about genomic structure or determine relationships between genomes from multiple organisms.

In this thesis, we study the problem of lossless compression of genome resequencing data using a reference-based approach. The thesis is divided in two major parts. In the first part, we perform a detailed empirical analysis of a recently proposed compression scheme called MLCX (Maximal Longest Common Substring/Subsequence). This led to a novel decomposition technique that resulted in an enhanced compression using MLCX. In the second part, we propose SMLCX, a new reference-based lossless compression scheme that builds on the MLCX. This scheme performs compression by encoding common substrings based on a sorted order, which significantly improved compression performance over the original MLCX method. Using SMLCX, we compressed the *Homo sapiens* genome with original size of 3,080,436,051 bytes to 6,332,488 bytes, for an overall compression ratio of 486. This can be compared to the performance of current state-of-the-art compression methods, with compression ratios of 157 (Wang et.al, Nucleic Acid Research, 2011), 171 (Pinho et.al, Nucleic Acid Research, 2011) and 360 (Beal et.al, BMC Genomics, 2016).

Acknowledgements

I would like to express deepest gratitude to my research advisor Dr. Donald A. Adjeroh, for the relentless support and guidance. I thank him to made me realize that with determination and hard work, pursuing a dream is not an impossible task. I also thank my committee members, Dr. Elaine M. Eschen and Dr. YanFang Ye for their suggestions and support. I also want to thank Dr. Richard Beal for being supportive and helpful in numerous ways and guided me through tough times. I am thankful to the LCSEE department for providing me with resources and funding through the Graduate Teaching Assistantship. My deepest gratitude to my father Mohammed Adbul Aleem, my mother Fahmeena Faryal and my brother for their unconditional love and support. I am grateful to be blessed with friends who were always available for the moral support and who helped in their respective capacities that made this journey (masters studies) easier for me. I express my obedience and gratitude to my creator and my sustainer Allah, for I believe his plans are better than my dreams.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Problem and Motivation

Data compression has been studied for the past few years as a means to cope with huge repositories of biological information. Scientists acquire the genetic information of a particular species by studying the DNA segments. The DNA segments are sequenced to determine the precise order of the four nucleotides/bases that compose a DNA molecule, i.e., adenine (A), guanine (G), cytosine (C), and thymine (T). Knowledge of these DNA sequences is important in biological research and in numerous applied fields such as medical diagnosis, biotechnology, forensic biology, virology and biological systematics. The early sequencing methods did not allow rapid sequencing of genomic species other than viruses and could not handle large amounts of DNA. Sanger sequencing [1], was a breakthrough that helped scientists determine the human genetic code, but was time-consuming and expensive.

Recent developments, such as Next Generation Sequencing (NGS) technologies, enabled faster sequencing with high data accuracy while reducing the cost. In addition, NGS made large-scale whole-genome sequencing accessible and practical for a researcher. Next-generation sequencing generates masses of DNA sequence data that is richer and more complete than is imaginable with Sanger sequencing. These newer technologies can deliver data outputs ranging from 300 KB to 1 TB in a single sequencing run depending on the instrument type and configuration. Sequencing data (in raw/assembled format) from a project are either archived by the respective laboratory, or submitted to major public sequence repositories, such as GenBank (NCBI), EMBL, DDBJ, etc. These repositories are currently witnessing exponential growth with respect to the number of sequences being submitted each year [2]-[4]. Moreover, new large-scale sequencing projects are frequently announced raising storage concerns. Statistics show storage requirement of high-throughput sequencers in the world to be in the range of 50-100 PB per year [5]. The need for efficient storage and communication of such data is heightened significantly. To address these challenges, genomic data compression methodologies have emerged.

The DNA sequence can be represented in two bits (00, 01, 10 and 11) to match each of the four bases to generate a compressed file. But, standard compression software such as Unix “compress” and “compact” or the MS-DOS archive programs “pkzip” and “arj” often result in data expansion [4][6][7] using more than two bits per base. The major reason for the expansion

is that these methods use models for traditional text and therefore fail to consider certain special characteristics of biological sequences. Genome sequences are known to convey important purposeful information between different generations of an organism. Moreover, biological sequences contain different types of repetitions and other hidden regularities. Long runs of tandem repeats and randomly interspersed repeats are prominent features of DNA sequences [4]. Thus, from the viewpoint of compression and sequence understanding, the repetitions inherent in biological sequences imply redundancies, which provide an avenue for a significant compaction. Specialized approaches capturing the exact/inexact repeat patterns have been suggested to achieve better compression than the general purpose compression algorithms. Few methods depict successful compression ratios in past few years [4][6][8]-[19].

Genome compression techniques have various design choices like lossy/loss-less compression, ability to handle variable length sequenced reads, random access, encoding strategies, reference dependency, single/multiple sequence alignment blocks. In this work, we study the problem of lossless compression of genome resequencing data using a reference based approach. This approach develops an end-to-end compression scheme to generate an efficient mapping between a reference and a target (to-be compressed) genome. Then the observed mapping is described in an efficient manner. This problem and its variations have been studied in [17] [18] [20] - [23]. In [18] the authors introduced GReEn, a compression tool based on arithmetic coding that overcomes some drawbacks of the previously proposed tool GRS [17]. In [19] an algorithm that uses the maximal longest common substrings/subsequences to compress a source sequence (target genome) given the reference is introduced. Motivated by the compression efficiency in [19], further analysis of this technique led to development of the new algorithm in this study.

1.2 Thesis Contributions

The contributions of the thesis are summarized as follows:

1. A detailed empirical analysis on the MLCX compression scheme proposed in [19].
2. Proposed a model to predict the influential *k*-parameter value using reference sequence without application of the compression methodology.
3. Introduced a decomposition technique, which works conjointly with MLCX compression to improve the compression.
4. A new compression model (called SMLCX) that performs compression using an ordered selection of the common sub-patterns. This results in a significant improvement over the MLCX, currently the best performing compression scheme for genome re-sequencing data.

1.3 Thesis Outline

Chapter 2 presents a brief background and a review on compression methods for genomic re-sequencing data. A quantitative comparison of the different compression methodologies – their efficacies and limitations are presented. Chapter 3 presents a detailed empirical analysis of the recently proposed method -MLCX (Maximal Longest Common Substrings/Subsequences). The chapter is divided into two parts. First part involves an empirical analysis of the MLCX compression methodology and the evaluation of *k*-parameter, which has a significant influence on the effectiveness of the MLCX compression methodology. Second part deals with the proposed new model for determining the *k* value and a decomposition technique, which improves the compression performance for MLCX compression scheme.

In Chapter 4 we present SMLCX, an improved compression scheme. Results from the developed new compression methodology has been tested with three sets of genomes, viz., *Arabidopsis thaliana, Oryza sativa* and *Homo sapiens.* The significance of k-value is studied. Furthermore, the developed decomposition technique has been applied on the *Homo sapiens* genome set.

Finally, conclusions are presented in Chapter 5, with a brief recommendation for future study.

1.4 Publications Resulting in part from this Thesis

1. R. Beal, T. Afrin, A. Farheen, and D. Adjeroh, "A new algorithm for 'the LCS problem' with application in compressing genome resequencing data," in *Proceedings - 2015 IEEE International Conference on Bioinformatics and Biomedicine,* 2015, pp. 69–74. [19% acceptance rate].
2. R. Beal, T. Afrin, A. Farheen and D. Adjeroh, "A new algorithm for 'the LCS problem' with application in compressing genome resequencing data," *BMC Genomics*, vol. 17, no. S4, p. 544, 2016.
3. R. Beal, A. Farheen and D. Adjeroh, "Compressing genome resequencing data via the maximal longest factor," in *Proceedings - 2016 IEEE International Conference on Bioinformatics and Biomedicine*, 2016. [19 % acceptance rate].

Chapter 2

Background and Related Work

Genomic study poses serious computing requirements for acquiring, distributing, and analyzing genomic data as more and more genomes are investigated. According to [24] by 2025, an estimated 100 million to 2 billion human genomes could have been sequenced. The data-storage demands for these could run to as much as 2 to 40 exabytes (1 exabyte $=10^{18}$ bytes). Although a single genome is only about 3.164 billion symbols, this increase is because the amount of data that must be stored for a single genome is about 30 times larger than the size of the genome itself, to make up for errors incurred during sequencing and preliminary analysis [24]. This problem is compounded by the fact that genomes from other organisms or species are also being sequenced at a rapid pace, some of which are even larger than the human genome.

Compression techniques are the traditional means for handling huge data storage. These methods reduce the space for storage and speed up the data transfer (e.g., among research institutes). Another application of genome compressibility is to measure the “relatedness" between two DNA sequences or two genomes important for biological research [4][44]. It is recognized that the compression of DNA sequences is a very difficult task [6][8][25]. The compression ratio achieved by general purpose compression algorithms for the genomic sequence data is suboptimal. Existing genome compressors capture the repeat patterns prevalent in biological sequences and then encode them using an optimal encoding scheme.

The scope of this work is to study reference based loss-less compression methodologies. The input of a compression algorithm is a sequence of symbols from a given alphabet ($\sum$). In referential compression, first a reference genome is fixed and all further sequencing data of the same species is mapped to the reference and only the differences (the compressed file) are stored. As differences between individuals of the same species are small, the new sequences frequently map very well to the reference, with differences occurring less than 1% of the bases [26]. The long similar referential match blocks are found using the index structures e.g. hash-based or suffix trees. Thus, the new sequences can be compressed by only noting location in and differences to the reference. Availability of reference sequence to the decompressor enables rapid compression rates. The Fig.2.0 shows the algorithmic sketch of referential compression schemes.

Algorithm 1 Sketch of a Referential Compression Algorithm

```
1: while input contains characters do
2:     Find longest matching substring in reference for current input position
3:     if length of match > K then
4:         Encode match as (match position, length)
5:     else
6:         Encode match with raw symbols
7:     end if
8: end while
```

Figure 2.0: Referential Compression Algorithm

Lossless compression allows us to reconstruct the complete original input from the compressed output unlike lossy compression. In most biomedical applications, lossless sequence compressors are preferred as every single base is important. Out of different available genomic data compression methods, each with different approaches for analyzing the sequence variation, we study the most popular methods GRS, GReEn, GDC 2, GDC ultra, FRESCO, TGC, RLZ, RLZopt and a new algorithm for the LCS problem with application in compressing genome resequencing data.

2.1 GRS: [17]

One reference-based approach to compressing whole genome sequence is via the use of reference SNP maps [46]. Wang et al. [17] recognized the genome sequences could be compressed without the reference SNP maps or information regarding sequence variation. The main idea of the GRS (Genome ReSequencing) method was to process the given genome sequence data without the usage of the reference SNPs or other sequence variation information, and then to rebuild the individual genome sequence data using a reference genome sequence. The compression technique involved stepwise procedures as follows:

1. Evaluate the varied sequence percentage parameter (δ) for the input chromosome file based on the reference chromosome.
2. For the chromosomes, where $\delta \leq 0.03$
    - Evaluate the longest common sequence to find and record the varied nucleotide sequence and pre-code the extracted difference.
    - Compress the reduced difference sequence file using BZIP2 using the Huffman coding
    - Generate the command file to decompress the compressed file.
3. For the chromosomes, where $0.03 \leq \delta \leq 0.1$
    - Segment each chromosome into 'n' number of pieces and calculate each difference rate δ_i $(1 \leq i \leq n)$.

- Find the position with minimal $\Sigma\delta_i$, then compress each piece following strategy in step 2.

4. For a chromosome with $\delta \geq 0.1$, the sequence is not suitable to compress with the tool. Report and terminate compression. No compression is performed.

The varied sequence percentage δ plays a key role in the GRS compression. The tool quantifies the usage of the correct reference chromosome by evaluating the percentage of the nucleotide sequence variation. High variation results in larger compressed file with more time for the individual genome sequence. GRS used the modified UNIX diff function to find the sequence variation in the technique. GRS achieved a ~159-fold compression tested against the sequencing datasets from Homo sapiens, Oryza sativa and *Arabidopsis thaliana*. The GRS can compress the *A. thaliana* genome data from 115.1 MB to 6.5 KB. The genomic sequences with the sequence variation beyond a threshold cannot be compressed by GRS.

2.2 GReEn: [18]

GReEn (Genome Resequencing Encoding) compression technique overcomes the major drawbacks of the GRS [17] which include the compression of the genome sequences with excessive difference to the reference sequence, storage space requirements, and long running times. GReEn also handles the arbitrary alphabets and uses arithmetic coding. Like most related compression schemes, the compression efficiency of this GReEn depends on the degree of similarity between both the reference and target sequences.

The compression was executed using arithmetic encoding on the fixed relative frequency of the characters in the target sequence and the estimated probability of the character subjected to change in encoding process. Compression efficiency depended upon the provision of good probability estimates. The probability distribution is given by two sources (i) an adaptive copy model which assumes the characters of the target sequence to be copies from the reference genome;(ii) a static model that relies on frequencies of characters in target sequence.

The GReEn copy model was inspired by the copy expert of the XM [13] DNA compression method. Here probability computations mainly depend on pointer positions in the reference sequence that have a good chance of containing a character identical to the character being encoded. To overcome this limitation a hash table is constructed with the occurrences of the positions in the reference sequence of all the k-most-recently-encoded substrings (k-mers) of a given size.

GReEn achieved faster running times and compression gains of over 100-fold for genome sequences. A study of GReEn compression method was made on genome datasets, namely *Arabidopsis thaliana, Oryza sativa, Homo sapiens* (for four different human genome assemblies,namely HuRef, Celera, YH and KOREF_20090224). Compression throughput of this

probabilistic copy model was compared with the GRS. The performance of the encoder significantly depended on the size of the *k*-mer and the number of prediction failures tolerated by the copy model before it is restarted. Optimization of these parameters was not performed.

2.3 New LCS-Motivated Approach [19]

Beal et al. [19] proposed a new approach to compressing genome re-sequencing data using common patterns between the reference and target sequences. We call this the MLCX method since it exploits the maximal longest common sub-patterns (subsequences/substrings). The MLCX approach measured the similarity between the genomic sequences using the Longest Common Subsequence (LCS) and utilized the LCS components rather than the LCS itself in the compression technique. The LCS has varied biological applications in sequence alignment for comparative genomics, phylogenetic construction and analysis, rapid search in huge biological sequences, compression for efficient storage of the genomic data sets. A new algorithm for the LCS problem using suffix trees and the shortest-path graph algorithms was proposed. LCS calculation involved three main steps (i)Construction of the Generalized Suffix Tree(GST) for two sequences to calculate the common substrings (CSS) shared between two sequences; (ii)Construction of the directed acyclic graph(DAG) of the obtained maximal CSS; and (iii)Computation of the LCS by finding the longest path in the DAG. The CSS were found by a preorder traversal of the GST.

The MLCX compression scheme exploits the common substrings (CSS) which constitute the building blocks for the LCS. The CSS between the reference(*R*) and target (*T*) are computed. The Longest Previous Factor (LPF) data structure [32][33] is used to choose only the maximal CSS's that make up the target. Chosen CSS are represented or encoded as tuples. The triplet encoding record CSS position in *T*, position in *Z* (*Z*=*T***R*), and length. Further compression of these tuples is done using the standard compression schemes such as PPMD, LZMA2(two modes of 7-ZIP), BZIP2(a standard BWT based compressor). The MLCX scheme when applied to the *Homo sapiens* genome resulted in a compression ratio of 199. This can be compared with the compression ratios of current state-of-art methods such as GRS, GReEn (159 and 171 respectively).

2.4 GDC 2: [27]

GDC 2[27] is a scheme which analyzed the problem of compressing large collections of complete genomic sequences. Compression of such large collections involve several sub problems. These include compression in (1) raw sequencing reads; (2) reads after mapping onto reference genomes; (3) results of variant calling ;(4) complete genomic sequences. GDC 2 utilizes the high similarity between genomes in large sets belonging to the same species for

compression. For instance, two human genomes are known to be 99.5% similar [28]. Such high similarity levels enable better compression ratios when a collection of genomes are compressed rather than individual genome. GDC 2 was motivated by GDC-ultra [29], FRESCO [26], TGC [30], three earlier compression schemes for collection of genomes.

GDC-ultra [29] constructs a hash table based on a reference. The first sequence in a given collection of genome set is compressed by evaluation of similarities between current sequence and the reference. Each processed sequence is used as an additional reference to compress next sequences in the genome set. Total number of references is limited to 40, i.e., the 41st sequence in the collection is compressed using 40 references. The differences are later Huffman coded and results are recorded. GDC-ultra compressed a collection of 69 human genomes with a compression ratio of about 1000 [27].

Wandelt et.al [26] developed **FRESCO**, another algorithm to compress collections of genomes. The algorithm divide collection into two sets:(i) additional references, and (ii) remaining sequences. A search structure (suffix tree) is constructed for the primary reference. The similarities between additional references and primary reference sequences are measured by performing classical Ziv–Lempel parsing of additional references. For each additional reference, an order of triples (position in the primary reference, length of the identical part, next symbol) is obtained. A search structure (hash table) is built for the Ziv–Lempel-parsed additional reference sequences, which is used to further compress the remaining sequences from the collection. Then, the sequence of triples is compressed using the additional Ziv–Lempel-parsed reference sequences serving as the second level reference. The obtained compression ratios are impressive. A compression ratio of about 3000 [27] was obtained for the collection of about 1000 haploid genomes from the 1000 Genome Project, when 70 additional reference sequences are used.

TGC algorithm [30] also compresses a collection of genomes but using a different input format. This algorithm processes the Varied Call Format (VCF) files that describes differences between genomes and the reference sequence. The main idea of TGC is to split the VCF file into two files. The first file serves as dictionary of variants and stores a description of each variant (i.e., its type, position, alternative alleles, etc.). The second file stores the binary representation of presence/absence of each single variant in each single sequence. The bit vectors corresponding to each individual genome are then compressed using a specialized Ziv–Lempel-based algorithm. The dictionary file is also compressed using a specialized algorithm. For the collection of 1092 diploid human genomes considering one reference sequence, the compression ratio is about 15,500 [27].

Combining the methods from GDC-ultra and FRESCO algorithms, GDC 2 compression algorithm uses a two level Ziv-Lempel factoring for genome set (S), after the construction of the search structure for the reference sequence R. At the first level, the Ziv-Lempel factoring of all

sequences from the collection S was done producing a sequence L^k for each sequence S^k composed of tuples. At the second level, GDC 2 performs a similar Ziv-Lempel factoring for each sequence L^k in the collection L to obtain each differential sequence D^k in the collection D. Each sequence D^k is composed of three kinds of tuples (1) first level literal (pair) ; (2) first level match (triple); and (3) Second level match(quadruple). After this encoding of the differences between the predictions (values of tuples e.g. matching positions) and real values is computed followed by the arithmetic encoding of the successive tuples.

Decompression of the GDC 2 is done by applying arithmetic decoding on the compressed file from which collection L is decoded. The collection of sequences S were constructed from L and R. The multithreaded design of GDC 2 [27] executed four times faster than the existing algorithms. For the human data set, the algorithm was quite fast (about 200 MB/s) and collection of 1092 human genomes was compressed to 700 MB, which can be compared to 6.7 TB of the uncompressed FASTA files. This is a compression ratio of over 9,570.

Another compression scheme RLZ [20] is based on self-indexing. The algorithm compresses input sequences with LZ77 [31] encoding relative to the suffix-array of a reference sequence. Raw sequences are never stored and even very short matches to the reference are encoded. Careful consideration of the reference sequence is vital for this method since initial results with cross-species compression were discouraging. RLZopt [21] is presented as an extension of RLZ. The key aspect in RLZopt compression is longest increasing subsequence computation that allows to efficiently encode positions. It incorporates improvements like local look-ahead optimizations and random access.

In this work, we focus on compression of individual genomes, rather than a collection of genomes. Thus our work is more closely related to RLZ, GRS, GReEn and MLCX. The methods we propose can be extended to compress multiple genomic sequences in a collection.

Chapter 3

Empirical Analysis and Improvement on MLCX Compression

This chapter details an empirical analysis performed on two representative genome datasets using the MLCX compression algorithm. The empirical analysis was performed on the genome datasets *Oryza sativa* and *Homo sapiens*. Thus the performance in data compression was evaluated and compared with that of the available reference based compression algorithms, GRS and GReEn.

The study revealed that genome alphabet size, symbol-case, and the parameter *k* were vital in enhancing performance of the compression process. Further, a new decomposition strategy was applied in conjunction with the MLCX algorithm, which further improved the results. It was observed that, under the decomposition approach, the specific representation used- in terms of the bitwise encoding of the symbols before applying the MLCX algorithm also has a definite impact on compression performance.

3.1 MLCX Algorithm [19]

The MLCX compressor implementation can be separated into two major phases – preprocessing and encoding. In preprocessing phase, the common substrings (CSS) that exist between the reference and target sequences are calculated and are made accessible in data structures. The maximal CSS that make up the common subsequences between the target (T) and reference (R) are also identified in this phase. At the encoding phase, exact matching compression methodology that use repeat positions and length to represent an exact repeat is implemented. The matching common substring blocks and unmatched plain characters/symbols between the reference and the target are stored in separate files. The maximal CSS's are chosen to encode repeat positions of the target (*T*) in sequential order. The encoded output files are further compressed using standard text compressors, such as LZ, PPM, or BZIP2.

The common substrings are computed using the generalized suffix tree (GST) structure. A GST is a suffix tree for a set of strings {S1, S2, ..., Sn} that contains all the suffixes from each sequence in the set. It can be built in linear time and linear space, and used to find all occurrences of the common substrings between the given strings. The label on any path from the root to an internal node in GST must be a substring of set of original strings. Hence, GST guarantees that each suffix is represented by a unique path on the tree [34][35]. To intelligently choose which CSS's are likely to be part of the common subsequences that will lead to improved compression, the Longest Previous Factor (LPF) [32][33] is used. The LPF data structure provides

the length of the longest factor (matching substring) of a string W starting at position i, that occurs previously in W. For instance, the LPF for the string "***abra**cad**abra***" is of length 4 and occurred at position i=0. Both the length and position of matches are stored using the LPF and POS data structures respectively.

