Addis Ababa University
Addis Ababa Institute of Technology
School of Electrical and Computer Engineering

COMPRESSION OF AMHARIC TEXT USING PREDICTION BY PARTIAL MATCH (PPM) CONTEXT-MODELING ALGORITHM

A thesis submitted to Addis Ababa Institute of Technology, School of Graduate Studies, Addis Ababa University

in partial fulfillment of the requirement for the Degree of Master of Science in Electrical Engineering (*Electronic Communication Engineering*)

By

Yalemsew Abate Tefera

Advisor: Dr. -Ing. Dereje Hailemariam

Addis Ababa, Ethiopia
May 2019

Addis Ababa University
Addis Ababa Institute of Technology
School of Electrical and Computer Engineering

COMPRESSION OF AMHARIC TEXT USING PREDICTION BY PARTIAL MATCH (PPM) CONTEXT-MODELING ALGORITHM

By Yalemsew Abate Tefera

APPROVED BY BOARD OF EXAMINERS

Chairman, Graduate Studies Committee	Signature
Dr. -Ing. Dereje Hailemariam	
Advisor	Signature
Internal Examiner	Signature
External Examiner	Signature

Declaration

I, the undersigned, declare that this thesis work is my original work, has not been presented for a degree in this or any other universities, and all sources of materials used for the thesis work have been fully acknowledged.

Yalemsew Abate Tefera _____
Name Signature

Place: Addis Ababa Institute of Technology, AAiT
Addis Ababa University, AAU
Addis Ababa,
Ethiopia

Date of Submission: _____

This thesis has been submitted for examination with my approval as a university advisor.

Dr. -Ing. Dereje Hailemariam _____
Advisor's Name Signature

Abstract

A recent study on entropy estimation of Amharic language showed that its 16-bit representation in Universal Transformation Format (UTF-8) very high as compared to the entropy of the language. The study showed a minimum of 1.074 $bits/symbol$ and a maximum of 7.981 $bits/symbol$ can be sufficient for transmission of text sources written in Amharic through telecom networks. In digital communication, the source encoding operation produces a compressed representation of an information source for efficient utilization of communication resources like bandwidth and energy. Practical source encoding approaches in text compression use Statistical Language Models (SLMs) based on Markov process to model redundancies exhibited in a language.

The Prediction by Partial Match (PPM) context-modeling algorithm is capable of high compression rates and is well suited for multiple alphabet sources like textual data. PPM adaptively combines different order Markov models to capture dependencies between successive symbols in a text. In this thesis, the PPM algorithm is used to show the advantages gained by context-modeling techniques in Amharic text source encoding and demonstrate how close practical compression gets to estimated entropy of Amharic language.

Two Versions of the PPM algorithm; namely PPMC and PPMD were used to model and encode eight source files written in Amharic. It is shown that the optimum order for efficient encoding is order-3 and it is possible to achieve an average of 84.2% reduction in file size. Using both algorithms, an average compression rate of 3.3 $bits/symbol$ is attainable for source encoding and storage applications. Modeling Amharic text sources using context models in general and PPM in particular can help to maximize efficiency in communication networks by reducing the average number of bits required for coding text sources.

Key Words: Amharic; Entropy; Source Encoding; Context; Modeling; Coding; PPM

Acknowledgments

First and for most I would like to thank **God** for giving me the strength and tenacity to complete this thesis work.

I owe a great debt of gratitude to my advisor **Dr. -Ing. Dereje Hailemariam**, for his relentless support, encouragement and generosity throughout this work. This thesis would not be possible without your guidance, constant follow-ups, comments and suggestions. Thank you very much!

I am grateful to **Mr. Tesgamlak Terefe** for helping me become familiar with his work in the early stage of my study and later for providing me the corpora of Amharic books that I used in my experiments.

I also like to thank the **School of Electrical and Computer Engineering** at **Addis Ababa University,** for giving me a chance to complete my Master's study.

Last but not least, my heartfelt thanks goes to my **parents** for their endless patience and understanding throughout this process.

Table of Contents

List of Tables

List of Figures

List of Acronyms

BACC …….	Bangor Arabic Annotated Corpus
CR ……...	Compression Ratio
cdf ……...	Cumulative Distribution Function
DMS ……...	Discrete Memory Less Source
DSP ……	Digital signal Processer
i.i.d. ……	Independent and Identically Distributed
LCMC ……	Lancaster Corpus of Mandarin Chinese
LSB …….	Least Significant Bit
LOB ……	Lancaster-Oslo-Bergen Corpus
MSB ……	Most Significant Bit
NLM ……..	Neural Language Model
pdf ……..	Probability Density Function
PPM ……...	Prediction by Partial Match
SLM ……	Statistical Language Model
UTF ……	Universal Transformation Format
VLC ……	Variable Length Codes

1. Introduction

Before the advent of digital communication and information age, people looked for different ways to optimize communication with written text messages. A codebook by the British Admiralty (1790), the Brail (1820) and the Morse code (1838) are some of the examples [2]. These methods of reducing text were developed to economize transmission bandwidth, reduce communication time and/or improve human-machine interface [2]. Compared to the early 19th century, now in this day and age the available bandwidth and communication speed improved significantly. But the objective of coding text sources remained the same because of the inherent human behavior, *the more you have the more you want*! We are generating lots of textual data while interacting using several communication platforms. Thus, efficient coding of text sources is going to save us resources from being wasted.

Source encoding is the fundamental first step in digital communications and its role is to represent the information source in a compressed (compact) form by removing unnecessary or redundant information. The source for written textual data are natural languages. Efficient coding of text sources thus relies on extracting the redundancies exhibited in a language structure. How much redundancy can be extracted? How is redundancy measured? Which information is redundant? These questions are explained using the *entropy* of the source (in this case the language) which measures the uncertainty per output symbol of the information source. The need for communication system arises because of the uncertainty of what is being communicated. But useful (meaningful) information has some order in it. By carefully analyzing this order, some part of the information becomes predictable with respect to the knowledge of the rest. This predictable part is thus redundant since it can be regenerated on the receiving end of the communication system by transmitting only parts necessary to disambiguate prediction.

Source encoding of text sources usually breaks down in to two steps. The first step called *modeling* involves representing redundancies in the language in a mathematical model producing a bunch of probabilities based on relation of successive symbols [1].

The next step, called *coding*, maps these probabilities to a binary code: a string of 1's and 0's [1]. To get close to compression ratio (coding rate) of entropy limits we need a model that represents dependencies between symbols and suitable for coding to integrate with a coding scheme which efficiently uses the code space. A combination of *context modeling* techniques and *Arithmetic coding* serve this purpose.

A context model (also known as Finite-Context model) is a probability model that uses the *context* (generally a few preceding characters) in which input symbols occur, in order to determine the number of bits required to code the current symbol [7]. The model estimates probabilities conditioned on previously seen context (group of symbols) and uses them to predict the next symbol. In an adaptive model, the probability estimates are simply frequency counts based on the text seen so-far.

Context model is generally combined with Arithmetic coding to form a data compression system. In a data compression system, a compression algorithm takes an input X and generates a compressed representation X_c that requires fewer bits than the input. The decompression algorithm then tries to generate an output Y from X_c. Based on the reconstruction method used the algorithms are dubbed *lossless* if Y is exactly the same as X or *lossy* where Y is not exact but an approximation of X. A predictive model for lossless compression assigns probability to every outcome. Likely outcomes are assigned short codes and unlikely outcomes are assigned long codes. This thesis uses one of the prominent context-modeling algorithms called *Prediction by Partial Match* (*PPM*) to show the advantages of context-modeling on Amharic text sources.

1.1. Statement of the Problem

The motivation to compress textual data is for it to take less resource (such as time, energy and bandwidth) to send it over a communication link or that it requires less storage space (memory) for archiving textual information. Natural language texts exhibit redundancies that can be exploited for compression. In addition, representation of some languages in the Universal Transformation Format (UTF) encoding system generally creates overhead which increases the required number of bits per symbol.