Consider compressing the target T with respect to the reference R. Let $Z = R°T$, where '°' denotes the concatenation operations. The MLCX algorithm works by building upon the target with tuples by choosing the maximal CSS's and thereafter compresses these tuples with other available compression schemes such as PPMD, LZMA2 and BZIP2. The algorithm adopts a left-to-right directive sequential scan of the LPF data structure and the position (POS) data structure on Z, to compress the target. The algorithm therefore outputs two files - the triples file and the symbols file, which are initially void of any data.

3.1.1 Compression Methodology:

Let $i = |R| + 1$. If $LPF[i] < k$, then the character symbol is encoded by appending 1-byte char T[i-IRI] to symbols file and thus the value of i is incremented. Else if $LPF[i] \geq k$, the recognized CSS is encoded with the triplet (pT, pZ, l), where $pT = i\text{-}|R|$ is the starting position of the CSS in T, $pZ = POS[i]$ is the starting position of the CSS in Z $[1 \ldots i\text{-}1]$, and $l = LPF[i]$ is the length of the CSS. In the aforementioned case, each of the 4-byte words pT, pZ, and l are appended to the triples file. Now $i = i + l$ is set, to consider compressing the suffix following the currently encoded CSS. This process continues unto $i \leq |Z|$. The algorithm is shown in Fig.3.1. The figure shows the portion of the script that handles the compression of the target T (given the reference R, parameter k, and the LPF and POS arrays on Z) into the symbols file (with filename symbols_fn) and triples file (with filename triples_fn).

```
void compress(String T,String R,int k,int LPF[z],
          int POS[z],String symbols_fn,String triples_fn){
  int i,posT,posZ,len,rlen=|R|
  File* symbols=open(symbols_fn,WRITE)
  File* triples=open(triples_fn,WRITE)
  for i=rlen+1 to z {
    posT=i-rlen
    len=LPF[i]
    if(len<k){
      write_char(symbols,T[posT])
    }else{
      write_int(triples,posT)
      posZ=POS[i]
      write_int(triples,posZ)
      write_int(triples,len)
      i=i+len-1
    }
  }close(symbols)
  close(triples)
}
```

Figure 3.1: Compression function for the MLCX compression scheme

The output files *triples* (long words file) and *symbols* (byte file) are binary sequences that can be further compressed with standard compression schemes. Based on the k-value in the constructed LPF data structure, the compression algorithm uses only one of the two files to encode the target. The algorithm proceeds by converting the observed maximal CSS's (based on k) either into triples (12 bytes) or by encoding a symbol with using one byte.

3.1.2 Decompression Methodology:

The decompression also involves a left-to-right scan on both triples and symbols files. For instance, consider the current long word $w1$ in triples. According to the proposed triple encoding, this will be the position of a CSS in T. If $i = w1$, then the next two long words $w2$ and $w3$ in triples file are selected. At this point $T[i...i+w3-1] = Z[w2...w2+w3-1]$. Since there is access to just R and $T[1...i-1]$ during decompression, it is only feasible to pick up each symbol of $Z[w2...w2 + w3-1]$. This selection is done with $R[j]$ if $j \leq |R|$ or $T[[j-|R|]$ otherwise, for $w2 \leq j \leq w2+w3-1$. After this step, i is incremented as $i = i+w3$. On the other hand, if $i \neq w1$, the next char c in symbols file is selected since $T[i] = c$. Here i is incremented and the process continues until $i \leq |T|$. Fig. 3.2 shows the algorithm to reconstruct the target T with the reference R using the symbols file (with filename symbols_fn) and the triples file (with filename triples_fn).

```
String decompress(String R,String symbols_fn,
                                        String triples_fn){
  String T
  int s=1,t=1,i=1,j,begins,len,loc,rlen=|R|
  int slen=count_file_bytes(symbols_fn)/sizeof(int)
  int tlen=count_file_bytes(triples_fn)/sizeof(char)
  File* symbols=open(symbols_fn,READ)
  File* triples=open(triples_fn,READ)
  bool more=true
  while(more){
    loc=peek_at_int(triples)
    if(t≤tlen ∧ i=loc){
      get_next_int(triples) // discard
      begins=get_next_int(triples)
      len=get_next_int(triples)
      t+=3
      for j=begins to begins+len−1 {
        i++
        if(j>rlen) T+=T[j−rlen]
        else T+=R[j]
      }
    }else if(s≤slen){
      i++
      s++
      T+=get_next_char(symbols)
    }else more=false
  }close(symbols)
  close(triples)
  return T
}
```

Figure 3.2: Decompression function for the MLCX compression scheme

3.2 Empirical Analysis

In the following sections, the MLCX compression scheme is applied on genomes from different species, and evaluated in terms of the genome alphabet (∑) size, letter-case of alphabet symbols, the k-parameter, and different reference genomes for the human genome. The data compression tests were conducted on Amazon Elastic Compute Cloud (EC2) with instance type m4.4xlarge. The specifications of m4.4xlarge instance include 16 vCPU count, 64 GiB RAM and high network performance.

The performance of the MLCX compression algorithm was examined on real genomic data belonging to different species. We use the hg18 release from the UCSC Genome Browser, the *Homo sapiens* Korean genomes KOREF20090131 and KOREF20090224 [36], the *Homo sapiens* genome of a Han Chinese referred to as YH [37]. Two versions of the genome of the thale cress *Arabidopsis thaliana*, TAIR8 and TAIR9 [38], and of the genome of the rice *Oryza sativa* TIGR5.0 and TIGR6.0 [39] were also used. This real-world genomic datasets often contain symbols beyond the traditional four letter DNA alphabet ∑= {A, C, G,T}, with each symbol denoting one of the four basic nucleotides: *adenine* (A), *cytosine* (C), *guanine* (G), *thamine* (T). The possible reason behind these extra non- ATGC characters is the representation of repetitive stretches, low-complexity regions by lowercase characters, gaps of indeterminate length by a hyphen (-) character or ambiguous cases with unknown character symbol [40]. The extended alphabet properties of the datasets considered in terms of alphabet size and symbols are also further discussed.

The ***Arabidopsis thaliana*** genome dataset consists of 5 chromosomes, with byte sizes ranging between 18.5 MB to 30.5 MB. The sequence alphabet size (|∑|) for the first two chromosomes is 11 symbols. For chromosomes 3, 4, 5 the |∑| is 10, 7, 5 symbols respectively. The ***Oryza sativa*** genome comprises of 12 chromosomes, with byte sizes ranging between 23 MB to 43 MB. Generally, the chromosomes in this dataset contain alphabet sizes of 5 symbols. Nevertheless, chromosome 3, 10 and 11 have alphabet sizes of 6, 9 and 11 symbols, respectively. The ***Homo sapiens*** KOREF genome dataset sequences are comprised of alphabet size with 21 symbols for all the 24 chromosomes. The 21 symbols are {A, C, G, K, M, R, S, T, W, Y, a, c, g, k, m, n, r, s, t, w, and y}. However, the mitochondria DNA chromosome M contains just 11 symbols, unlike other members of the dataset. This dataset consists of chromosomes with byte sizes approximately ranging between 50 MB to 247 MB.

3.2.1 Analysis of *Oryza sativa* Genome

Genomic compression tests were performed on *Orzya sativa* using TIGR6.0 as target and TIGR5.0 as reference. The output files symbols and triples were further compressed using

standard compression schemes, LZMA2 and PPMD. The compression results are tabulated in Table 3.1.

Table 3.1: Results of MLCX Compression Scheme on *Oryza sativa* genome TIGR6.0 using TIGR5.0 as reference. $\mathbb{C}$ denotes the result of the encoded genome using basic MLCX encoding. $\mathbb{L}(\mathbb{C})$, $\mathbb{P}(\mathbb{C})$ denotes result of applying LZMA2 and PPMD compression on $\mathbb{C}$.|X| denotes the length or size of X in bytes.

Chromosome	Size of Chromosome (bytes)	MLCX			GRS [17] (bytes)	GReEn[18] (bytes)
		$\lvert\mathbb{C}\rvert$ (bytes)	$\lvert\mathbb{L}(\mathbb{C})\rvert$ (bytes)	$\lvert\mathbb{P}(\mathbb{C})\rvert$ (bytes)		
1	43,268,879	15,207	4,735	4,551	1,502,040	4,972
2	35,930,381	4,645	1,649	1,517	1,409	1,906
3	36,406,689	54,234	15,693	15,556	47,764	17,890
4	35,278,225	21,474	6,636	6,432	36,145	6,750
5	29,894,789	17,030	5,431	5,359	6,177	5,539
6	31,246,789	12	146	141	14	482
7	29,696,629	5,899	2,064	1,972	4,067	2,448
8	28,439,308	23,126	8,794	10,115	118,246	9,507
9	23,011,239	12	146	141	14	366
10	23,134,759	175,228	49,713	50,277	788,542	60,449
11	28,512,666	41,407	13,006	13,351	2,397,470	14,797
12	27,497,214	12	146	141	14	429
Sum	**372,317,567**	**358,286**	**108,159**	**109,553**	**4,901,902**	**125,535**

From Table 3.1, it is evident that the encoding itself achieved a compression ratio of 1039 which was further improved to 3442 by application of the LZMA2 compression. While, the standard GRS and GReEn algorithms compressed the dataset with ratios of 76 and 2966 respectively. This translates to a 97.7% improvement over GRS (4,901,902 bytes) and a 13.8% improvement over GReEn (125,535 bytes) using MLCX algorithm.

Observe that, the compressed data obtained upon encoding chromosomes 6, 9 and 12 were 12 bytes each. These chromosomes are identical with the reference TIGR5.0 and only needed one triplet in triples file for encoding. However, further compression of the obtained output files in MLCX using the LZMA2 or PPMD resulted in data expansion. Procedure on how to handle such distinct case is discussed in section 3.2.2.

3.2.2 *Homo sapiens* genome- KOREF

Data compression tests were conducted by compressing the *Homo sapiens* genome with KOREF assembly [1] using KOREF_20090224 as target and KOREF_20090131 as reference. The dataset was tested on different variations of symbol-case in the extended genome alphabet. Firstly, MLCX compression was deployed on the original extended genome alphabet with $|\sum|$ of

21 symbols {A, C, G, K, M, R, S, T, W, Y, a, c, g, k, m, n, r, s, t, w, and y} in KOREF dataset. Later, compression methodology was applied for the genome alphabet where character-case is not significant with |∑| of 11 symbols. for the 24 chromosome. Further, all the non-ATCG characters of the extended alphabet were mapped to a symbol 'N' and compression scheme was applied with |∑| of 5 symbols {a, c, g, t, n}.

As a special case for the mitochondria DNA chromosome M, LZMA2 technique was not applied on $\mathbb{C}$, the basic encoded sequence, since this led to data expansion rather than compression. Therefore, the chromosome M was indicated by simply appending a 25-bitstring header, without further compression. This technique increases the size of the overall encoded file by 4 bytes, but it is still better than the GRS and GreEn results.

3.2.2.1 Compressing the original sequence using the original extended genome alphabet

Table 3.2 summarizes the results for the case where the original sequence alphabet was preserved. The results obtained by compressing the encoded symbols and triples (see Section 3.1.1) files with LZMA2 were 21.39% and 13.97% improvement respectively, over GRS and GReEn algorithms respectively. On the other hand, compression performed with PPMD algorithm for the encoded symbols and triples, were significantly higher when comparison with GRS and GReEn. The original 3,080,436,051 bytes of data from KOREF_20090224 dataset is compressed to 15,460,323 bytes using MLCX encoding technique in conjunction with LZMA2 compression.

3.2.2.2 Case when the character-case of the alphabet symbol is not significant

In this section, the results are summarized for the scenario where in the character-case (upper/lower) of the alphabet considered were not significant. Here, the genome alphabet in the KOREF assembly will contain just 11 symbols {A, C, G, K, M, R, S, T, W, Y, N} for 24 chromosomes and the chromosome M with only 7 symbols. Detailed results are shown in Table 3.3. In this scenario, a total of 3,080,436,051 bytes in the considered dataset was compressed to 2,178,031 bytes using LZMA2 and to 3,429,666 bytes using PPMD.

3.2.2.3 Case with an alphabet composed only of characters from set {a, c, g, t, n}

In this scenario, the entire set of characters were transformed to lowercase prior to compression and the unknown nucleotides were all mapped to symbol 'n'. After the said mapping transformation, the sequences in the genome dataset will comprise only of characters from the set {a, c, g, t, n} reducing the alphabet set size to 5 instead of 21. In this variation, the original 3,080,436,051 bytes in the considered genome dataset was compressed to 2,142,110 bytes using LZMA2 as shown in Table 3.4. Post the encoding step, the encoded symbols and triple's files using LZMA2 compression technique provided better results than RLZ and GRS. The GReEn approach [18] recorded the best result in this scenario.

Table 3.2: Compression results for KOREF_20090224 using KOREF_20090131 as reference while preserving the original extended genome alphabet. $\mathbb{C}$ denotes the result of the encoded genome using basic MLCX encoding. $\mathbb{L}(\mathbb{C})$, $\mathbb{P}(\mathbb{C})$ denote results of applying LZMA2, PPMD compression on $\mathbb{C}$. |X| denotes the length or size of X in bytes.

Chromosome	Size of Chromosome (bytes)	MLCX			GRS [17] (bytes)	GReEn [18] (bytes)
		$\lvert\mathbb{C}\rvert$ (bytes)	$\lvert\mathbb{L}(\mathbb{C})\rvert$ (bytes)	$\lvert\mathbb{P}(\mathbb{C})\rvert$ (bytes)		
1	247,249,719	2,836,652	1,082,859	1,529,521	1,336,626	1,225,767
2	242,951,149	2,871,186	1,050,170	1,538,040	1,354,059	1,272,105
3	199,501,827	2,115,410	790,444	1,168,800	1,011,124	971,527
4	191,273,063	2,398,432	910,898	1,298,385	1,139,225	1,074,357
5	180,857,866	2,064,874	764,458	1,119,632	988,070	947,378
6	170,899,992	1,902,067	710,355	1,042,874	906,116	865,448
7	158,821,424	2,326,721	844,194	1,216,189	1,096,646	998,482
8	146,274,826	1,617,884	617,996	877,833	764,313	729,362
9	140,273,252	1,877,509	704,205	971,220	864,222	773,716
10	135,374,737	1,623,010	617,633	872,519	768,364	717,305
11	134,452,384	1,586,558	604,901	861,654	755,708	716,301
12	132,349,534	1,476,523	566,997	810,150	702,040	668,455
13	114,142,980	1,100,576	399,527	587,787	520,598	490,888
14	106,368,585	1,026,227	377,695	552,091	484,791	451,018
15	100,338,915	1,055,663	398,720	549,673	496,215	453,301
16	88,827,254	1,225,378	443,009	630,605	567,989	510,254
17	78,774,742	1,081,739	396,371	566,697	505,979	464,324
18	76,117,153	865,138	320,361	448,896	408,529	378,420
19	63,811,651	862,129	320,789	459,144	399,807	369,388
20	62,435,964	605,179	229,418	320,095	282,628	266,562
21	46,944,323	488,340	180,096	435,182	226,549	203,036
22	49,691,432	568,734	205,244	446,176	262,443	230,049
X	154,913,754	7,525,925	2,494,884	3,649,064	3,231,776	2,712,153
Y	57,772,954	1,343,260	429,099	581,344	592,791	481,307
M	16,571	151	151(*)	151(*)	183	127
Sum	**3,080,436,051**	**42,445,265**	**15,460,474**	**22,533,722**	**19,666,791**	**17,971,030**

Table 3.3: Compression results for KOREF_20090224 using KOREF_20090131 as reference when the symbol case in the extended genome alphabet is ignored. $\mathbb{C}$ denotes the result of the encoded genome using basic MLCX encoding. $\mathbb{L}(\mathbb{C})$, $\mathbb{P}(\mathbb{C})$ denote results of applying LZMA2, PPMD compression on $\mathbb{C}$. |X| denotes the length or size of X in bytes.

Chromosome	Size of Chromosome (bytes)	MLCX		
		$\lvert\mathbb{C}\rvert$ (bytes)	$\lvert\mathbb{L}(\mathbb{C})\rvert$ (bytes)	$\lvert\mathbb{P}(\mathbb{C})\rvert$ (bytes)
1	247,249,719	381,577	161,319	256,974
2	242,951,149	356,526	153,805	239,789
3	199,501,827	284,096	119,348	190,691
4	191,273,063	330,381	137,301	219,597
5	180,857,866	259,922	109,768	173,917
6	170,899,992	265,222	110,544	175,675
7	158,821,424	292,797	121,289	192,652
8	146,274,826	222,972	93,378	146,574
9	140,273,252	309,512	132,957	201,428
10	135,374,737	245,264	103,115	160,554
11	134,452,384	222,735	92,471	145,730
12	132,349,534	214,123	88,447	138,954
13	114,142,980	148,938	62,730	97,607
14	106,368,585	141,128	57,354	93,249
15	100,338,915	138,219	58,777	91,122
16	88,827,254	151,606	62,779	97,541
17	78,774,742	136,168	57,030	88,495
18	76,117,153	113,469	47,122	73,573
19	63,811,651	130,468	53,531	82,966
20	62,435,964	94,273	38,689	60,088
21	46,944,323	71,121	28,744	44,373
22	49,691,432	81,329	33,663	50,898
X	154,913,754	523,282	196,868	322,567
Y	57,772,954	152,464	57,002	84,652
M	16,571	64	64(*)	64(*)
Sum	**3,080,436,051**	**5,267,656**	**2,178,095**	**3,429,666**

Table 3.4: Compression results for KOREF_20090224 using KOREF_20090131 as reference when alphabet only composed of characters {a, c, g, t, n}. $\mathbb{C}$ denotes the result of the encoded genome using basic MLCX encoding. $\mathbb{L}(\mathbb{C})$ denotes result of applying LZMA2 compression on $\mathbb{C}$. |X| denotes the length or size of X in bytes.

Chromosome	Size of Chromosome (bytes)	MLCX		RLZ [20] (bytes)	GRS [17] (bytes)	GReEn[18] (bytes)
		$\lvert\mathbb{C}\rvert$ (bytes)	$\lvert\mathbb{L}(\mathbb{C})\rvert$ (bytes)			
1	247,249,719	375,756	152,562	591,629	152,388	90,555
2	242,951,149	357,368	146,673	576,769	146,754	89,440
3	199,501,827	284,821	117,372	472,814	117,544	72,708
4	191,273,063	330,939	135,034	471,157	134,628	83,611
5	180,857,866	260,446	108,110	428,287	108,407	66,597
6	170,899,992	265,974	108,776	411,404	109,866	67,264
7	158,821,424	293,461	119,001	395,524	119,223	71,898
8	146,274,826	223,280	92,707	350,337	94,139	56,650
9	140,273,252	309,882	131,109	357,584	119,647	68,607
10	135,374,737	245,797	101,536	335,464	101,486	60,303
11	134,452,384	223,127	90,831	326,836	91,380	54,966
12	132,349,534	214,769	87,643	320,444	89,170	55,408
13	114,142,980	149,195	58,306	266,378	64,313	36,962
14	106,368,585	141,483	58,643	248,165	58,865	34,245
15	100,338,915	138,595	58,020	235,094	56,569	32,693
16	88,827,254	151,886	61,496	217,748	60,580	35,315
17	78,774,742	136,504	56,172	193,700	55,582	33,836
18	76,117,153	113,625	46,447	182,604	48,098	29,191
19	63,811,651	130,763	52,873	162,826	53,355	30,505
20	62,435,964	94,494	38,178	149,403	38,114	22,969
21	46,944,323	71,135	28,501	112,822	29,048	16,620
22	49,691,432	81,440	33,387	119,791	32,562	18,423
X	154,913,754	523,473	202,113	428,878	224,997	129,497
Y	57,772,954	152,219	56,620	150,901	61,306	33,312
M	16,571	64	64(*)	56	75	54
Sum	**3,080,436,051**	**5,270,496**	**2,142,174**	**7,506,615**	**2,168,096**	**1,291,629**

3.2.3 *Homo sapiens* Genome- YH Vs KOREF

The KOREF genomes (KOREF_20090224 and KOREF_20090131) are obtained from two individuals with the same ancestry (Korean). Thus they are relatively more similar when compared with genomes from two people with different ancestry. This section details the test case conducted on the *Homo sapiens* genome using YH assembly as target and KOREF_20090224 as reference. The YH genome is taken from a person with Han Chinese origin. Thus, there are more differences between YH and KOREF genomes when compared with KOREF_20090224 and KOREF_20090131. YH dataset consists of 24 chromosomes with byte sizes ranging between 50 MB to 247 MB and consists of sequence alphabet size of 11 symbols {A, C, G, K, M, N, R, S, T, W, Y}. The mitochondria DNA chromosome M have lesser alphabet symbols. Although, the alphabet case dependency was not implemented for this test scenario, it could be a potential work of interest to investigate in the future.

Table 3.5 summarizes the compression results where the original sequence alphabet was preserved in target YH and reference KOREF_20090224. GRS could not compress most of the chromosomes in this assembly given the significant differences between the genome sequences (here δ>0.3) [17]. The original 3,080,436,051 bytes in the considered dataset was compressed to 19,365,065 bytes using LZMA2. This translates to a 38.4% improvement over GReEn (31,415,217 bytes).

The compression ratio observed using target YH and reference KOREF_20090224 is 159. In Table 3.2, the target KOREF_20090131 was compressed with the reference KOREF_20090224 with a compression factor of 199. In the reference based compression methodologies the compression rates highly depend on the similarity between the to-be-encoded sequence (target) and the reference. Referentially compressed DNA sequences comprise lists of long stretches of interval matches and reach highest compression rates for in-species compression. However, compressing, for instance, a human genome against a mouse genome leads to considerably worse rates due to the many short matches found. The variation in the two assemblies YH and KOREF in the *Homo sapiens* genome lead to reduction in compression performance.

3.3 Choice of *k*-Parameter

The *k*-parameter is a threshold value to determine if it's beneficial to either encode a given CSS as symbol (i.e. 1 byte) or as triple (i.e. 12 bytes). Based on compression results obtained for genomic dataset *Arabidopsis Thaliana* using the target TAIR9 and reference TAIR8, *k*-parameter value was chosen to be 31 and used in MLCX compression scheme. This value is acknowledged to change for other genomic datasets.

Table 3.5: Compression results for YH genome using KOREF_20090224 as reference with original extended alphabet preserved. $\mathbb{C}$ denotes the result of the encoded genome using basic MLCX encoding. $\mathbb{L}(\mathbb{C})$ denotes result of applying LZMA2 compression on $\mathbb{C}$.|X| denotes the length or size of X in bytes.

Chromosome	Size of Chromosome (bytes)	MLCX		GRS [17] (bytes)	GReEn [18] (bytes)
		$\lvert\mathbb{C}\rvert$ (bytes)	$\lvert\mathbb{L}(\mathbb{C})\rvert$ (bytes)		
1	247,249,719	4,848,860	2,618,422		2,349,124
2	242,951,149	3,985,810	1,431,230		2,420,007
3	199,501,827	3,437,153	1,230,786	17,410,946	1,730,477
4	191,273,063	3,546,960	1,257,452		1,877,056
5	180,857,866	3,086,822	1,104,082		1,792,278
6	170,899,992	3,243,274	1,149,085	25,815,446	1,588,739
7	158,821,424	2,958,977	1,059,713		1,820,425
8	146,274,826	2,590,052	915,747		1,358,770
9	140,273,252	2,293,134	836,928		1,476,495
10	135,374,737	2,383,251	848,497		1,353,193
11	134,452,384	2,469,256	874,995		1,274,433
12	132,349,534	2,352,218	807,042	16,136,610	1,174,966
13	114,142,980	1,687,363	602,282	11,227,954	866,266
14	106,368,585	1,648,189	563,406		826,672
15	100,338,915	1,460,972	521,666		892,429
16	88,827,254	1,560,767	547,736		1,015,246
17	78,774,742	1,303,343	470,271		864,710
18	76,117,153	1,372,841	485,903	13,187,892	713,787
19	63,811,651	1,146,510	411,414		589,422
20	62,435,964	1,115,182	387,408	8,409,776	493,404
21	46,944,323	795,214	275,232	726,269	374,383
22	49,691,432	756,235	262,401		444,932
X	154,913,754	1,637,742	583,009		3,258,188
Y	57,772,954	376,380	119,850		859,688
M	16,571	591	508	321	127
Sum	**3,080,436,051**	**52,057,096**	**19,365,065**	**92,915,214**	**31,415,217**

MLCX compression method uses the LPF data structure to represent the CSS's in the target. A left-to-right scan of LPF data was done choosing leftmost CSS in target (say $T[i \ldots i+l-1]$, where l is length of LPF[i]). A decision is made whether to encode this CSS as either a triple or a symbol. The next CSS is positioned at target, $T[i+l \ldots i+l+LPF[i+l]-1]$ of length $LPF[i+l]$ if previous CSS is encoded as a triple otherwise positioned at target, $T[i+1 \ldots i+LPF[i+1]]$ of length $LPF[i+1]$. Obviously, it is better to encode a length $(l = 1)$ CSS with a 1-byte symbol, rather than a 12-byte triple. It is clearly the case that for any CSS length $1 \leq l < 12$, since it is

better to encode the first symbol with 1-byte and take a chance that the next CSS to the right will be significantly larger. Why can we afford to take this chance? One LPF property, which also allows for an efficient construction of the data structure is that $LPF[i+1] \geq LPF[i] - 1$. That is, if we pass upon encoding the CSS at *i* of length $(LPF[i] = l)$ as a triple, we can encode $T[i]$ as a symbol and (1) are guaranteed that there is at least a length $(l-1)$ CSS with a prefix of $T[i+1 \ldots n]$ and (2) the longest CSS common to a prefix of $T[i+1 \ldots n]$ is of length $LPF[i+1]$, maybe even larger than $LPF[i]$. Clearly, we want to encode most CSSs as triples to take advantage of the concise triple representation. Now, the question becomes: how large should we set *k*, such that we can afford to take a risk passing up length $(l < k)$ CSSs in hopes of finding even larger CSSs, better suited as triples?

3.3.1 Impact in Compression Method

First, we analyze the impact of the *k* parameter on compression. We performed compression on a few chromosomes in *Arabidopsis thaliana, Oryza sativa* and *Homo sapiens* genome datasets for *k* values ranging from 1 to 200. The empirical analysis was performed and a model was proposed for determining the best *k* value rather than using a fixed *k* parameter. Compression results for fixed and best *k* values are summarized in Table 3.6.

Table 3.6 shows an improvement of 1.87% for *Arabidopsis thaliana*, 12.5% for *Oryza Sativa* and 1.4% for *Homo sapiens* genomic dataset. Undoubtedly, the results indicate that algorithm becomes too optimistic for $k \geq 100$. This reduces the possibility to encode smaller CSS's triples in notion that larger CSS's might exist. In conclusion, *k* should be at least 11 and not so large, that only CSS's which are worthy of encoding into triples are considered.

3.3.2 Predicting the best *k* values

From the analysis as detailed in Section 3.3.1, it is evident that the MLCX compression can output better compression. This improvement in output is attributed to the usage of best-determined *k* value for each chromosome, rather than the using a generic/fixed '*k*-value' for all chromosomes in a genome species. Hence, there arises a need for the creation of an adaptive model to predict the '*k*-value' for any given chromosome in a genomic dataset.

As discussed in section 3.3, *k* parameter determines whether it's beneficial to encode a particular Common SubString (CSS) either as a tuple or symbol. The *k*-value controls whether we encode a given CSS as a triple to gain the benefit of concise triple representation. In triplet encoding, best *k*-value is the optimal length CSS that cannot be neglected for encoding in a notion of finding even larger CSS. *k*-value is dependent on both total number and length of the larger CSS obtained between a given reference and target for compression.

Table 3.6: Results of MLCX compression scheme on *Arabidopsis thaliana* genome, *Oryza sativa* genome, *Homo sapiens* genome for *k*=31 and best *k* range. $\mathbb{C}$ denotes the result of the encoded genome using basic MLCX encoding. |X| denotes the length or size of X in bytes.

Chromosome	Size of Chromosome (bytes)	MLCX Compression (in bytes)		Best Range for *k*- parameter
		$\|\mathbb{C}\|$ (where *k*=31)	$\|\mathbb{C}\|$ (best *k* range)	
Arabidopsis thaliana Genome				
2	19,698,289	504	504	16-200
4	18,585,056	4,555	4,418	18
Sum	119,146,348	**7,324**	**7,187**	-
Oryza sativa Genome				
2	35,930,381	4,645	4,011	11-17
6	31,246,789	12	12	1-200
11	28,512,666	41,407	36,281	13-18
Sum	95,689,836	**46,064**	**40,304**	-
Homo Sapiens Genome				
1	247,249,719	2,836,652	2,805,483	23
8	146,274,826	1,617,884	1,603,437	24
18	76,117,153	865,138	855,557	23
22	49,691,432	568,734	559,764	21
X	154,913,754	7,525,925	7,411,675	22
Y	57,772,954	1,343,260	1,305,661	19
Sum	732,019,838	**14,757,593**	**14,541,577**	-

One in many ways to evaluate the behavior of CSS between two strings is to assess Longest Common Prefixes (LCPs) amongst all pairs of consecutive suffixes in a suffix array. The suffix array is a sorted array considering all suffixes of a string. LCP array stores shared CSS lengths between two suffixes calculated using the KASAI et al. algorithm [41]. For instance, LCP array for S= "banana" has common prefixes "ana", "na" and "a" between suffices for S with corresponding lengths 3, 2 and 1. The *k*-value essentially depends on the longest independent CSS blocks (i.e., "ana" in case of S= "banana") but not on all the common prefixes. These independent CSS blocks were acquired from LCP array by first sorting the calculated LCP array values in descending order and then each sorted CSS is marked on $Z=T^{\circ}R$ to obtain only CSS block lengths with no repeated common prefixes. Here '$^{\circ}$' denotes concatenation.

The statistical behavior of independent CSS pave the way to finding the best *k*-value. In MLCX compression model statistics like mean, standard deviation, co-efficient of variation (CV), entropy and median are used to predict the *k*-value best suited for a particular genome sequence in compression. Few genomic sequences of *Arabidopsis thaliana*, *Oryza sativa*, and

Homo sapiens datasets were chosen as training data. Regression analysis was then performed on training data to determine the predicted $\hat{k}$ parameter. We used the following equation for linear regression,

$$\hat{k} = \beta_0 + \beta_1 \mu + \beta_2 \left(\frac{\sigma}{\mu}\right)$$

where μ is the mean , σ is the standard deviation , β_0 is the intercept , β_1 is the regression coefficient for μ (the mean) and β_2 is the regression coefficient for $\frac{\sigma}{\mu}$ (the coefficient of variation). Table 3.7, illustrates computed mean (μ), standard deviation (σ), coefficient of variation (CV = $\frac{\sigma}{\mu}$) of the independent CSS's for the selected reference chromosomes in each genome dataset.

Table 3.7: Statistics on training set belonging to *Arabidopsis thaliana* genome, *Oryza sativa* genome, *Homo sapiens* genome.

Chromosome	Size of Chromosome (bytes)	Mean (μ)	Standard Deviation(σ)	Co-efficient of Variation ($\frac{\sigma}{\mu}$)
Arabidopsis thaliana Genome				
2	19,698,289	6.47	9.24	1.43
4	18,585,056	6.55	21.26	3.25
Oryza sativa Genome				
2	35,930,381	7.30	20.76	2.84
6	31,246,789	7.59	26.69	3.52
11	28,512,666	7.57	15.60	2.06
Homo sapiens Genome				
1	247,249,719	8.27	3710.89	448.81
8	146,274,826	7.47	680.51	91.05
18	76,117,153	7.128	417.89	58.63
22	49,691,432	10.07	6,495.40	645.28
X	154,913,754	7.541	664.70	88.14
Y	57,772,954	16.61	16,095.15	968.77

From Table 3.7 it is observed that there is significant variation in the statistics from different chromosomes, even for the same genome. Thus, to improve the prediction, different models were developed for groups of chromosomes. Based on calculated CV value, three clusters (CV<10, CV>100 and 10≤CV≤100) were identified. Linear regression on each cluster determines the approximate prediction function and coefficients required to predict the *k*-value. For this cluster based approach, we obtained the following models for the *k*-parameter prediction:

$$\text{CV} < 10 \quad : \hat{k} = \beta_0 + \beta_1 \mu + \beta_2 \left(\frac{\sigma}{\mu}\right)$$

$$0 \leq \text{CV} \leq 100 \quad : \hat{k} = \beta_3 + \beta_4 \mu + \beta_5 \left(\frac{\sigma}{\mu}\right)$$

$$\text{CV} > 100 \quad : \hat{k} = \beta_6 + \beta_7 \mu + \beta_8 \left(\frac{\sigma}{\mu}\right)$$

Here, we obtained β_0 =37.8442, β_1 =-3.9993, β_2 =2.7233, β_3 =3.9138, β_4 =4.0355, β_5 =-0.0042, β_6 = 22.5897, β_7 =1.1965, β_8 = -0.0211.

To evaluate $\hat{k}$, compression outputs obtained using the generic/fixed *k*-value and predicted *k*-value are compared. For *Arabidopsis thaliana* genome, the prediction model degrades the compression by 0.3% but for both *Oryza sativa* and *Homo sapiens* improves by 11.3% and 0.44% respectively over fixed *k*-value compression. Also, required computing resources (i.e. disk space and execution time) for this model are relatively high. Henceforward, it was inferred that rather than prediction, a better approach would be to search for best range of *k*-parameter,which can be determined from the compression results on the training dataset. In the further sections, the best *k*-value range search in combination with new a decomposition technique improves the compression performance using less computing resources.

3.4 Decomposition Technique:

From the study in Section 3.2.2, it was found that the scenario involving character case of alphabet symbol being non-significant yielded a compression ratio of 585 which is much better when compared with the compression ratio of 199 obtained by compressing the original sequence with extended alphabet. It is known that the IUB/IUPAC amino acid and nucleic acid codes use only upper-case letters [43]. Also some environments and formats (such as FASTA) do not distinguish between lower-case and upper-case. According to the NCBI website for BLAST input formats “Sequences [in FASTA format] are expected to be represented in the standard IUB/IUPAC amino acid and nucleic acid codes, with these exceptions: lower-case letters are accepted and are mapped into upper-case; …..”. Further for improved visualization some programs/ environments (e.g. USC Genome Browser) use character-case to show repeats from Repeat Masker and Tandem Repeats Finder [43]. To handle such environments, a new decomposition method is developed with the objective of retaining the letter-case for the alphabet symbols with the better compression results obtained when MLCX compression scheme is applied. This technique is capable of handling non-ATGC characters as well as lowercase characters. This makes it suitable for practical application to real-world sequencing.

The new technique will decompose each chromosomes into two parts: (1) the payload (ρ), representing the chromosome character-case, and (2) the character-case bitstring (α), whereby each bit records whether the corresponding position in the target is an upper-case or lower-case character. Next, we apply MLCX compression algorithm to compress ρ into C_ρ and α into C_α. For implementation, it is beneficial to pack β consecutive bits from bitstring α into a

primitive data type. For improved representation, the payload (ρ) can also be converted into a bitstream, e.g., using fixed-length encoding of the symbols into integers. Thus, similar to the bitstring (α), we pack γ consecutive bits from the payload bitstream into a primitive datatype. Thus, the parameters (γ, β, k) essentially define the encoding scheme. While k affects only the encoding stage after preprocessing to the compute LPF, β and γ have direct impact on both the compression ratio and compression time. Figures 3.3.1 - 3.3.4 and Appendix A show the impact of the parameters (γ, β, k) on the compression performance using the MLCX method. It is observed that the performance varied significantly with these parameters. For the symbol-case bitstring (α), the best overall results were obtained at β= 28 for each chromosome, but at different k values (k_α), for given β. Similarly, for the payload bitstring (ρ), the best results were obtained at γ= 8, and again at different k values (k_ρ)for given .

The MLCX compression algorithm was improved so that after preprocessing, encoding using a given k value is relatively very fast requiring less than 1 second for the longest chromosomes. This motivate the idea for the search of k, for a given method. Let the following functions denote our previous algorithms: (triples; symbols) = MLCX(R; T; k) for reference R, target T, and minimum-encoding length k. One challenge in the experiments is to select a k. The best result can be achieved by executing the algorithms over several k values and then choosing the scheme with the smallest encoded output. That is, we can specify the encoding as follows: $\mathbf{C}$ =MLCX(R; T; k^*); k^*=argmin$_k${MLCX(R; T; k)}. Similarly, we can perform the search on the compressed data: $\mathbf{L(C)}$ =LZMA2(MLCX (R; T; k^*)); k^*=argmin$_k${LZMA2(MLCX(R; T; k))}. Table 3.8 shows the compression results using the improved MLCX algorithm with total compressed size of the genomes rather than the individual chromosomes. The table shows results under different variations of the algorithmic parameters ($\gamma, \beta, k_\rho, k_\alpha$).

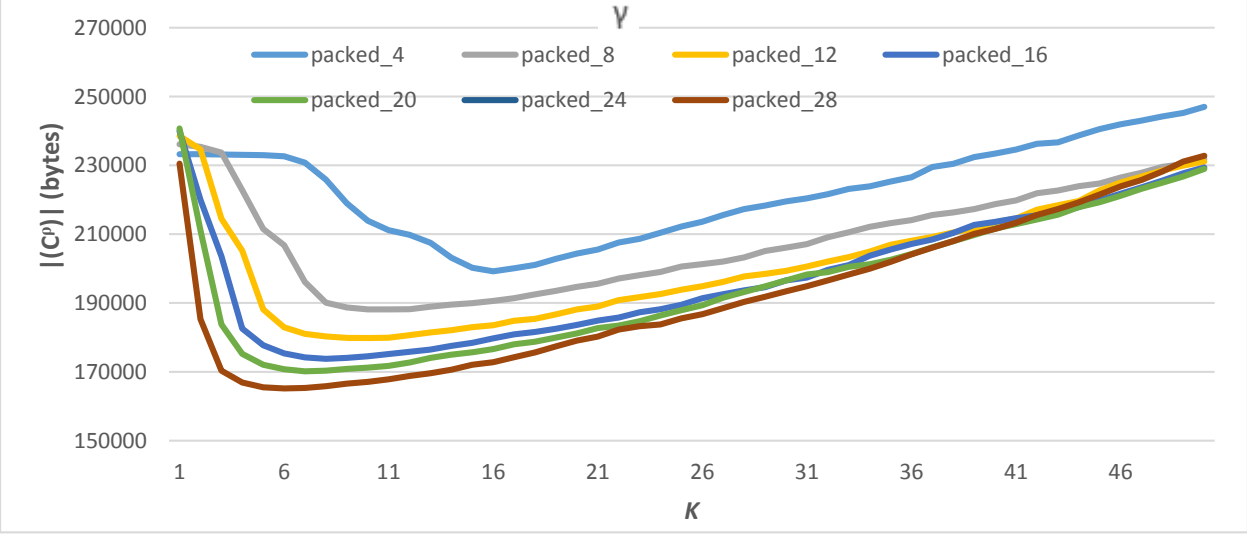

Figure 3.3.1 *Homo sapiens* chromosome 13: MLCX Encoding payload, ρ.

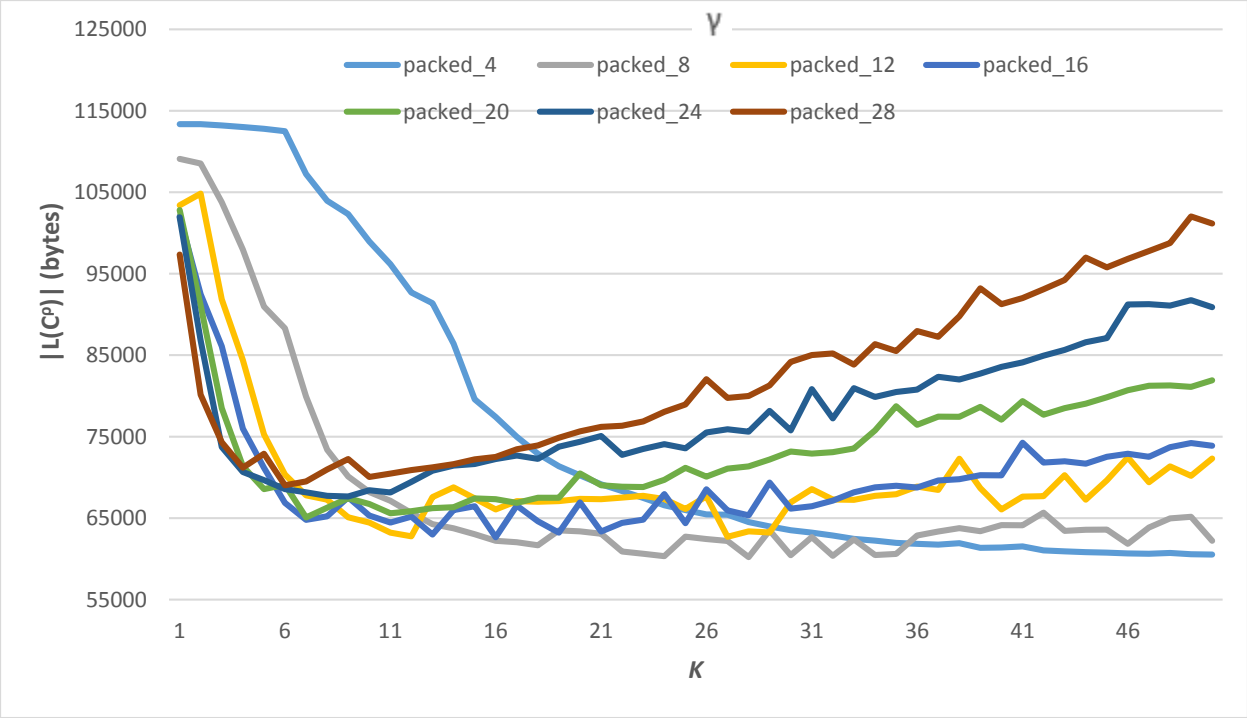

Figure 3.3.2 *Homo sapiens* chromosome 13: Using MLCX then LZMA2 to compress the encoded payload, (C^{ρ}).