Amharic is the official language of the Government of Ethiopia. About one quarter of population is literate in written form of Amharic [14]. UTF-8 allocates 16 bits to represent the character set known as "Fidel" which has 384 characters including numerals and punctuation marks [13]. It was shown in [3] that this 16-bit representation is very high when it is compared to the entropy of the language, estimated to $7.981\ bis/symbol$ for $N = 1$ and $1.074\ bits/symbol$ for $N = 15$, using Shannon's N-gram model [3]. In addition, Huffman and Arithmetic coding were used resulting in average coding rate of 5.805 bits/symbol and producing a 72.45% reduction in file size for first order language model [3]. Observing the results in [3], the following questions can be formulated:

i. Can context-based modeling lead to efficient coding of Amharic text sources?
ii. Can compression results get close to the theoretical entropy estimation?

This thesis explores *context-modeling* techniques to provide answers to these questions. Context models, in general, have been shown to yield good compression performance for other languages [2, 8, 9, 12, 15, 16].

1.2. Objectives

1.2.1. General Objective

The main objective of this thesis work is study and analyze the Prediction by Partial Match (PPM) context-modeling techniques for efficient compression of Amharic text sources and to provide additional insight in Amharic language modeling.

1.2.2. Specific Objectives

Particularly, the thesis aims to:

- Assess the PPM algorithm in relation to natural language modeling to determine the best approaches to implement;
- Implement PPM based compression algorithms on Amharic text source using software;

- Analyze, compare and interpret results in relation with entropy estimate of Amharic language.

1.3. Literature Review

In their pioneer work on entropy estimation of Amharic language, Tsegamlak and Dereje used Shannon's N-gram model [3]. They estimated block entropy (G_N) and conditional entropy(F_N), up to $N = 60$, based on a corpora of six books. The minimum number of bits per symbol (zero crossing entropy) H_z is found out to be 1.074 $bits/symbol$ at $N = 15$ with a 1% error margin [3]. Further, they used entropy coders (Huffman and Arithmetic coding) resulting in average of 5.805 $bits/symbol$ and producing a 72.45% reduction in file size for first order language model [3]. They concluded that Amharic can be compressed far greater than the theoretical estimations (86.54% redundancy) due to the assignment of bits in UTF-8 [3].

Early in the 1950's Shannon set out to estimate the entropy of written English and showed the role of context using experiments. First, Shannon used statistical models to obtain an entropy of 3.1 $bits/letter$ [5, 6]. Then, he used predictions generated by people to estimate upper and lower bounds of 1.3 and 0.6 $bits/letter$, respectively [6]. Cover and King developed a gambling methodology to estimate symbol probabilities of 27 symbol English (26 letters and space) instead of using guesses [20]. They estimated the minimum (best-case) entropy by selecting the results of the most successful gambler. Their result indicate an entropy between 1.25 $bits/symbol$ and 1.35 $bits/symbol$.

The approaches used by Shannon and the like capture statistical dependencies efficiently though subject to sampling errors while choosing subjects. The longer the context the better it's predictive value. However, in modern computing, storing the probability model with respect to all contexts of a given length is impractical [1]. The number of contexts grow exponentially with the length of the context. Furthermore, given that the source imposes some structure on its output, many of this contexts may correspond to strings that would never occur in practice. Therefore, context modeling in text compression schemes tend to be an adaptive strategy in which the probabilities for different symbols in different contexts are updated as they are encountered. To this

end, John G. Cleary and Ian H. Witten developed an adaptive context-based compression mechanism called Prediction by Partial Match (PPM) which is capable of achieving high compression rates, 2.2 $bits/character$, for different English texts [4, 7, 8]. Alistair Moffat tweaked Cleary and Witten's algorithm with some modifications and achieved a 15% improvement [8]. PPM has been used as a modeling technique in modeling and compression studies of Arabic, Armenian, Chinese, Czech, Persian and Russian languages [12, 15, 16]. Compression results for these languages in $bits/character$ are summarized in Table-1.1.