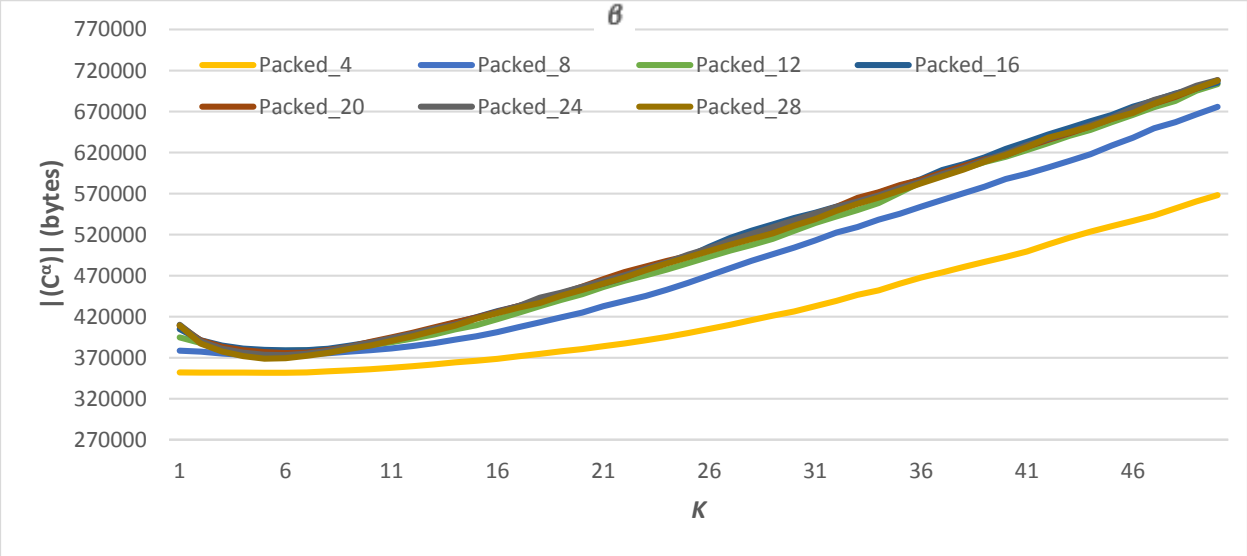

Figure 3.3.3 *Homo sapiens* chromosome 13: MLCX Encoding character-case bitstring, *α.*

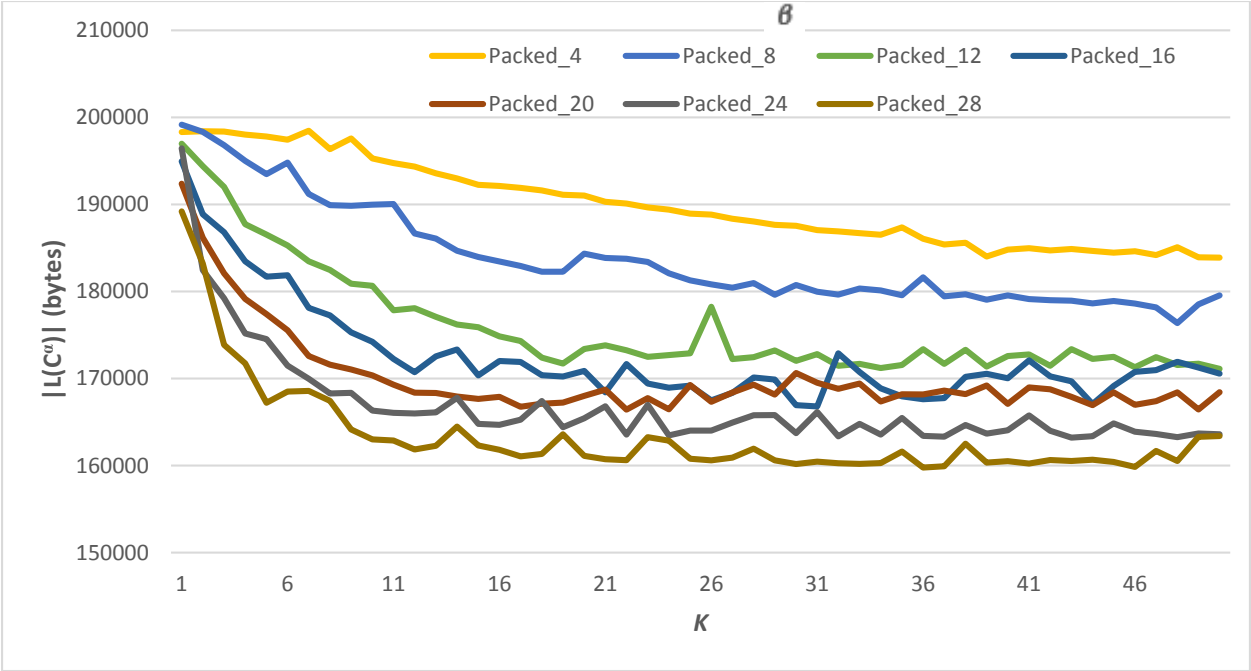

Figure 3.3.4 *Homo sapiens* chromosome 13: Using MLCX then LZMA2 to compress the encoded character-case bitstring, (C^{α}).

Table 3.8: Results (in bytes) for compressing the complete *Homo sapiens* genome $\mathbb{U}$ using the improved MLCX with different parameter variations γ, β, kρ,kα. $\mathbb{C}$ denotes the result of the encoded genome using improved MLCX encoding. $\mathbb{L}(\mathbb{C})$ denotes result of applying LZMA2 compression on $\mathbb{C}$. |X| denotes the length or size of X in bytes.

Parameters				Total $\|\mathbb{U}\|$	Improved MLCX Encoding $\|\mathbb{C}\|$		Improved MLCX compression $\|\mathbb{L}(\mathbb{C})\|$		Total $\|\mathbb{L}(\mathbb{C}^{\rho})\|+\|\mathbb{L}(\mathbb{C}^{\alpha})\|$
γ	β	kρ	kα		Total $\|\mathbb{C}^{\rho}\|$	Total $\|\mathbb{C}^{\alpha}\|$	Total $\|\mathbb{L}(\mathbb{C}^{\rho})\|$	Total $\|\mathbb{L}(\mathbb{C}^{\alpha})\|$	
8	28	22-34	21-46	3,080,436,051	7,289,776	20,549,726	2,108,097	5,605,288	**7,713,385**
20	28	8-15	21-46	3,080,436,051	5,997,460	20,549,726	2,223,688	5,605,288	7,828,976
28	28	4-10	21-46	3,080,436,051	5,766,916	20,549,726	2,379,526	5,605,288	7,984,814

Table 3.9 shows the compression results using the improved MLCX algorithm by packing $\beta = 28$ consecutive bits from bitstring (α) and packing $\gamma = 8$ consecutive bits from the payload (ρ). The ranges for kρ and kα were chosen to be 22-34 and 21-46 respectively by investigating the graphs in Figures 3.3.1 - 3.3.4 and Appendix A that depict influence of *k* both on the compressing bitstring and compressing the payload. The table also shows the comparison of the improved MLCX method against the state-of-the-art methods GRS and GreEn. The chromosome mitochondria DNA (M) was not further compressed using LZMA2 and handled as mentioned in the section 3.2.2. The improved MLCX compression produced significant

improvement with compression ratio of 399 over the original MLCX method with compression ratio of 199.

Table 3.9: Compression results for *Homo sapiens* genome using KOREF_20090224 as target and KOREF_20090131 as reference using the improved MLCX. ($\gamma = 8, \beta = 28$, kρ=[22-34], kα=[21-46])

Chromosome	Size of Chromosome (bytes)	Improved MLCX (bytes)	GRS [17] (bytes)	GReEn [18] (bytes)
1	247,249,719	546,657	1,336,626	1,225,767
2	242,951,149	543,557	1,354,059	1,272,105
3	199,501,827	416,949	1,011,124	971,527
4	191,273,063	472,449	1,139,225	1,074,357
5	180,857,866	424,557	988,070	947,378
6	170,899,992	377,809	906,116	865,448
7	158,821,424	429,718	1,096,646	998,482
8	146,274,826	320,016	764,313	729,362
9	140,273,252	369,472	864,222	773,716
10	135,374,737	327,461	768,364	717,305
11	134,452,384	314,419	755,708	716,301
12	132,349,534	295,639	702,040	668,455
13	114,142,980	219,995	520,598	490,888
14	106,368,585	205,825	484,791	451,018
15	100,338,915	207,114	496,215	453,301
16	88,827,254	231,513	567,989	510,254
17	78,774,742	207,982	505,979	464,324
18	76,117,153	170,949	408,529	378,420
19	63,811,651	169,397	399,807	369,388
20	62,435,964	128,469	282,628	266,562
21	46,944,323	99,924	226,549	203,036
22	49,691,432	110,924	262,443	230,049
X	154,913,754	934,563	3,231,776	2,712,153
Y	57,772,954	187,584	592,791	481,307
M	16,571	443(*)	183	127
Sum	**3,080,436,051**	**7,713,385**	**19,666,791**	**17,971,030**

In the next Chapter, a new and improved compression scheme is proposed by making some key modifications in the MLCX algorithm. Test results for new scheme on three representative genome datasets, comparative analysis of compression performance, execution times are documented. The compression and decompression times for the decomposition technique with pack bits are discussed.

Chapter 4

New Compression Scheme

Motivated by the MLCX compression, a new compression scheme is proposed in this chapter. Compression and comparative analysis for this new method is tested on three genome datasets namely *Arabidopsis thaliana*, *Oryza sativa*, *Homo sapiens* are discussed.

The backbone of the MLCX method lies in the total number and lengths of the common substring (CSS's) chosen to encode between target and reference sequences for compression. The compression improves by encoding the longer CSS's as triplets rather than encoding individual symbols. The major drawback in MLCX scheme is the sequential scan on both LPF and POS (position) data structures [19] to choose CSS for encoding. Also, the chosen CSS is encoded as either triplet or symbol based on the *k* value. Due to this procedure, two short CSS's may be encoded rather than one longer CSS. This means more bytes are used to encode the substrings which reduces the compression efficiency.

To address this limitation, a new compression scheme (SMLCX) is proposed in this study. SMLCX stands for Sorted Maximal Longest Common Substring/Subsequence. The SMLCX method selects and encodes, the longest CSSs of length at least *minimal encoding length* (*k)* first. The CSSs are sorted based on the LPF and POS data structures and then the longest CSSs are chosen instead of the left to right directive scan. Unlike the MLCX which uses two output files (symbols and triples), in this method only one output file 'words' is used to write with the long words. Compression and decompression methodologies for SMLCX are described in the sections in this chapter.

4.1 SMLCX Methodology

In compression via the LPF, we know the sizes of substrings that may be encoded, but we need a policy to tell us which substrings should be selected/encoded. MLCX algorithm selects the substrings sequentially, left-to-right from the LPF. When compressing the factor matching a prefix of target T[i...n], we encode this l-length factor only if $l \geq k$. Otherwise, we encode the (l =1) length symbol T[i]. Then, we consider compressing a prefix of the uncompressed T[i + l...n]. An important observation is that we can view the LPF as a mechanism that gives us a variety of ways to efficiently store target (T) in a contiguous manner on disk. The notion of an LPF selection policy naturally leads to exploring compression algorithms with different policies, motivating the following technique.

One limitation of the MLCX method is embedded in the sequential left-to-right scan policy, which is historically the policy with LPF processing. The problem is: factors are selected

in a left-to-right way and so, two leftmost short factors may be used to encode a substring rather than one significantly longer factor. To resolve this issue, we propose a new policy: the Sorted MLCX (SMLCX). The SMLCX will not use the LPF in the traditional sequential way and instead, we will select the longest such factors available to encode the substrings of the target. The challenges here are due to the fact that we are not processing factors left-to-right. By selecting the longest factors available, (1) we require bookkeeping to guarantee that each symbol is compressed exactly once and (2) we cannot immediately write the (likely) unordered compressed encodings to file, since storing encodings in a left-to-right way reduces the amount of information that is encoded per factor and allows for efficient decompression. These were not challenges needed to be addressed by MLCX.

Like MLCX, we will only encode factors of length at least k, the minimal-encoding length. Unlike the MLCX, the compressed output of the SMLCX will be one file named words. The challenge is that both the (a) factor encodings and (b) singletons, i.e. individual symbol encodings, will exist in the same file. Since the encoded data is in one file, we only need to represent factors with two pieces of information: (1) the position of the factor in the dictionary and (2) the length of the factor. Symbols can simply be encoded with their negated integer representation so that we can distinguish between symbols and encoded factors in words. Below, we detail the compression and decompression.

4.1.1 Compression

(1) Given the m-length reference R and the n-length target T, let $Z = R \circ T$ and compute $(LPF_Z , POS_Z) = lpfpos(Z)$.

(2) We collect the LPF/POS information for T by creating P. Each $P[i] = (pos, len)$, for 1 ≤ i ≤ n, where $pos = i + m$ and $len = LPF_Z[i + m]$. We sort P by the len attribute.

(3) Now, we define a way to record these encodings as they would appear in the target T. We use an n-length working space W to record (a) whether a symbol was already encoded and (b) the factor encoding information. As we choose to compress factors, we will update W. We set $W[i] = 0$ to signify that the symbol $T[i]$ was not yet compressed. We set $W[i] = (dpos, len)$ to signify the start of a length-len factor at position $dpos$ in the dictionary. For other symbols encoded by the factor, we set $W[i] = 1$. Initially, $W[1...n] = \{0, ..., 0\}$, signifying that no symbols in T are compressed.

(4) Let i = n (length of target), where $P[i]$ is the longest factor length.

(5) If $P[i].len \geq k$, then we consider the factor with info $P[i]$. Otherwise, go to (6).

(a) When $W[P[i].pos] \neq 0$, then W is already part of a longer factor, so we disregard this factor and consider the next longest factor: decrement i and go to (5).

(b) Otherwise, $W[P[i].pos]$ is not part of a current factor and we must verify that the factor has a prefix of sufficient length not already encoded. Find the leftmost position q, $P[i].pos \leq q \leq P[i].pos + P[i].len - 1$, in W with $W[q] \neq 0$. If q does not exist, we can encode the entire factor; set $len^1 = P[i].len$. Otherwise, only $W[P[i].pos \ldots q-1]$ can be encoded. Adjust the factor length: $len^1 = q - P[i].pos$. If $len1 < k$, then the factor is too short to be encoded, so, go to († [2]). Otherwise, the factor is long enough to encode. So, we set the information for the start of the factor: $W[P[i].pos] = (POS[P[i].pos], len^1)$ and mark the other symbols in the factor as encoded: set $W[j] = 1$ for $P[i].pos + 1 \leq j \leq P[i].pos + len^1 - 1$. Note that each $W[j] = 1$ prevents the respective symbol from being encoded in another factor. († [2]) Decrement i and go to (5).

(6) Lastly, we need to write the encodings of target T to the words file. Let i = 1.

(a) If $W[i] = 0$, then this symbol was not encoded by any factor, so we must encode the individual $T[i]$. So, write to words the integer $-int(T[i])$ and increment i. Otherwise, $W[i]$ will be a pair $(dpos, len)$, so we will write two integer words: the position of the factor in $Z[1 \ldots i-1]$ $(W[i].dpos)$, and the factor length $(W[i].len)$. Set $i = i + W[i].len$. If $i \leq |T|$, go to (6a). Otherwise, the complete T is compressed in words.

Now, we discuss the running time. The time required to compute (LPF_Z, POS_Z) is in $O(m + n)$ [45]. Since R and T are sequences for the same chromosome, we can assume that $m \in O(n)$ and so, constructing (LPF_Z, POS_Z) requires $O(n)$ time. The time required to then sort the LPF information of length-n is in $O(n \log n)$.

Given the LPF/POS information in sorted form, the running time of SMLCX is bounded by $O(nk)$. Since each element of the n-length working space W is processed to write the words file, we need at least $\Omega(n)$ time. In the worst-case execution, we will have all $LPF_Z[i] = k$, $m + 1 \leq i \leq m + n$. By choosing the first factor, we will require $O(k)$ time to set W; this choice is a successful selection. Then, the next factors chosen will have overlapping conflicts with the previously set factors, each requiring $O(k)$ time to scan W and deem an unsuccessful selection. In the worst case, we will need to process each of the $O(n)$ potential factors, each requiring $O(k)$ time. Thus, $O(nk)$ time is required. Another way to analyze the complexity is to observe that successful selections will require a total of $O(n)$ work and each of the potentially $O(n)$ unsuccessful selections will require at most $O(k)$ time. The worst-case running time is formalized as: Given a reference text R, an n-length target text T, sorted LPF/POS information on $R \circ T$, and a minimum encoding length k, the SMLCX compresses T with respect to R in time bounded by $\Omega(n)$ and $O(nk)$. If we run SMLCX on the same R and T for other choices of k, each execution will require $O(nk)$ time since the sorted LPF/POS information can be reused.

4.1.2 Decompression

Since SMLCX stores the encodings in left-to-right order with respect to their appearance in the uncompressed target *T*, decompression is a simple left-to-right scan of words. The task is to reconstruct *Z*, since the uncompressed target is then *T* = Z[m*+1...m+n*]. Note that *Z* does not need to be physically constructed during decompression. We only refer to *Z* for ease of discussion since: *Z*[*i*] = *R*[*i*], if *1 ≤ i ≤ m T*[*i – m*], if *m + 1 ≤ i ≤ m + n* Let *Z = R* and *i = m + 1*. († [3]) We retrieve the next word *w1* from words. If *w1 < 0,* this represents an encoded symbol, so, set *Z[i] = −w1* and increment *i.* Otherwise, we get the next word *w2*, and then set *Z*[*i...i + w2 – 1*] = *Z*[*w1...w1 + w2 – 1*] (these symbols are set sequentially from left-to-right). Then, let *i = i + w2*. Lastly, if *i ≤ m + n* then go to (†[3]). Otherwise, the uncompressed target is *T = Z*[*m + 1...m + n*]. Clearly, the running time of the MLF decompression algorithm is linear in the length of the uncompressed target T. Given an n-length target text *T* compressed by SMLCX with respect to a reference text R, the decompression of T requires *O(n)* time.

4.2 Compression Results

For consistency and comparison with earlier work, the proposed SMLCX compression scheme is implemented to compress three similar biological genomic datasets – *Arabidopsis thaliana*, *Oryza sativa*, *Homo sapiens*. In the dataset of *Homo sapiens*, two different assemblies, KOREF and YH are analyzed. The experiments were conducted on the AWS platform with EC2 instance embedded with the m4.4x large environment properties.

The impact of the influential *k*-parameter in SMLCX compression is studied by compressing the genome for a range of *k* values in this chapter. Further, compression of the obtained encoded output 'words' file is done using the standard compression scheme LZMA2.

4.2.1 *Arabidopsis thaliana* genome

SMLCX compression of the *Arabidopsis thaliana* genome chromosomes was performed using TAIR 9 as target and TAIR 8 as reference. Dataset comprises of five chromosomes with byte sizes approximately ranging somewhere between 18.5MB to 30.5MB. Generally, the chromosomes in this dataset contain alphabet sizes of 11 symbols. Nevertheless, chromosomes 4 and 5 have alphabet sizes of 7, 5 respectively.

To find the best *k*, close inspection of compression results for chromosome 1 with varying *k* values is done. From Figure 4.1, best *k* parameter value is observed at 9. Therefore, we choose *k=9* for this compression technique. Table 4.1 shows the compression results for all the chromosomes in this genome dataset.

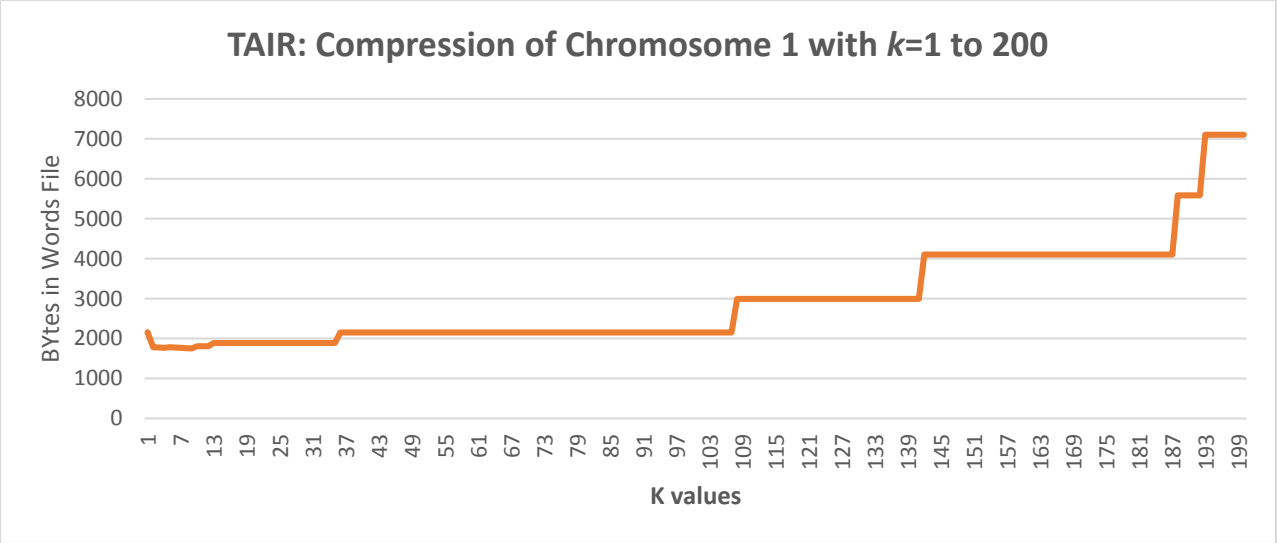

Figure 4.1: SMLCX Compression of chromosome 1 of TAIR genome for different *k* values

Table 4.1 SMLCX Compression results for the *Arabidopsis thaliana* genome alphabet Results for TAIR9 genome using TAIR8 as reference.
$\mathbb{C}$ denotes the result of the encoded genome using SMLCX encoding.
$\mathbb{L}(\mathbb{C})$ denotes result of applying LZMA2 *c*ompression on $\mathbb{C}$.|X| denotes the length or size of X in bytes

Chromosome(C)		SMLCX		MLCX [19] (bytes)	GRS [17] (bytes)	GReEn[18] (bytes)
#	Size in bytes	$\lvert\mathbb{C}\rvert$ (bytes)	$\lvert\mathbb{L}(\mathbb{C})\rvert$ (bytes)			
1	30,427,671	876	784	963	715	1,551
2	19,698,289	416	457	584	385	937
3	23,459,830	620	606	759	2,989	1,097
4	18,585,056	3,812	2,227	2,507	1,951	2,356
5	26,975,502	336	395	502	604	618
Sum	119,146,348	6,060	**4,469**	5,315	6,644	6,559

It is evident that SMLCX scheme showed an improvement of 15.9% over the MLCX scheme for this genomic dataset. All of the chromosomes in this genome dataset are competitive with the GRS and GReEn systems, except for chromosome 4. We attribute this to chromosome 4 having the smallest average CSS length of about 326K, followed by chromosome 3 (≈455K), chromosome 1 (≈458K), chromosome 2 (≈510K), and chromosome 5 (≈1,704K). Overall, SMLCX compressed all the chromosomes in *Arabidopsis thaliana* genome in less bytes, when compared with GRS, GReEn and MLCX scheme.

4.2.2 *Oryza sativa* Genome

SMLCX compression is tested on the *Orzya sativa* genome using TIGR6.0 as reference and TIGR5.0 as target. For this test, *k* is chosen to be 9. Compression results for each chromosome are shown in Table 4.2.

Table 4.2: SMLCX Compression results for *Orzya sativa* genome alphabet results for TIGR5.0 genome using TIGR6.0 as reference. $\mathbb{C}$ denotes the result of the encoded genome using SMLCX encoding. $L(\mathbb{C})$ denotes result of applying LZMA2 *c*ompression on $\mathbb{C}$. |X| denotes the length or size of X in bytes

Chromosome(C)		SMLCX		MLCX [19] (bytes)	GRS [17] (bytes)	GReEn[18] (bytes)
#	Size in bytes	$\lvert\mathbb{C}\rvert$ (bytes)	$\lvert L(\mathbb{C})\rvert$ (bytes)			
1	43,268,879	14,344	5,626	4,735	1,502,040	4,972
2	35,930,381	4,592	1,939	1,649	1,409	1,906
3	36,406,689	52,460	19,107	15,693	47,764	17,890
4	35,278,225	20,724	7,902	6,636	36,145	6,750
5	29,894,789	16,652	6,180	5,431	6,177	5,539
6	31,246,789	8	8(*)	12(*)	14	482
7	29,696,629	5,536	2,384	2,064	4,067	2,448
8	28,439,308	20,068	7,970	8,794	118,246	9,507
9	23,011,239	8	8(*)	12(*)	14	366
10	23,134,759	170,828	59,439	49,713	788,542	60,449
11	28,512,666	38,776	14,434	13,006	2,397,470	14,797
12	27,497,214	8	8(*)	12(*)	14	429
Sum	**372,317,567**	**344,004**	**125,005**	**107,757**	**4,901,902**	**125,535**

As seen in the MLCX scheme the results for the chromosomes 6,9,12 were bloated if further compression of the words file is done using LZMA2. This problem was handled as discussed in section 3.2.1. The SMLCX method results were comparable to GReEn and MLCX method. The GRS method was not competitive on this task. The overall best result on this genome was obtained with MLCX.

4.2.3 *Homo sapiens* Genome- KOREF

We conducted experiments compressing the Homo sapiens genome with KOREF assembly using KOREF_20090224 as the target and KOREF_20090131 as the reference. For this test, *k* is chosen to be 9. Compression results for each chromosome are shown in Table 4.3. As seen earlier, the results were comparable to MLCX method with improvement over GRS, GReEn. The mitochondria DNA chromosome M results were bloated if further compression of the words file is done using the LZMA2 and was handled in the similar way discussed in section 3.2.2.

Table 4.3: SMLCX Compression results while preserving the original extended *Homo sapiens* genome alphabet results for KOREF_20090224 genome using KOREF_20090131 as reference. $\mathbb{C}$ denotes the result of the encoded genome using SMLCX encoding. $\mathbb{L}(\mathbb{C})$ denotes result of applying LZMA2 compression on $\mathbb{C}$. |X| denotes the length or size of X in bytes.

Chromosome(C)		SMLCX		MLCX [19] (bytes)	GRS [17] (bytes)	GReEn[18] (bytes)
#	Size in bytes	|$\mathbb{C}$| (bytes)	|$\mathbb{L}(\mathbb{C})$| (bytes)			
1	247,249,719	351,577	252,420	161,319	1,336,626	1,225,767
2	242,951,149	336,526	234,220	153,805	1,354,059	1,272,105
3	199,501,827	261,112	115,265	119,348	1,011,124	971,527
4	191,273,063	302,436	134,034	137,301	1,139,225	1,074,357
5	180,857,866	237,192	106,856	109,768	988,070	947,378
6	170,899,992	243,076	108,620	110,544	906,116	865,448
7	158,821,424	267,912	118,502	121,289	1,096,646	998,482
8	146,274,826	204,532	91,669	93,378	764,313	729,362
9	140,273,252	272,488	119,530	132,957	864,222	773,716
10	135,374,737	222,700	98,705	103,115	768,364	717,305
11	134,452,384	204,336	90,296	92,471	755,708	716,301
12	132,349,534	196,444	85,890	88,447	702,040	668,455
13	114,142,980	139,004	62,305	62,730	520,598	490,888
14	106,368,585	129,688	56,986	57,354	484,791	451,018
15	100,338,915	126,128	55,301	58,777	496,215	453,301
16	88,827,254	138,732	60,961	62,779	567,989	510,254
17	78,774,742	123,560	55,178	57,030	505,979	464,324
18	76,117,153	105,668	46,786	47,122	408,529	378,420
19	63,811,651	118,016	51,941	53,531	399,807	369,388
20	62,435,964	87,308	38,798	38,689	282,628	266,562
21	46,944,323	66,752	29,065	28,744	226,549	203,036
22	49,691,432	73,904	32,381	33,663	262,443	230,049
X	154,913,754	503,080	209,588	196,868	3,231,776	2,712,153
Y	57,772,954	142,660	58,107	57,002	592,791	481,307
M	16,571	56	56(*)	64(*)	183	127
Sum	**3,080,436,051**	**4,854,887**	**2,313,460**	**2,178,095**	**19,666,791**	**17,971,030**

4.2.4 *Homo sapiens* Genome- YH Vs KOREF

The proposed compression methodology was applied in compressing the Homo sapiens genome using the YH assembly as the target and KOREF_20090224 as reference. A detailed study on the alphabet character-case was not performed for this batch of experiments.

Table 4.4: SMLCX Compression results while preserving the original extended *Homo sapiens* genome alphabet for YH genome using KOREF_20090224 as reference. $\mathbb{C}$ denotes the result of the encoded genome using SMLCX encoding. $L(\mathbb{C})$ denotes result of applying LZMA2 compression on $\mathbb{C}$. |X| denotes the length or size of X in bytes.

Chromosome(C)		SMLCX		MLCX [19] (bytes)	GRS [17] (bytes)	GReEn [18] (bytes)
#	Size in bytes	$\lvert\mathbb{C}\rvert$ (bytes)	$\lvert L(\mathbb{C})\rvert$ (bytes)			
1	247,249,719	4,548,860	2,145,631	2,618,422		2,349,124
2	242,951,149	3,615,608	1,384,772	1,431,230		2,420,007
3	199,501,827	3,121,396	1,187,878	1,230,786	17,410,946	1,730,477
4	191,273,063	3,220,656	1,223,732	1,257,452		1,877,056
5	180,857,866	2,804,072	1,071,502	1,104,082		1,792,278
6	170,899,992	2,949,028	1,112,749	1,149,085	25,815,446	1,588,739
7	158,821,424	2,678,568	1,019,833	1,059,713		1,820,425
8	146,274,826	2,357,436	895,181	915,747		1,358,770
9	140,273,252	2,066,628	794,013	836,928		1,476,495
10	135,374,737	2,161,292	823,306	848,497		1,353,193
11	134,452,384	2,244,832	851,871	874,995		1,274,433
12	132,349,534	2,137,476	816,001	807,042	16,136,610	1,174,966
13	114,142,980	1,547,220	588,776	602,282	11,227,954	866,266
14	106,368,585	1,500,004	571,194	563,406		826,672
15	100,338,915	1,329,604	506,261	521,666		892,429
16	88,827,254	1,408,464	531,196	547,736		1,015,246
17	78,774,742	1,173,280	451,096	470,271		864,710
18	76,117,153	1,259,076	477,600	485,903	13,187,892	713,787
19	63,811,651	1,021,480	391,876	411,414		589,422
20	62,435,964	1,017,300	384,103	387,408	8,409,776	493,404
21	46,944,323	730,600	271,385	275,232	726,269	374,383
22	49,691,432	688,000	259,989	262,401		444,932
X	154,913,754	1,465,676	578,227	583,009		3,258,188
Y	57,772,954	335,052	123,221	119,850		859,688
M	16,571	576	576(*)	508	321	127
Sum	**3,080,436,051**	**47,382,184**	**18,461,969**	**19,365,065**	**92,915,214**	**31,415,217**

For this dataset, improvement of 4.6 % over MLCX method was seen. Significant improvement was seen over GRS and GReEn. The mitochondria DNA chromosome M results were bloated if further compression of the words file is done using the LZMA2 and was handled in the similar way discussed in section 3.2.2.

4.2.5 SMLCX Results using Decomposition Technique

Similar to MLCX method an analysis is performed for packing of bit-level data with decomposition technique to improve the compression of the SMLCX. As seen in Section 3.4, the influence of parameters (γ, β, k_ρ,k_α) on compression throughput is studied. Figures 4.2.1-4.2.4 and Appendix B show the impact of these parameters on the compression performance using the SMLCX method. For the symbol-case bitstring (α), the best overall results were obtain ed at β= 20 for each chromosome, but at different *k* values (k_α), for given β. Similarly, for the payload bitstring (ρ), the best results were obtained at γ= 8, and again at different *k* values (k_ρ) for given γ. Table 4.5 shows the compression results using the improved MLCX algorithm with total compressed size of the genomes rather than the individual chromosomes. The table shows results under different variations of the algorithmic parameters (γ, β, k_ρ,k_α).

Table 4.5: Results (in bytes) for compressing the complete *Homo sapiens* genome $\mathbb{U}$ using the SMLCX with different parameter variations γ, β, k_ρ,k_α.

Parameters				Total $\lvert\mathbb{U}\rvert$ (bytes)	SMLCX Encoding (bytes)		SMLCX compression (bytes)		Total $\lvert\mathbb{L}(\mathbb{C}^\rho)\rvert$+ $\lvert\mathbb{P}(\mathbb{C}^\alpha)\rvert$ (bytes)
γ	β	k_ρ	k_α		Total $\lvert\mathbb{C}^\rho\rvert$	Total $\lvert\mathbb{C}^\alpha\rvert$	Total $\lvert\mathbb{L}(\mathbb{C}^\rho)\rvert$	Total $\lvert\mathbb{P}(\mathbb{C}^\alpha)\rvert$	
8	20	3-100	30-100	3,080,436,051	4,831,508	57,750,988	2,142,475	4,190,013	**6,332,488**
8	28	3-100	6-20	3,080,436,051	4,831,508	15,810,401	2,142,475	5,300,274	7,442,749
28	20	3-50	30-100	3,080,436,051	4,401,756	57,750,988	2,494,039	4,190,013	6,684,052
28	28	3-50	6-20	3,080,436,051	4,401,756	15,810,401	2,494,039	5,300,274	7,794,313

Table 4.6 shows the compression results using the SMLCX algorithm by packing $\beta = 20$ consecutive bits from bitstring (α) and packing $\gamma = 8$ consecutive bits from the payload (ρ). The ranges for k_ρ and k_α were chosen to be 3-100 and 30-100 respectively by investigating the graphs in figures 4.2.1-4.2.4 and Appendix B that depict influence of *k* both on the bitstring and payload. The table also shows the comparison of the SMLCX method against the state-of-the-art methods MLCX, GRS and GreEn. The chromosome mitochondria DNA (M) was not further compressed using LZMA2 and handled as mentioned in the section 3.2.2. The SMLCX compression produced significant improvement with compression ratio of 486 over the improved MLCX method with compression ratio of 399.

Table 4.6: SMLCX Compression results for *Homo sapiens* genome using KOREF_20090224as target and KOREF_20090131 as reference.

Chromosome	Size of Chromosome (bytes)	SMLCX (bytes)	Improved MLCX Compression [from Table 3.9] (bytes)	GRS [17] (bytes)	GReEn [18] (bytes)
1	247,249,719	450,440	546,657	1,336,626	1,225,767
2	242,951,149	446,709	543,557	1,354,059	1,272,105
3	199,501,827	337,298	416,949	1,011,124	971,527
4	191,273,063	376,490	472,449	1,139,225	1,074,357
5	180,857,866	324,210	424,557	988,070	947,378
6	170,899,992	308,658	377,809	906,116	865,448
7	158,821,424	353,782	429,718	1,096,646	998,482
8	146,274,826	261,692	320,016	764,313	729,362
9	140,273,252	306,677	369,472	864,222	773,716
10	135,374,737	268,475	327,461	768,364	717,305
11	134,452,384	255,167	314,419	755,708	716,301
12	132,349,534	241,581	295,639	702,040	668,455
13	114,142,980	182,753	219,995	520,598	490,888
14	106,368,585	167,683	205,825	484,791	451,018
15	100,338,915	169,927	207,114	496,215	453,301
16	88,827,254	190,932	231,513	567,989	510,254
17	78,774,742	170,379	207,982	505,979	464,324
18	76,117,153	140,284	170,949	408,529	378,420
19	63,811,651	139,849	169,397	399,807	369,388
20	62,435,964	104,536	128,469	282,628	266,562
21	46,944,323	81,549	99,924	226,549	203,036
22	49,691,432	134,093	110,924	262,443	230,049
X	154,913,754	758,955	934,563	3,231,776	2,712,153
Y	57,772,954	160,043	187,584	592,791	481,307
M	16,571	326(*)	443(*)	183	127
Sum	**3,080,436,051**	**6,332,488**	**7,713,385**	**19,666, 791**	**17,971,030**

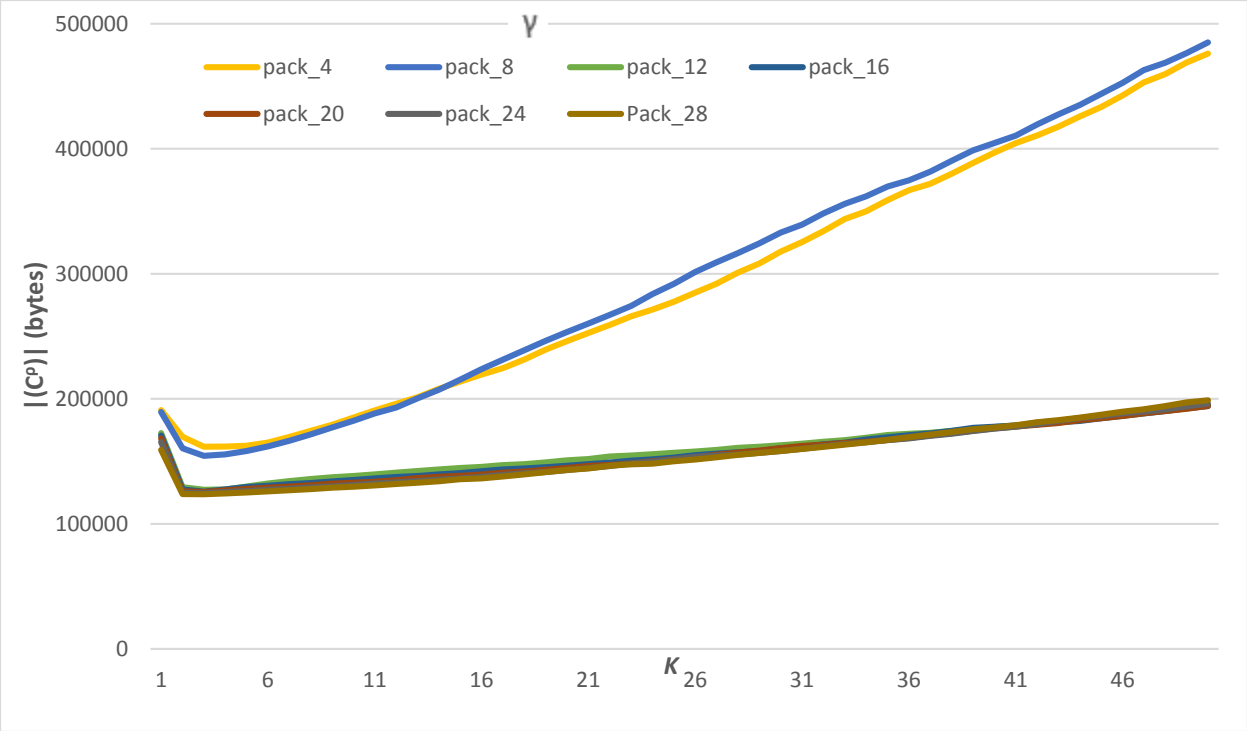

Figure 4.2.1 *Homo sapiens* chromosome 13: SMLCX Encoding payload, ρ.

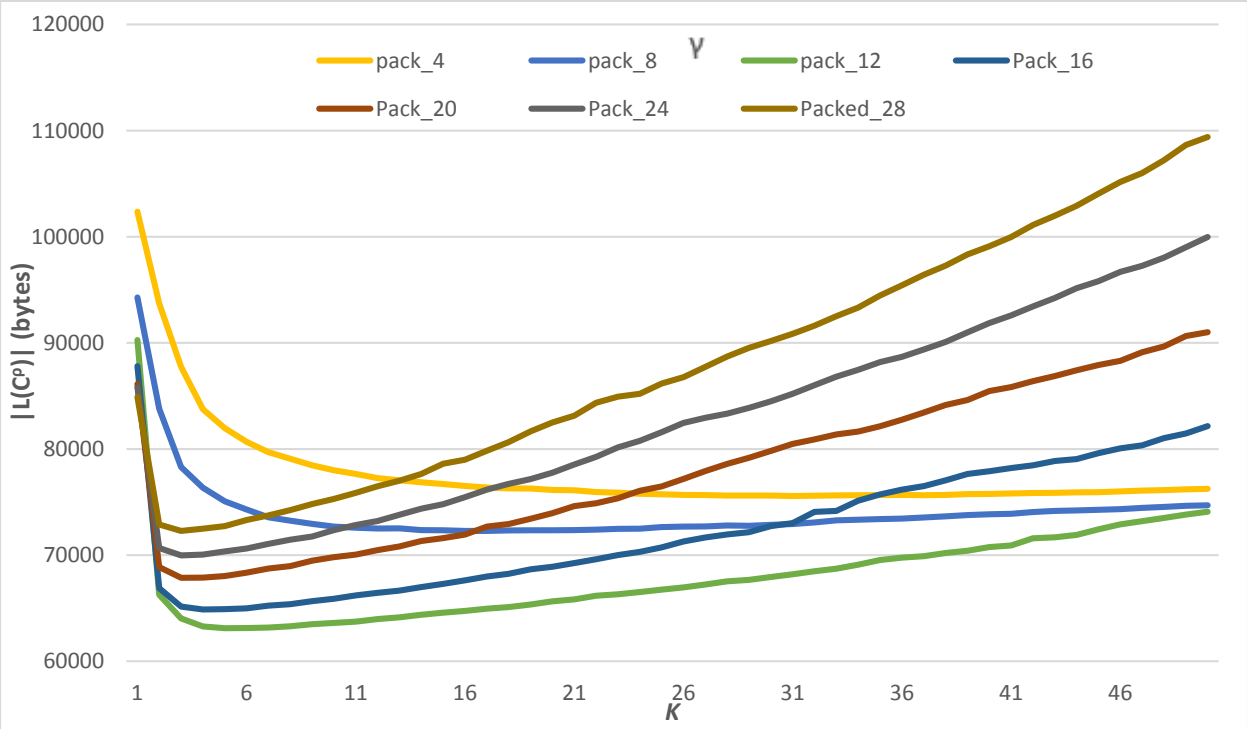

Figure 4.2.2 *Homo sapiens* chromosome 13: Using SMLCX then LZMA2 to compress the encoded payload, (C^{ρ}).