Table-1.1 : PPM compression results for different languages [15]

Language	**Corpora**	**Order**						
		1	**2**	**3**	**4**	**5**	**6**	**7**
Arabic	BACC	2.42	2.17	2.04	1.79	1.66	1.51	1.44
Armenian	HC	2.43	2.02	1.90	1.69	1.61	1.42	1.36
Chinese	LCMC	4.03	3.01	2.66	2. 49	2.46	2.47	2.49
English	Brown	3.67	3.02	2.51	2.23	2.16	2.17	2.22
	LOB	3.66	2.99	2.48	2.21	2.14	2.15	2.19
Persian	Hamshahri	2.29	2.09	1.96	1.75	1.6	1.42	1.33
Russian	HC	2.47	2.08	1.97	1.73	1.63	1.41	1.33

This thesis work helps to have a complete picture in Amharic language modeling and coding by producing similar results to the ones shown in Table-1.1 for Amharic. Adaptive context-modeling approach is used to present an alternative and find improvement on previous attempts of Amharic text source encoding.

1.4. Methodology

The following three tasks are important to achieve the stated objectives.

- Literature review of publications and books related to context-based models in text compression;
- Modeling of Amharic text sources using context-modeling techniques;
- Implement encoding algorithms based up on models;
- Analysis of results.

A literature review of related works is presented in Section 1.3. PPM is the best context-modeling algorithm when it comes to multiple alphabet sources like natural languages. Particularly PPMC and PPMD versions of the algorithm are tuned to provide the best performance and are chosen for implementation in the software development phase of the thesis.

Context-modeling approaches try to capture dependencies between successive symbols by combining different order Markov models in their symbol probability estimations. Models are built from scratch and are updated as more and more symbols are observed throughout the text source.

Adaptive nature of context-modeling approaches integrates the modeling and coding processes producing a one-pass approach. Symbol encoding using Arithmetic coding is done alongside probability estimations and updating of models.

Modeling and encoding steps are implemented using software. Originally the attractive features of *Python* like library supports for Unicode string processing, being dynamically typed and easy to understand helped in implementation phase of this work. A Python script was developed implementing PPMC and PPMD versions of the PPM context-modeling algorithm. Symbol probabilities generated in the modeling phase are then used to provide cumulative probabilities for the final coding stage which is implemented using Arithmetic coding on a separate script.

While context-modeling algorithms provide very good compression, they suffer from the disadvantages of being relatively slow and requiring large amounts of main memory in which to execute. To minimize the effects of this weakness requires using efficient data structures in representation of models through software programs. Thus, optimizations were necessary for improved performance since Python being dynamically typed is comparatively slow with respect to other programming languages. *Cython*, a bridge between C and Python, used to close the gap and achieve good compression performance with reasonable execution time.

Finally, interpretation of results obtained from the previous two tasks are combined to deliver the proposed main objective of the thesis.

1.5. Scope and Limitations

1.5.1. Scope of the Thesis

This thesis tries to address the issue of efficient encoding of Amharic text sources using context-modeling techniques. The scope is limited to analyzing multiple-alphabet context-modeling algorithms with finite order. The software implementation is aimed at achieving higher compression ratio/rate in order to show how close these results can get to estimated entropy of Amharic language. The study presents an alternative in Amharic language modeling for source coding and storage applications.

1.5.2. Limitations of the Thesis

Even if there are various efficient binary-alphabet context-modeling approaches presented in literatures related to source encoding, the algorithm used in this thesis is based on multiple-alphabet context-modeling techniques. But extended versions of binary-alphabet context-models exist that can be applied to text sources.

The software implementation using Python was not efficient in memory and processer utilization. Memory and processer usage optimizations to some extent were implemented using Cython which reduced execution time significantly but there is always room for an improvement perhaps using a different programming language.

1.6. Contributions

Considering the main objective of this thesis, the following can be considered as main contributions of the thesis:

- It presented the advantages of context-modeling for Amharic language in communication and storage applications;
- It showed a relative improvement (around 5%) in compression of Amharic text sources can be achieved using context-modeling techniques;
- The optimum order of context for good compression performance has been determined which can give directions for similar applications such as dictionary based systems.

1.7. Thesis Layout

The thesis is organized in five chapters. This chapter gives basic introduction, problem statement and brief literature review of related works. Chapter two focuses on the fundamentals of source encoding in relation with text sources, the different alternatives and their performance measures. The PPM context-modeling algorithm, its principles, smoothing techniques and variations are explained in Chapter three. Chapter four presents the experimental setup together with the observed results, discussion and analysis. Finally, conclusions and recommendations on future work are given in Chapter five.