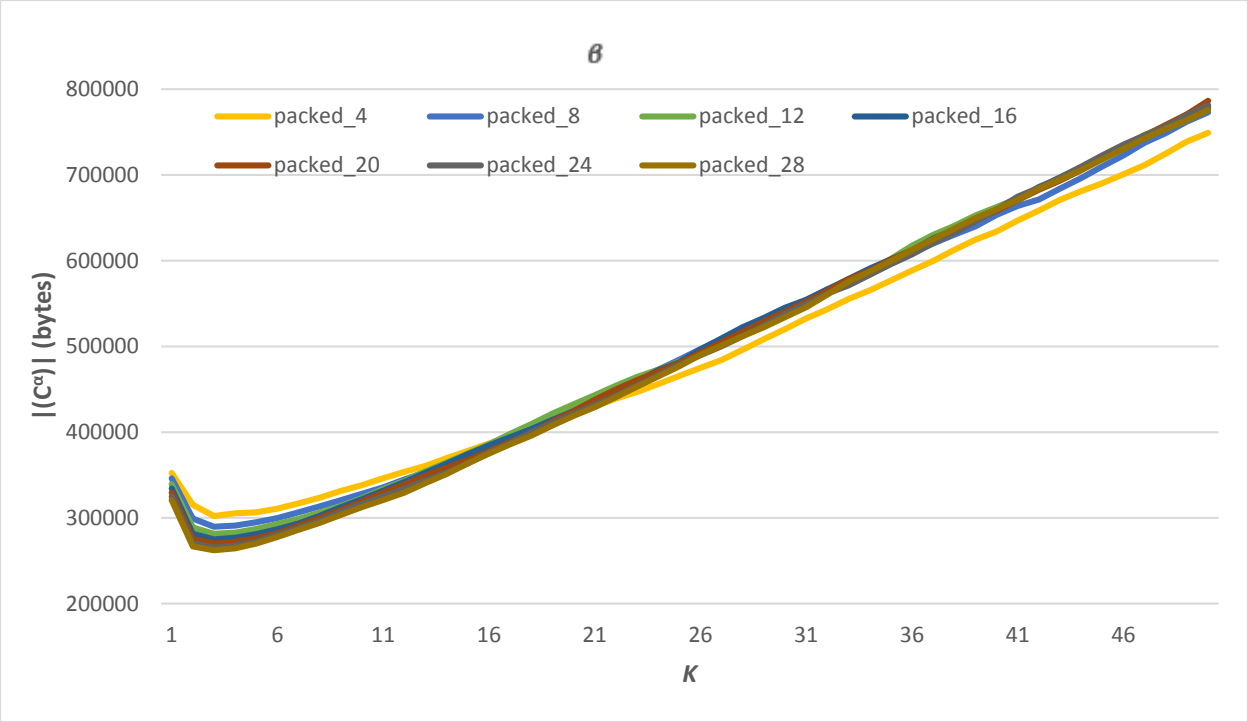

Figure 4.2.3 *Homo sapiens* chromosome 13: SMLCX Encoding character-case bitstring, *α.*

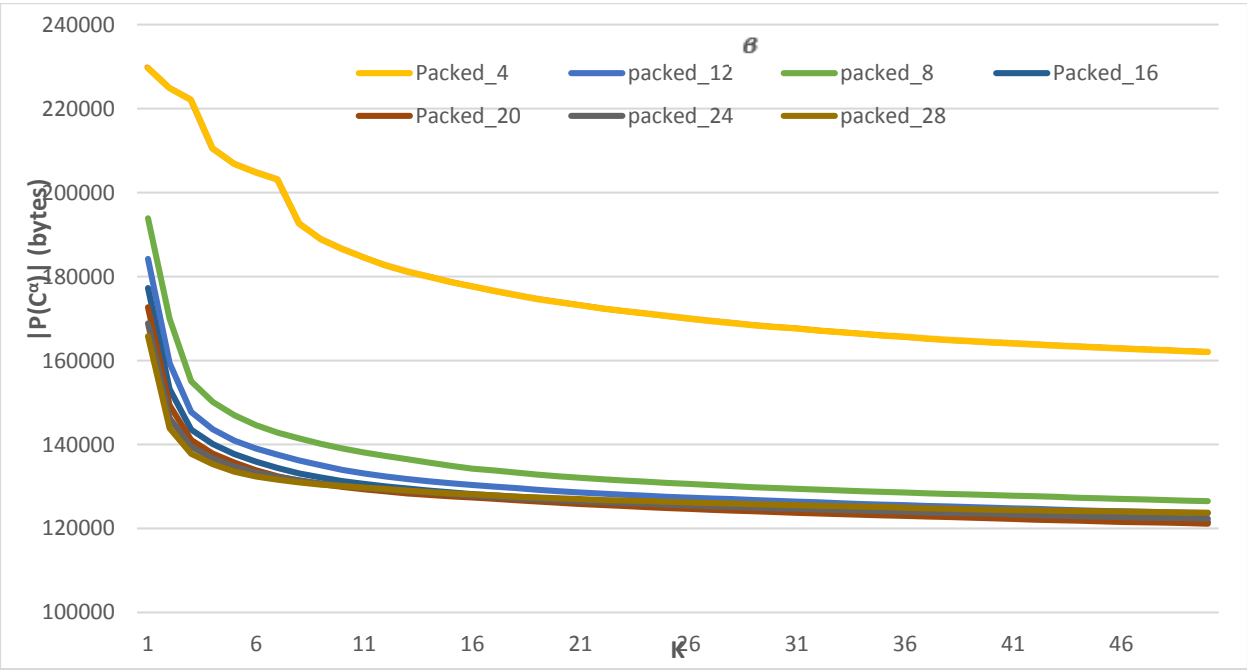

Figure 4.2.4 *Homo sapiens* chromosome 13: Using SMLCX then ppmd to compress the encoded character-case bitstring, (C^{α}).

4.3 Compression Time

We compared the compression time results for the Homo sapiens genome under the decomposition technique. Table 4.7, shows the compression performance, compression times and decompression times.

Table 4.7: Comparison of the compression results for *Homo sapiens* genome using KOREF_20090224 as target and KOREF_20090131 as reference using the SMLCX and improved MLCX methods.

T_C : Compression Time ; T_D : DeCompression Time

#	Size of Chromosome (bytes)	Improved MLCX Method			SMLCX Method		
		Compressed (bytes)	T_C (sec)	T_D (sec)	Compressed (bytes)	T_C (sec)	T_D (sec)
1	247,249,719	546,657	897.21	0.57	450,440	927.98	0.76
2	242,951,149	543,557	922.08	0.56	446,709	960.30	0.74
3	199,501,827	416,949	748.53	0.47	337,298	769.15	0.63
4	191,273,063	472,449	717.84	0.44	376,490	747.05	0.60
5	180,857,866	424,557	677.74	0.42	324,210	702.03	0.56
6	170,899,992	377,809	651.06	0.40	308,658	660.55	0.53
7	158,821,424	429,718	587.26	0.37	353,782	611.46	0.50
8	146,274,826	320,016	542.87	0.35	261,692	559.44	0.46
9	140,273,252	369,472	479.85	0.33	306,677	493.86	0.44
10	135,374,737	327,461	497.28	0.32	268,475	514.80	0.43
11	134,452,384	314,419	475.89	0.31	255,167	494.64	0.41
12	132,349,534	295,639	482.37	0.31	241,581	500.32	0.39
13	114,142,980	219,995	375.94	0.26	182,753	392.19	0.34
14	106,368,585	205,825	349.75	0.25	167,683	360.06	0.32
15	100,338,915	207,114	322.07	0.23	169,927	334.88	0.31
16	88,827,254	231,513	305.05	0.21	190,932	310.18	0.28
17	78,774,742	207,982	282.27	0.19	170,379	291.34	0.26
18	76,117,153	170,949	272.29	0.18	140,284	181.62	0.24
19	63,811,651	169,397	208.10	0.16	139,849	208.50	0.20
20	62,435,964	128,469	214.29	0.16	104,536	224.33	0.19
21	46,944,323	99,924	141.05	0.12	81,549	146.70	0.15
22	49,691,432	110,924	144.30	0.13	134,093	150.94	0.14
X	154,913,754	934,563	589.84	0.43	758,955	622.20	0.83
Y	57,772,954	187,584	143.34	0.15	160,043	159.96	0.21
M	16,571	443(*)	4.71	0.01	326(*)	4.98	0.01
Sum	**3,080,436,051**	**7,713,385**	**11032.98**	**7.33**	**6,332,488**	**11329.46**	**9.93**

While the improved MLCX method resulted in improvements over the state-of-art, the new SMLCX resulted in the best compression for every chromosome, except for Chromosome 22 and mitochondria DNA (M). The compression ratios obtained were 399 for the improved MLCX and 486 for the SMLCX. These can be compared with those from related work, namely, 157 for GRS [17], 171 for GReEn [18] and 360 for MLCX [43].

However, the new SMLCX method required relatively more time for both compression and decompression, when compared with MLCX. The major reason is the sorting stage required by SMLCX in order to select the CSS's with the longest lengths first.

Chapter 5

Conclusion

5.1 Summary

We study the problem of lossless compression of genome resequencing data using a reference based approach. We analyzed various reference based genomic compression algorithms like GRS, GReEn and MLCX. The study conducted on MLCX compression scheme helped to understand the usage mechanism of the common substrings (CSS) for compression in detail. During this study we proposed improved MLCX compression method and a new decomposition technique. Using the proposed methodologies, a significant improvement in compression ratio from 199 to 360 was observed for the *Homo sapiens* genome. For the MLCX compression, the impact of the minimal encoding length the *k*-parameter was also studied in terms of compression size in bytes. Further, we propose a prediction model to predict the influential *k* value prior to compression. Proposed *k*-model further improved the MLCX compression throughput for two genomes namely *Oryza sativa* and *Homo sapiens* by 11.3% and 0.44% respectively.

Our investigation on the nature and structure of the common substrings used by the MLCX compression methodology led to a new compression scheme SMLCX. SMLCX, a new reference-based lossless compression scheme builds on MLCX. This scheme performs compression by encoding common substrings based on a sorted order, which significantly improved compression performance over the original MLCX method. Using SMLCX, we compressed the *Homo sapiens* genome with original size of 3,080,436,051 bytes to 6,332,488 bytes, for an overall compression ratio of 486. This can be compared to performance of current state-of-the-art compression methods, with compression ratios of 157 (Wang et al. [17]), 171 (Pinho et al.[18]) and 360 (Beal et al., [43])

5.2 Future Work

For both improved MLCX and the SMLCX methods, we have used length-*k* CSS's for the compression. One direction for future work would be to identify techniques to improve these algorithms to make use of CSS's of various lengths. A dynamic approach to decide length-*k* based on the structure of the CSS reduces the impact of the influential *k*-parameter value on the compression obtained. Another direction for future work would be a study for prior knowledge of pack bits (β) best suited for a given genome, or a given chromosome. The scope for using the proposed methods in compressing the collections of target genomes using the collection of reference genomes in a specific genome species can also be inspected.

References

[1] F. Sanger and A. R. Coulson, “A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase,” *J. Mol. Biol.*, vol. 94, no. 3, 1975.

[2] T. Bose, M. H. Mohammed, A. Dutta, and S. S. Mande, “BIND - An algorithm for loss-less compression of nucleotide sequence data,” *J. Biosci.*, vol. 37, no. 4, pp. 785–789, 2012.

[3] S. D. Kahn, “On the Future of Genomic Data,” *Science (80-.).*, vol. 331, no. 6018, pp. 728–729, 2011.

[4] D. Adjeroh and F. Nan, “On compressibility of protein sequences,” *Data Compression Conf. Proc.*, pp. 422–434, 2006.

[5] S. Deorowicz and S. Grabowski, “Data compression for sequencing data,” *Algorithms Mol. Biol.*, vol. 8, no. 1, p. 25, 2013.

[6] S. Grumbach and F. Tahi, “A new challenge for compression algorithms: Genetic sequences,” *Inf. Process. Manag.*, vol. 30, no. 6, pp. 875–886, 1994.

[7] E. Rivals *et al.*, “Detection of significant patterns by compression algorithms: the case of approximate tandem repeats in DNA sequences,” *Comput Appl Biosci*, vol. 13, no. 2, pp. 131–136, 1997.

[8] E. Rivals, J. Delahaye, M. Dauchet, and O. Delgrange, “A guaranteed compression scheme for repetitive DNA sequences,” in *Data Compression Conference, 1996. DCC’96. Proceedings*, 1996, p. 453.

[9] T. Matsumoto, K. Sadakane, and H. Imai, “Biological sequence compression algorithms.,” *Genome Inform. Ser. Workshop Genome Inform.*, vol. 11, pp. 43–52, 2000.

[10] X. Chen, M. Li, B. Ma, and J. Tromp, “DNACompress: Fast and effective DNA sequence compression,” *Bioinformatics*, vol. 18, no. 12, pp. 1696–1698, 2002.

[11] G. Manzini and M. Rastero, “A simple and fast DNA compressor,” *Softw. - Pract. Exp.*, vol. 34, no. 14, pp. 1397–1411, 2004.

[12] B. Behzadi and F. Le Fessant, “DNA compression challenge revisited,” *Comb. Pattern Matching Proc. CPM-2005*, vol. 3537, pp. 190–200, 2005.

[13] M. D. Cao, T. I. Dix, L. Allison, and C. Mears, “A simple statistical algorithm for biological sequence compression,” in *Data Compression Conference Proceedings*, 2007, pp. 43–52.

[14] G. Korodi and I. Tabus, “Normalized maximum likelihood model of order-1 for the compression of DNA sequences,” in *Data Compression Conference Proceedings*, 2007, pp. 33–42.

[15] A. J. Pinho, A. J. R. Neves, and P. J. S. G. Ferreira, “Inverted-repeats-aware finite-context models for DNA coding,” in *European Signal Processing Conference*, 2008.

[16] A. J. Pinho, P. J. S. G. Ferreira, A. J. R. Neves, and C. A. C. Bastos, “On the representability of complete genomes by multiple competing finite-context (Markov) models,” *PLoS One*, vol. 6, no. 6, 2011.

[17] C. Wang and D. Zhang, “A novel compression tool for efficient storage of genome resequencing data,” *Nucleic Acids Res.*, vol. 39, no. 7, 2011.

[18] A. J. Pinho, D. Pratas, and S. P. Garcia, “GReEn: A tool for efficient compression of genome

resequencing data," *Nucleic Acids Res.*, vol. 40, no. 4, 2012.
[19] R. Beal, T. Afrin, A. Farheen, and D. Adjeroh, "A new algorithm for 'the LCS problem' with application in compressing genome resequencing data," in *Proceedings - 2015 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2015*, 2015, pp. 69–74.
[20] S. Kuruppu, S. J. Puglisi, J. Zobel, E. Chavez, S. Lonardi, and Eds, "Relative Lempel-Ziv compression of genomes for large-scale storage and retrieval," *String Process. Inf. Retr.*, vol. 6393/2010, pp. 201–206, 2010.
[21] S. Kuruppu, S. J. Puglisi, and J. Zobel, "Optimized relative Lempel-Ziv compression of genomes," *Conf. Res. Pract. Inf. Technol. Ser.*, vol. 113, pp. 91–98, 2011.
[22] L. S. Heath, A. Hou, H. Xia, and L. Zhang, "A genome compression algorithm supporting manipulation," *Proc LSS Comput Syst Bioinform Conf.*, pp. 38–49, 2010.
[23] N. Ma, K. Ramchandran, and D. Tse, "A Compression Algorithm Using Mis-aligned," *IEEE Int. Symp. Inf. Theory*, no. 1, pp. 16–20, 2012.
[24] Z. D. Stephens *et al.*, "Big data: Astronomical or genomical?," *PLoS Biol.*, vol. 13, no. 7, 2015.
[25] R. N. Curnow and T. B. L. Kirkwood, "Statistical Analysis of Deoxyribonucleic Acid Sequence Data--A Review," *J. R. Stat. Soc. Ser. A (Statistics Soc.*, vol. 152, no. 2, pp. 199–220, 1989.
[26] S. Wandelt and U. Leser, "FRESCO: Referential compression of highly similar sequences," *IEEE/ACM Trans. Comput. Biol. Bioinforma.*, vol. 10, no. 5, pp. 1275–1288, 2013.
[27] S. Deorowicz, A. Danek, and M. Niemiec, "GDC 2: Compression of large collections of genomes," *Sci. Rep.*, vol. 5, p. 11565, 2015.
[28] S. Levy *et al.*, "The diploid genome sequence of an individual human," *PLoS Biol.*, vol. 5, no. 10, pp. 2113–2144, 2007.
[29] S. Deorowicz and S. Grabowski, "Robust relative compression of genomes with random access," *Bioinformatics*, vol. 27, no. 21, pp. 2979–2986, 2011.
[30] S. Deorowicz, A. Danek, and S. Grabowski, "Genome compression: A novel approach for large collections," *Bioinformatics*, vol. 29, no. 20, pp. 2572–2578, 2013.
[31] S. Grabowski and S. Deorowicz, "Engineering relative compression of genomes," 2011.
[32] R. Beal and D. Adjeroh, "parameterized longest previous factor," *Theor. Comput. Sci.*, vol. 437, pp. 21–34, 2012.
[33] R. Beal and D. Adjeroh, "Variations of the parameterized longest previous factor," in *Journal of Discrete Algorithms*, 2012, vol. 16, pp. 129–150.
[34] D. Adjeroh, T. Bell, and A. Mukherjee, *The burrows-wheeler transform: Data compression, suffix arrays, and pattern matching*. 2008.
[35] D. Gusfield, "Algorithms on strings, trees, and sequences: computer science and computational biology," *Theory and Practice*, vol. 28, no. 4. p. 554, 1997.
[36] S.-M. Ahn *et al.*, "The first Korean genome sequence and analysis: Full genome sequencing for a socio-ethnic group," *Genome Res.*, vol. 19, no. 9, pp. 1622–1629, 2009.
[37] J. J. Wang *et al.*, "The diploid genome sequence of an Asian individual," *Nature*, vol. 456, no. 7218, pp. 60–5, 2008.
[38] E. Huala *et al.*, "The Arabidopsis Information Resource (TAIR): a comprehensive database and web-based information retrieval, analysis, and visualization system for a model

plant.," *Nucleic Acids Res.*, vol. 29, no. 1, pp. 102–5, 2001.
[39] S. Ouyang *et al.*, "The TIGR Rice Genome Annotation Resource: Improvements and new features," *Nucleic Acids Res.*, vol. 35, no. SUPPL. 1, 2007.
[40] H. B. F. Dixon, H. Bielka, and C. R. Cantor, "Nomenclature for incompletely specified bases in nucleic acid sequences. Recommendations 1984," *Journal of Biological Chemistry*, vol. 261, no. 1. pp. 13–17, 1986.
[41] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park, "Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications," *Proc. 12th Annu. Symp. Comb. Pattern Matching*, pp. 181–192, 2001.
[42] S. Wandelt, M. Bux, and U. Leser, "Trends in Genome Compression," *Curr. Bioinform.*, vol. 9, no. 3, pp. 315–326, 2014.
[43] R. Beal *et al.*, "A new algorithm for 'the LCS problem' with application in compressing genome resequencing data," *BMC Genomics*, vol. 17, no. S4, p. 544, 2016.
[44] H. H. Otu and K. Sayood, "A new sequence distance measure for phylogenetic tree construction," *Bioinformatics*, vol. 19, no. 16, pp. 2122–2130, 2003.
[45] M. Crochemore and L. Ilie, "Computing Longest Previous Factor in linear time and applications," *Inf. Process. Lett.*, vol. 106, no. 2, pp. 75–80, 2008.
[46] M. H. Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney, "Efficient storage of high throughput DNA sequencing data using reference-based compression," *Genome Res.*, vol. 21, no. 5, pp. 734–740, 2011.

Appendices

A Impact of Parameters ($\gamma, \beta, k_\rho, k_\alpha$) on MLCX compression

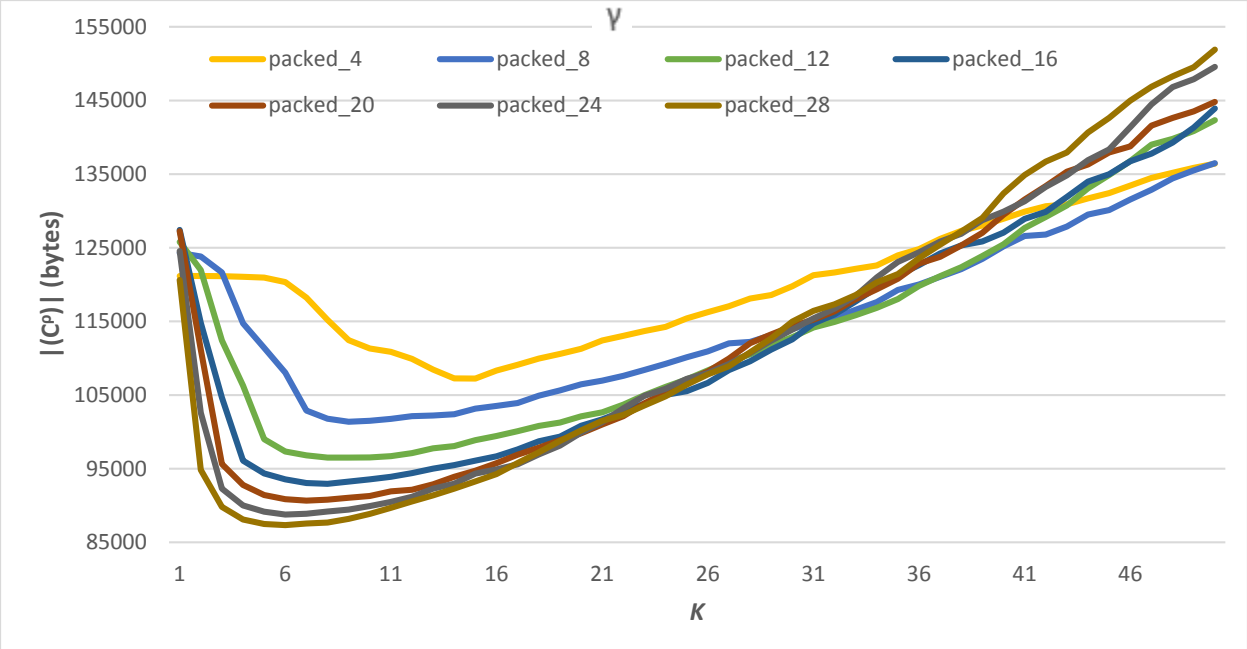

Figure 1 *Homo sapiens* chromosome 22: MLCX Encoding payload, ρ.