2. Text Source Encoding

2.1. Source Encoding Fundamentals

2.1.1. Digital Communication System Model

A simple model of a digital communication system is shown in Figure-2.1. The information source can be either discrete in time with finite (countable) alphabet, such as written languages, or analog (continuous in time with uncountable alphabet) such as audio or video. In a digital communication system, messages produced by the source are usually converted into a sequence of binary digits by the source encoder. Ideally, we would like to represent the source output (message) by as few binary digits as possible. In other words, we seek an efficient representation of the source output that results in little or no redundancy. The process of efficiently converting the output of either an analog or a digital source into a sequence of binary digits is called source encoding or data compression [4].

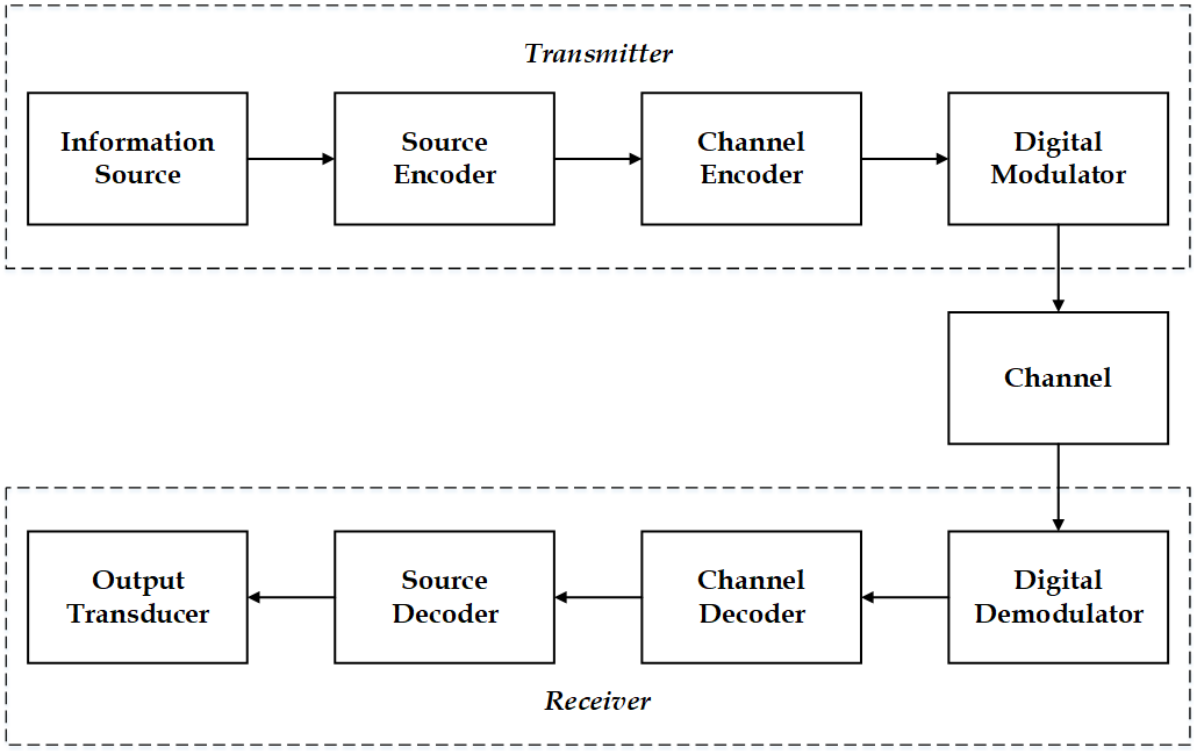

Figure-2.1 : Basic elements of a digital communication system [4].

The purpose of the channel encoder is to enable the reliable reproduction of the source encoder output after its transmission through a noisy communication channel. This is achieved by adding redundancy, in a controlled manner, to the source encoder output. The binary sequence at the output of the channel encoder is then passed to the digital modulator which serves as an interface to the communication channel. The modulator transforms the channel encoder output into a waveform suitable for transmission over the physical channel. This is usually accomplished by varying the parameters of a sinusoidal signal in proportion with the binary sequence provided by the channel encoder.