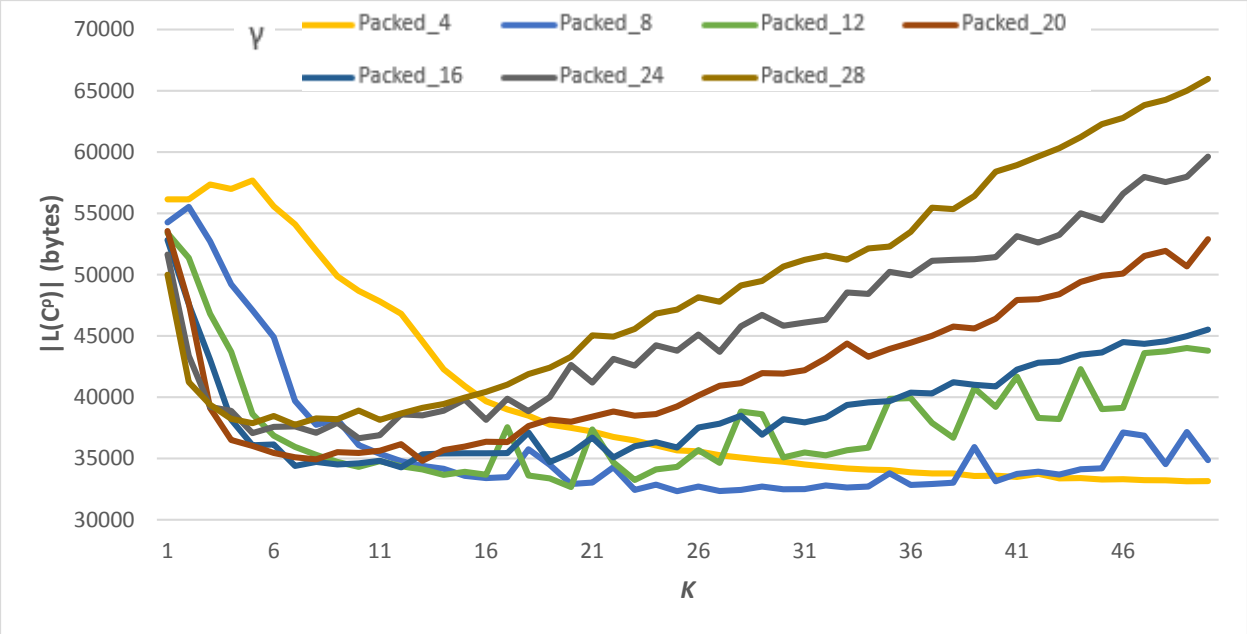

Figure 2 *Homo sapiens* chromosome 22: Using MLCX then LZMA2 to compress the encoded payload, (C^ρ).

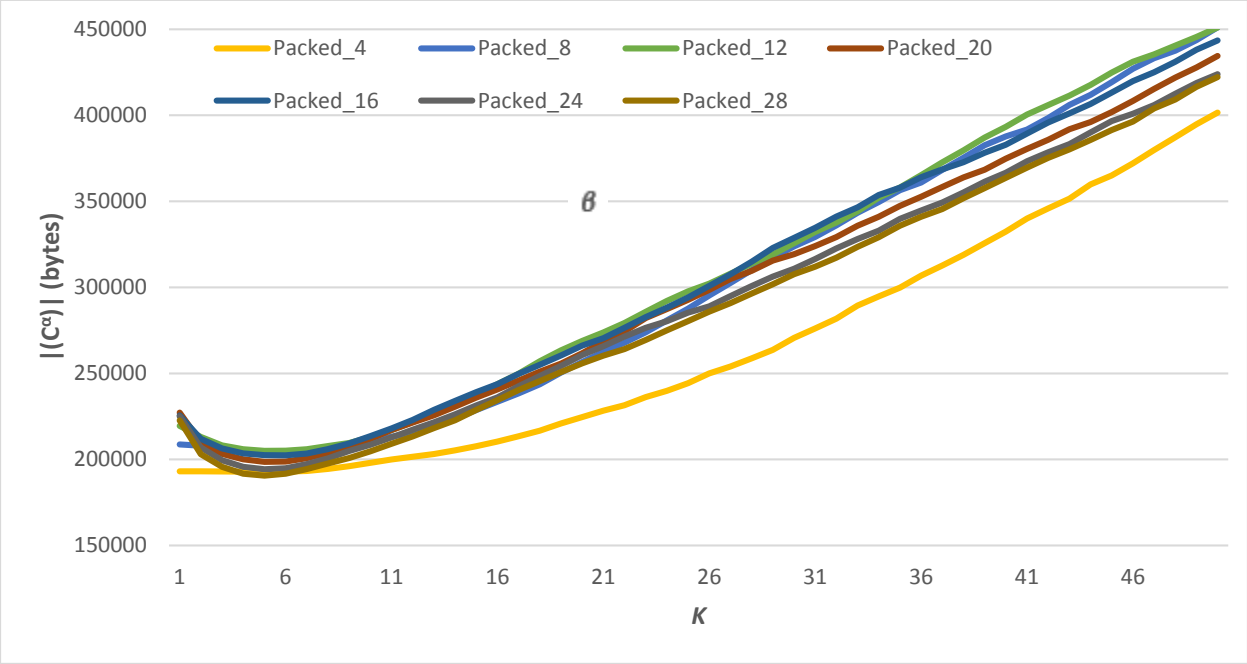

Figure 3 *Homo sapiens* chromosome 22: MLCX Encoding character-case bitstring, *α.*

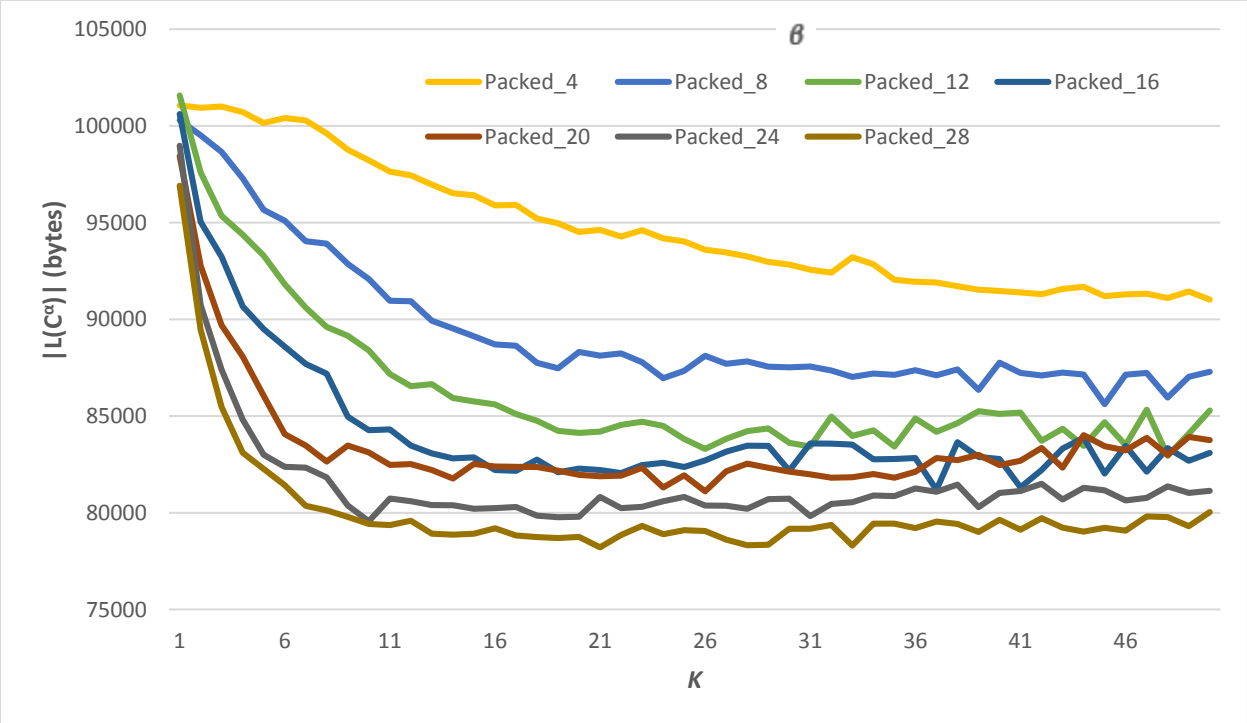

Figure 4 *Homo sapiens* chromosome 22: Using MLCX then LZMA2 to compress the encoded character-case bitstring, (C^{α}).

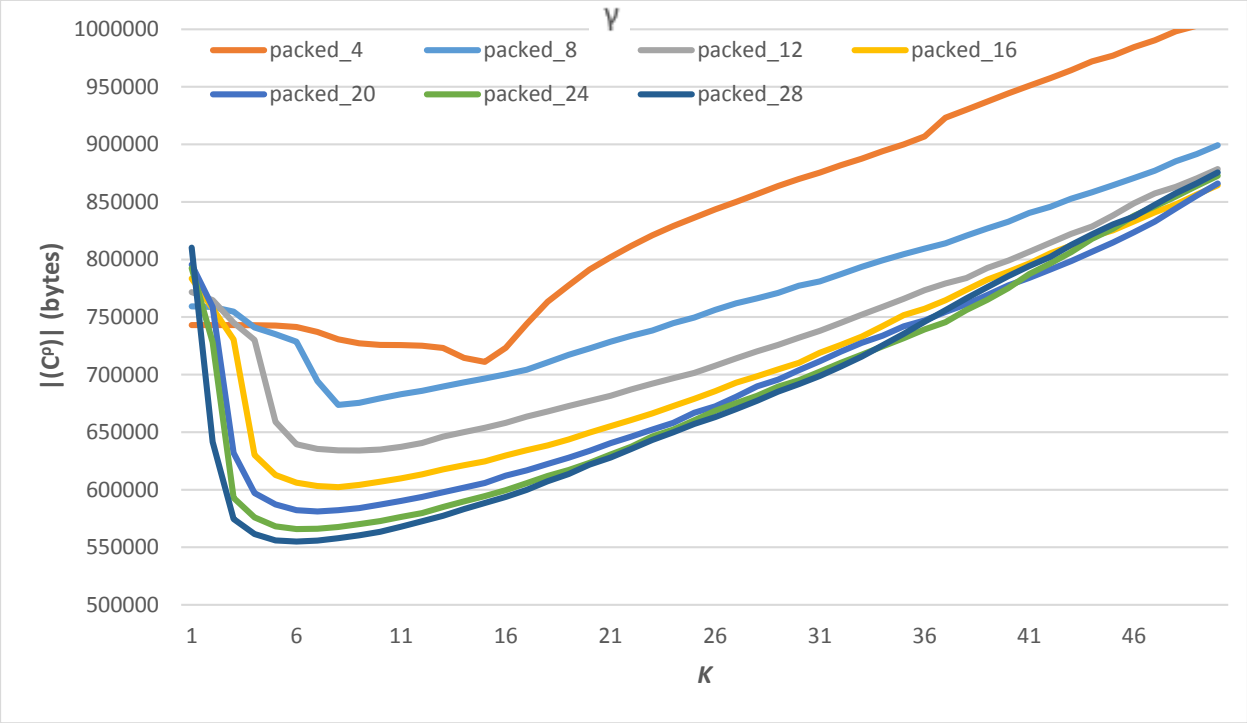

Figure 5 *Homo sapiens* chromosome X: MLCX Encoding payload, ρ.

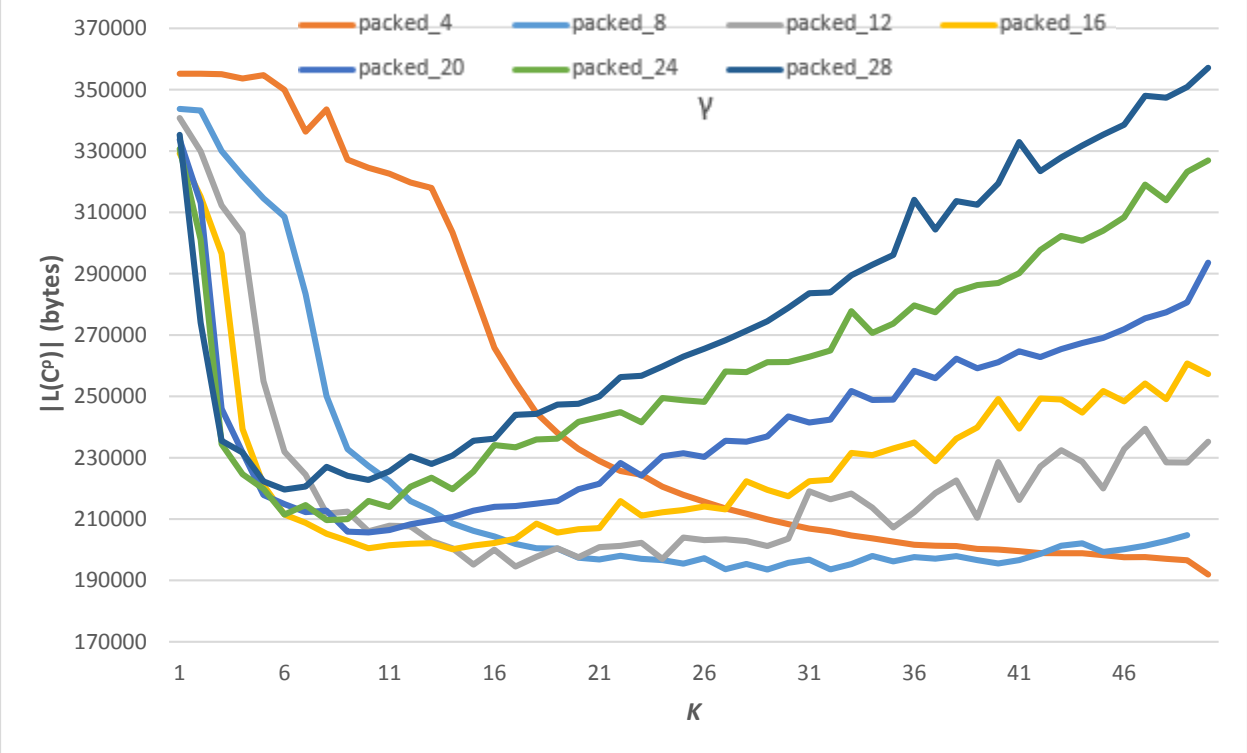

Figure 6 *Homo sapiens* chromosome X: Using MLCX then LZMA2 to compress the encoded payload, (C^{ρ}).

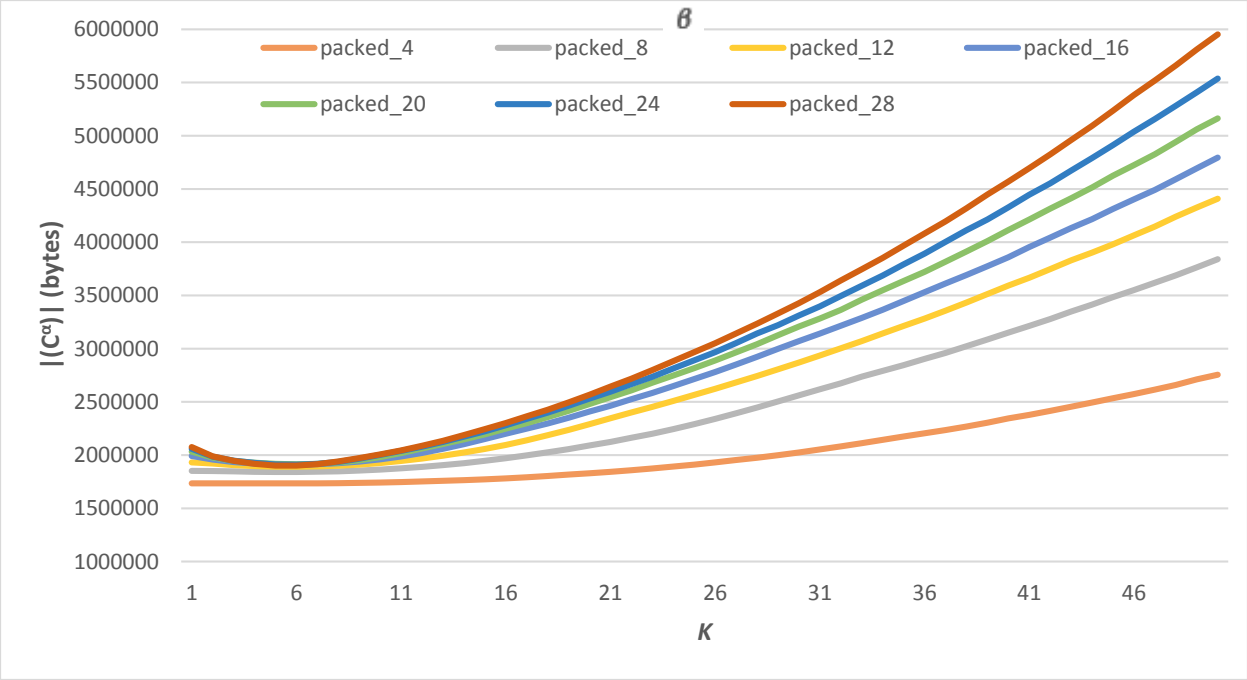

Figure 7 *Homo sapiens* chromosome X: MLCX Encoding character-case bitstring, *α*.

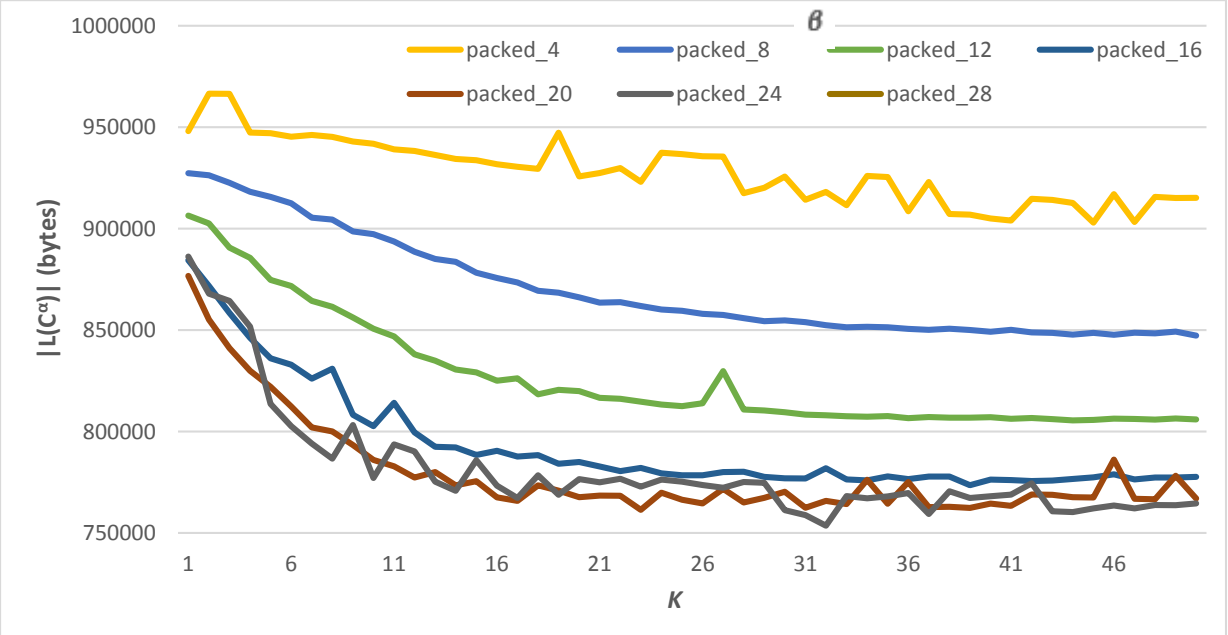

Figure 8 *Homo sapiens* chromosome X: Using MLCX then LZMA2 to compress the encoded character-case bitstring, (C^{α}).

B Impact of Parameters (γ, β, k_ρ,k_α) on SMLCX compression

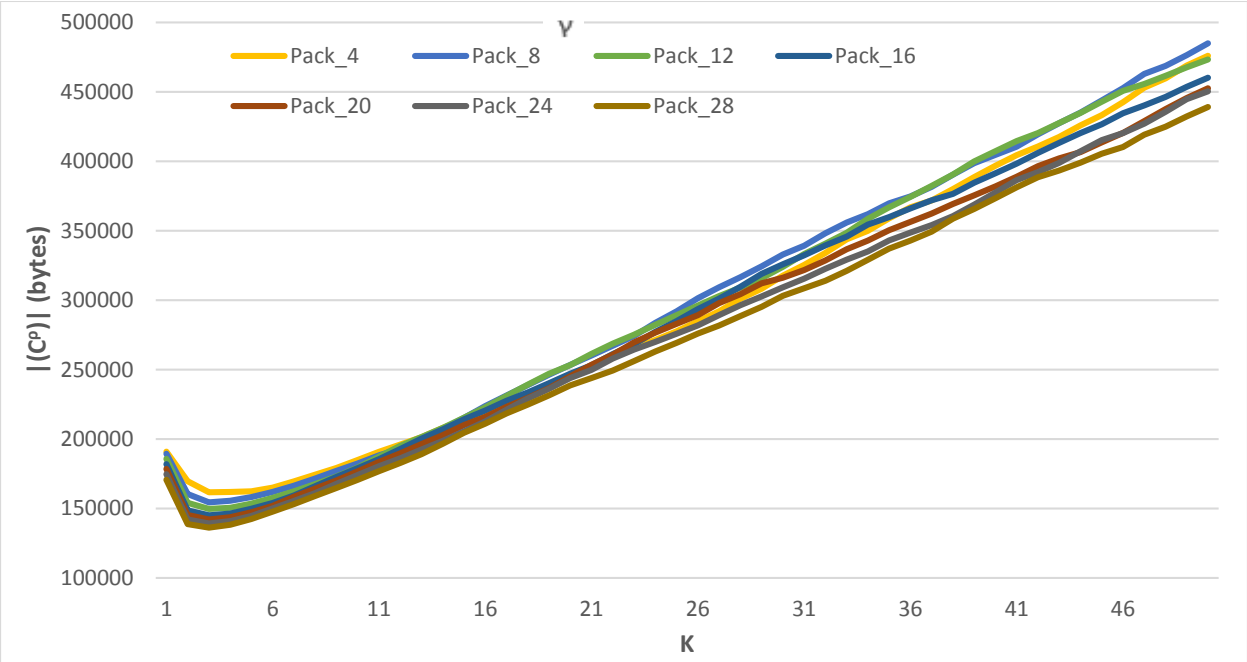

Figure 9 *Homo sapiens* chromosome 22: SMLCX Encoding payload, ρ.