The communication channel is a noisy, unreliable physical medium over which the information will be transmitted from the source(transmitter) to the receiver. In wireless transmission, the channel is usually the atmosphere (space). On the other hand, telephone channels usually employ a variety of physical media, including wirelines, optical fiber cables, and wireless (microwave radio). Whatever the physical medium for signal transmission, the transmitted signal is corrupted in a random manner due to the channel imperfections. The main goal of the communication system is ensuring reliable communication by minimizing the distortions introduced by the channel.

The receiver part consists of the demodulator, the channel decoder and the source decoder were the reverse operations are performed. The digital demodulator processes the channel-corrupted transmitted waveform and reduces each waveform to a single value that represents the estimate of the transmitted data symbol.

The main focus of this thesis will be on the first two parts of the system model sown in Figure-2.1; namely, the information source and source encoder. Respectively the source decoder is implemented to emulate a closed system. A corpora of eight written Amharic literatures composed of both fiction and non-fiction are taken as an input to the system (the information source). As stated in the previous chapter source encoding generally consists of two parts: modeling of the data to be compressed (source to be coded) and coding with respect to the model. These concepts are elaborated in this Chapter in a greater detail.

2.1.2. The Information Source

Amharic (አማርኛ) is a Semitic language that is spoken mainly in Ethiopia. There are ninety languages that are spoken in Ethiopia according to the 1994 Ethiopian census conducted by Ethnologue [14]. Among this Amharic is the most popular and widely used. It is the official language of the Federal Government of Ethiopia and is used throughout the country for official government business. Amharic is spoken by more than 25 million people in Ethiopia.

Amharic has its own semi-syllabic writing system called Fidel (ፊደል). There are 26 consonant symbols that have seven variations. The variations according to a vowel coupled with a consonant result in 182 basic symbols. Inter-indigenous language translation and borrowed words are represented by 56 unique symbols. There are also 40 ligatures, 9 punctuation marks, 20 number symbols and 3 combining marks. The dominant version of Unicode, UTF-8 encoding allocates 16-bits code range (1200-137F) for Ethiopic Unicode block (Figure-2.2) containing characters for writing the Amharic, G-e'ez (ግዕዝ), Tigrinya (ትግርኛ) and other Ethiosemitic languages [13]. There are 384 symbols represented including additional 8 punctuation marks and 26 reserved code points.

2.1.3. Lossless Source Encoding

Based on the requirements of reconstruction, source encoding schemes (mechanisms) can be divided in to two classes: *lossless source encoding* and *lossy source encoding*. As the name implies in lossless source coding there is no loss of information when the message is reconstructed back from the binary sequences that are obtained from the source coder [1]. Coding operates by removing the redundant information contained in source and it is completely reversible so that the original data can be reconstructed exactly. Lossy source coding schemes, on the other hand, involve the loss of some information during compression and exact reproduction of the original message is not possible [1]. Such a method makes sense especially compressing image, audio or video sources. If the loss of data is handled carefully, it may not make a noticeable difference during reconstruction. In contrast, while compressing written text sources very small differences may result in statements with very different meanings.