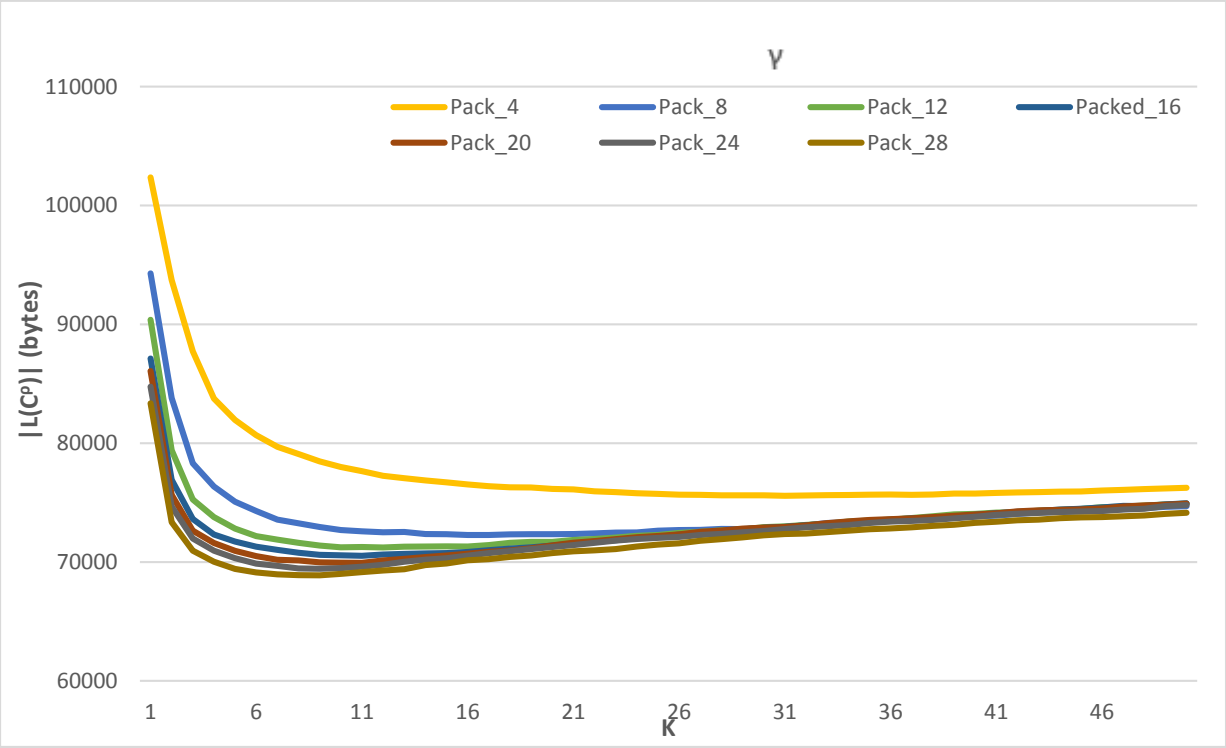

Figure 10 *Homo sapiens* chromosome 22: Using SMLCX then LZMA2 to compress the encoded payload, (C^ρ).

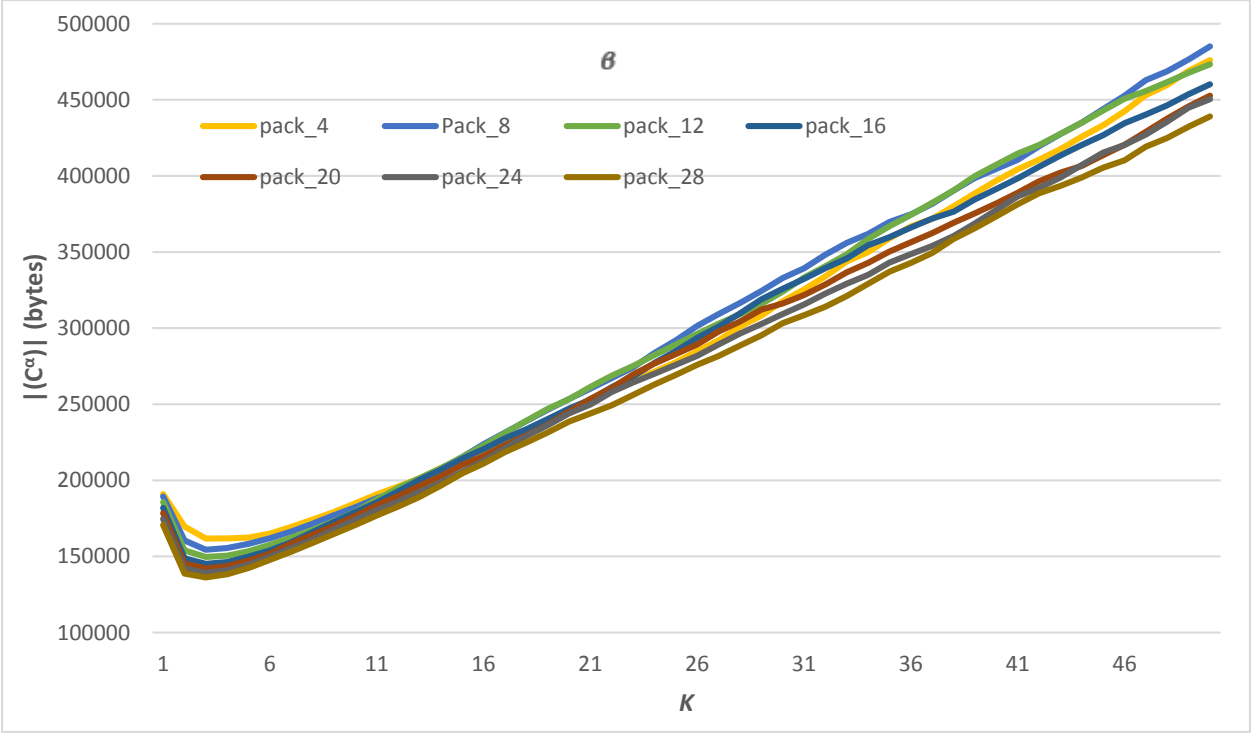

Figure 11 *Homo sapiens* chromosome 22: SMLCX Encoding character-case bitstring, *α.*

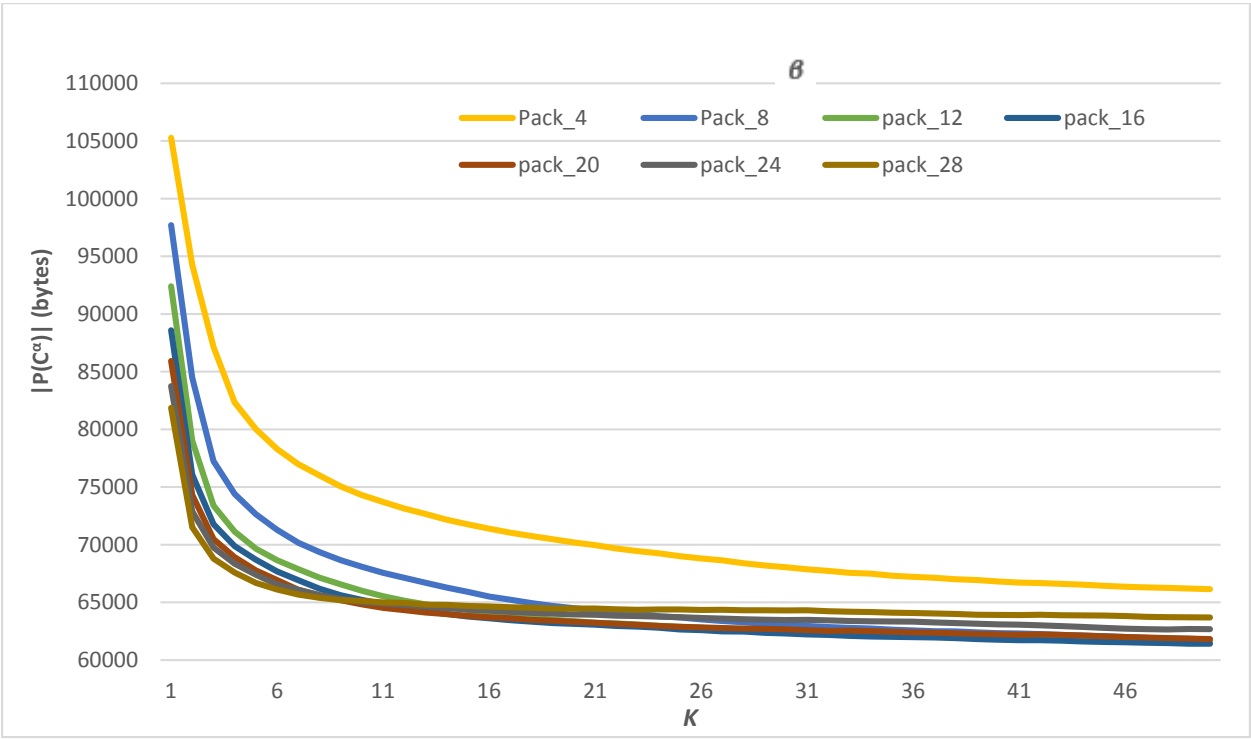

Figure 12 *Homo sapiens* chromosome 22: Using SMLCX then PPMD to compress the encoded character-case bitstring, (C^{α}).

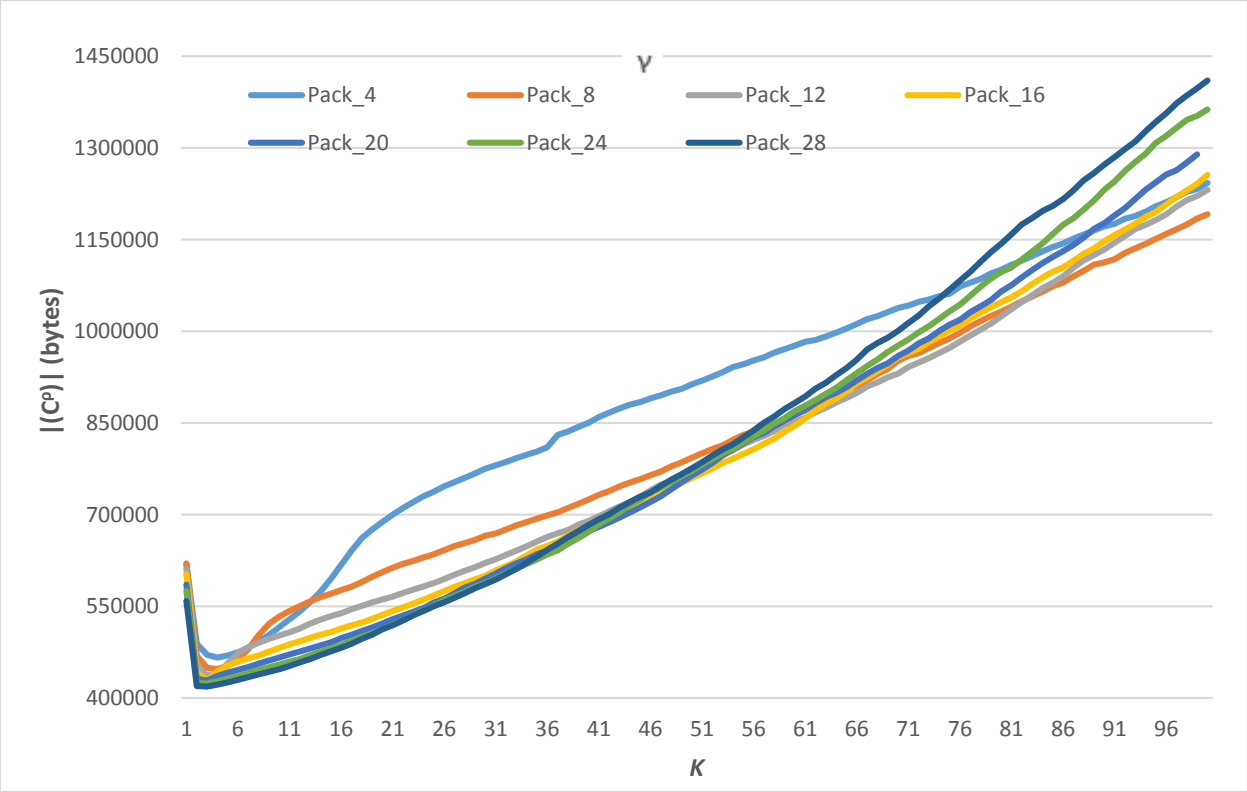

Figure 13 *Homo sapiens* chromosome X: SMLCX Encoding payload, ρ.

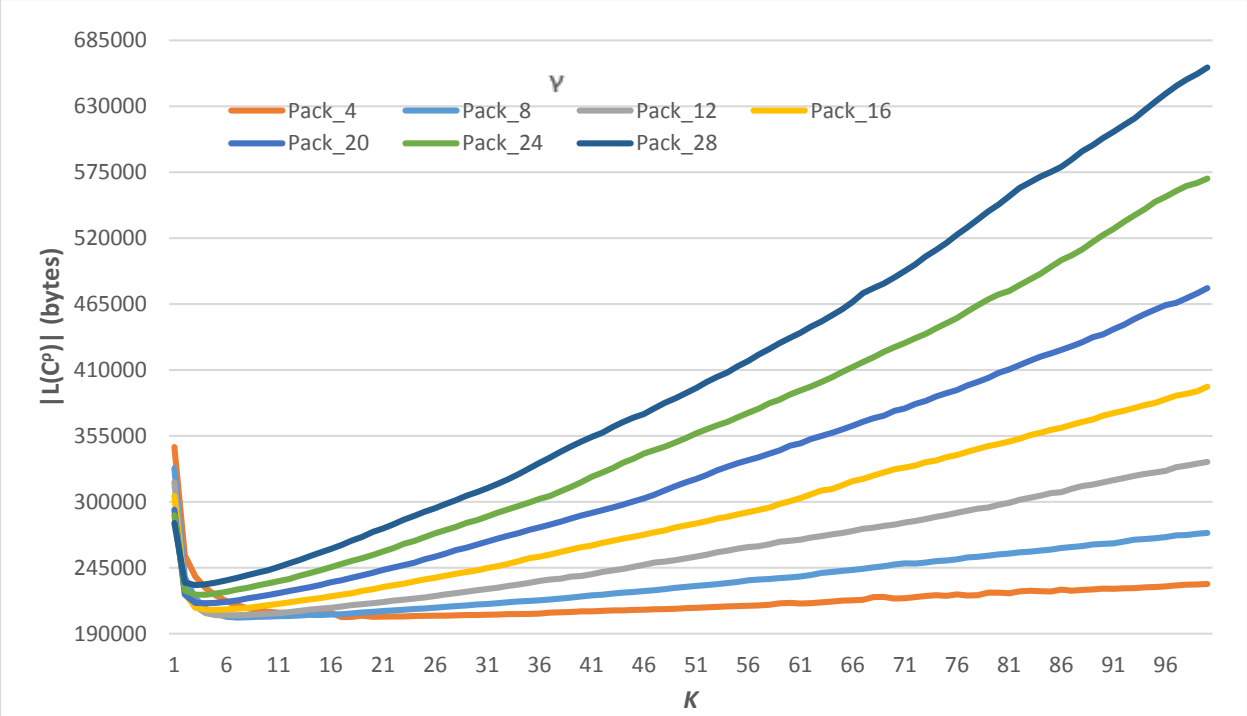

Figure 14 *Homo sapiens* chromosome X: Using SMLCX then PPMD to compress the encoded payload, (C^{ρ}).

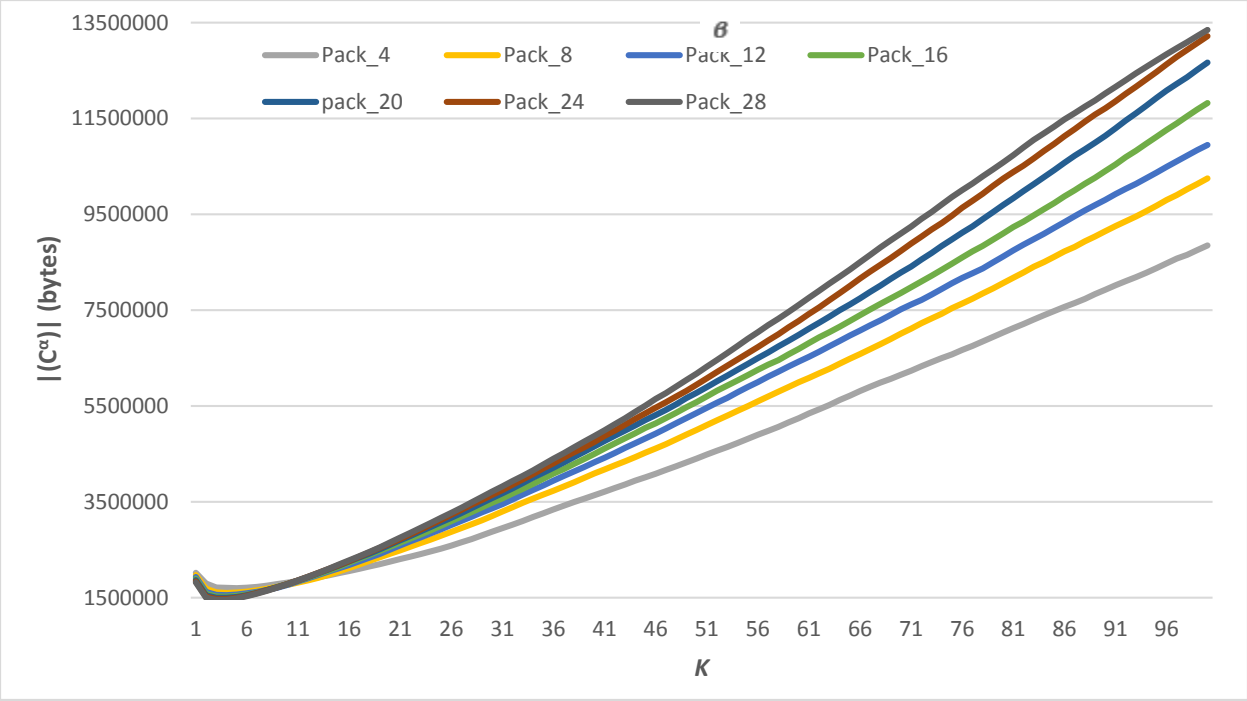

Figure 15 *Homo sapiens* chromosome X: SMLCX Encoding character-case bitstring, *α.*

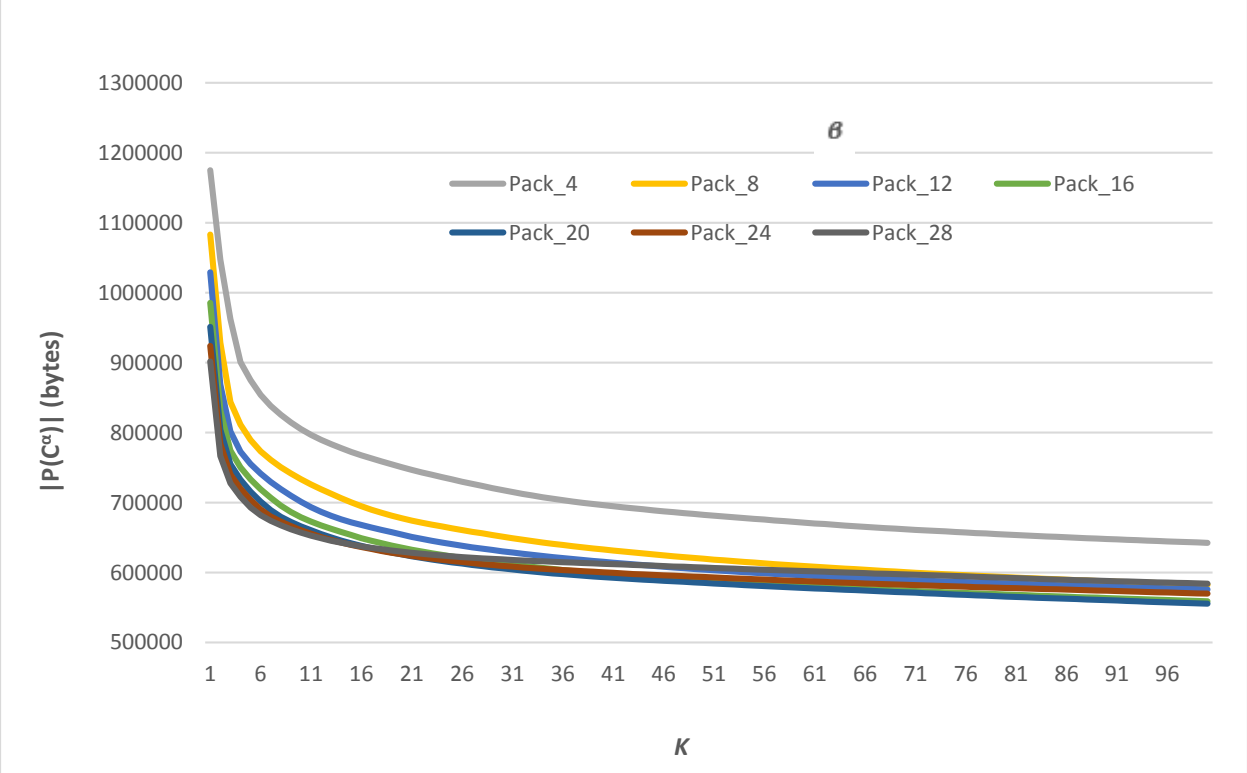

Figure 16 *Homo sapiens* chromosome X: Using SMLCX then PPMD to compress the encoded character-case bitstring, (C^{α})