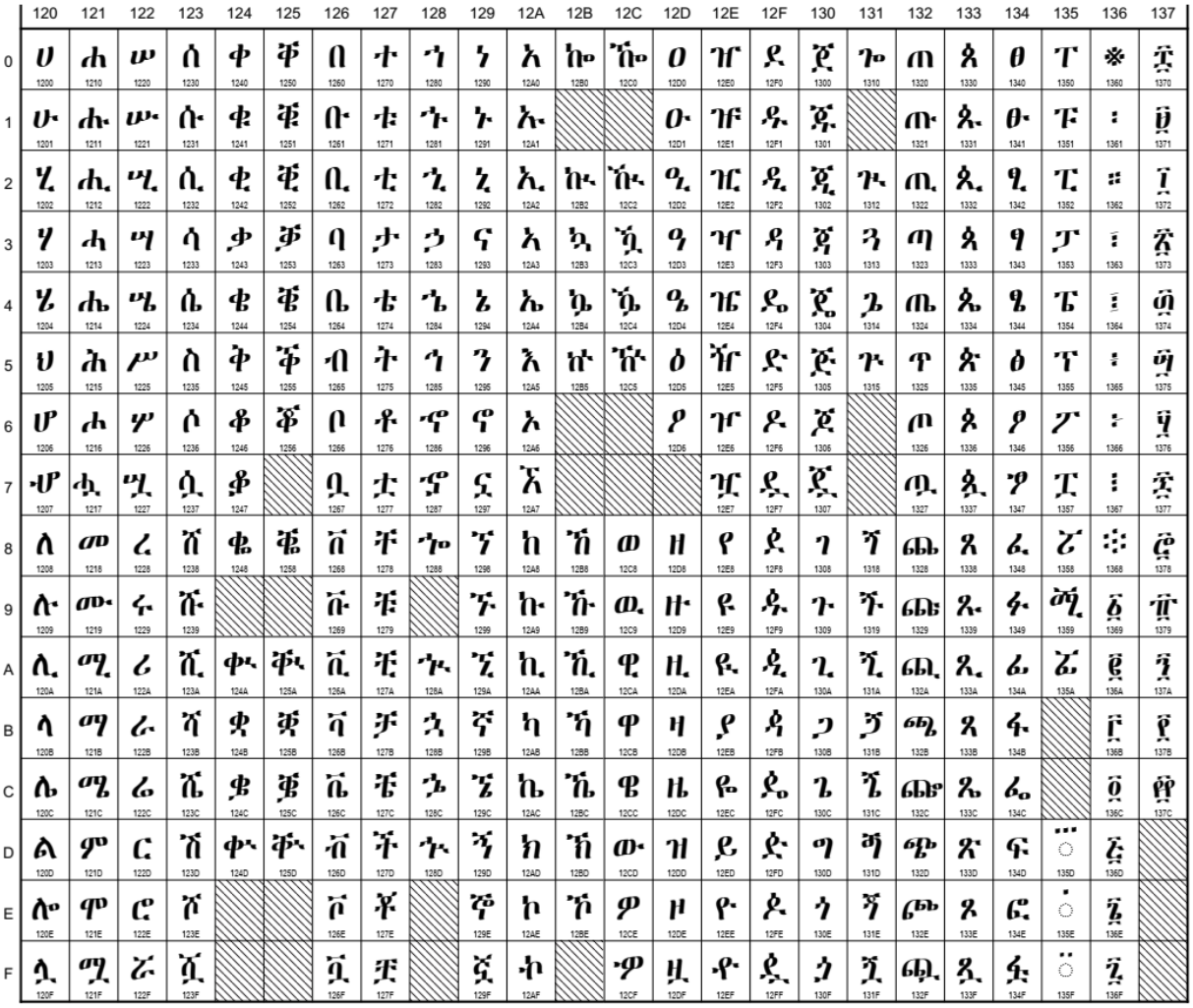

Figure-2.2 : Ethiopic Unicode Table [11]

For example, consider the sentences "*Do not send money*" and "*Do now send money*", a change in one symbol (character) changes the whole meaning of the sentence. For this reason, compression of text sources requires lossless source encoding even if lossy source encoding techniques can achieve better compression.

2.1.4. Entropy and Source Redundancy

In Section 2.1.1 it is indicated that source encoder aims to minimize the number of bits used to represent the information source. Lossless source coding, which applies to written text sources, achieves this by removing the redundant information in the source. But what portion or type of information is regraded redundant? Answering this question requires a quantitative measure of information content of the source.

2.1.4.1. Self-Information

Assume the output of a discrete information source is modeled as a discrete random variable A which takes symbols form a finite alphabet

$$\mathcal{A} = \{a_1, a_2, \dots, a_N\} \tag{2.1}$$

of size N, with probabilities

$$P(A = a_k) = p_k, \quad k = 1, 2, \dots, N \tag{2.2}$$

This model of an information source is called a *discrete memory less source* (DMS). A DMS generates a sequence of independent and identically distributed (i.i.d) random variables taking values form alphabet $\mathcal{A}$.

The self-information associated with the event $A = a_k$, which occurs with probability p_k is given by [4, 26]:

$$I(a_k) = \log_2 \frac{1}{P(A = a_k)} = \log_2 \frac{1}{p_k} = -\log_2 p_k \ [bits] \tag{2.3}$$

Self-information has the following properties [6]:

i. $I(a_k) = 0$ for $p_k = 1$, no information is gained from certain events.

ii. $I(a_k) \geq 0$ for $0 \leq p_k \leq 1$, the occurrence of an event $A = a_k$ provides some or no information, but never brings a loss of information.

iii. $I(a_k) > I(a_i)$ for $p_k < p_i$, less probable events convey more information.

iv. $I(a_k a_l) = I(a_k) + I(a_l)$, if events a_k and a_l are statistically independent.

2.1.4.2. Entropy – Average Self-Information

Following the definition of the self-information, the information content of a discrete information source is the mean of the information associated with each possible realization of the event $A = a_k$, $k = 1, 2, \dots, N$ and is given by [4, 26]:

$$H(A) = \mathrm{E}[I(a_k)] = \sum_{k=1}^{N} p_k I(a_k) = -\sum_{k=1}^{N} p_k \log_2 p_k \quad \left[\frac{bits}{symbol}\right] \tag{2.4}$$

$H(A)$ is called the *entropy* of the source and measures the average information content of a symbol.

2.1.4.3. Mutual Information

For two random variables $A_1 = a_i$ and $A_2 = a_j$, the joint information associated with both random variables is [26]:

$$I(a_i, a_j) = \log_2 \frac{1}{P(A_1 = a_i, A_2 = a_j)} = -\ \log_2 P(A_1 = a_i, A_2 = a_j) \qquad (2.5)$$

where $P(A_1 = a_i, A_2 = a_j)$ is the joint probability of the two events.

The quantity of information associated with the realization of event $A_1 = a_i$ conditioned on event $A_2 = a_j$ is [26]:

$$I(a_i|a_j) = \log_2 \frac{1}{P(A_1 = a_i|A_2 = a_j)} = -\ \log_2 P(A_1 = a_i|A_2 = a_j) \qquad (2.6)$$

Using the relation $P(A_1 = a_i, A_2 = a_j) = P(A_1 = a_i|A_2 = a_j)P(A_2 = a_j)$

$$\begin{aligned} I(a_i, a_j) &= -\ \log_2 P(A_1 = a_i, A_2 = a_j) \\ &= -\log_2[P(A_1 = a_i|A_2 = a_j)P(A_2 = a_j)] \\ &= I(a_i|a_j) + I(a_j) \end{aligned} \qquad (2.7)$$

The mutual information $I(a_i; a_j)$ between events A_1 and A_2 is the reduction in uncertainty of A_1 due to the knowledge of A_2(or vice versa) [26];

$$I(a_i; a_j) = I(a_i) - I(a_i|a_j) = I(a_i) + I(a_j) - I(a_i, a_j) \qquad (2.8)$$

Equation (2.8) is important when dealing with sources with memory or dependent events. If the two events are independent $P(A_1 = a_i|A_2 = a_j) = P(A_1 = a_i)$, $I(a_i|a_j) = I(a_i)$, consequently, the mutual information

$$I(a_i; a_j) = I(a_i) - I(a_i) = 0$$

On the other hand, if event A_1 is equivalent to event A_2, $P(A_1 = a_i|A_2 = a_j) = 1$,

$$I(a_i|a_j) = 0 \text{ and } I(a_i; a_j) = I(a_i).$$

2.1.4.4. Joint and Conditional Entropy

Two random variables A and B taking symbols from alphabets $\mathcal{A} = \{a_0, a_1, \dots, a_{N-1}\}$ and $\mathcal{B} = \{b_0, b_1, \dots, b_{M-1}\}$ respectively, the joint entropy $H(A,B)$ is defined as [26]:

$$\begin{aligned} H(A,B) &= \sum_{i=1}^{N}\sum_{j=1}^{M} P\big(A=a_i, B=b_j\big)\, I\big(a_i, b_j\big) \\ &= -\sum_{i=1}^{N}\sum_{j=1}^{M} P\big(A=a_i, B=b_j\big) \log_2 P\big(A=a_i, B=b_j\big) \end{aligned} \tag{2.9}$$

The conditional entropy of random variable A given random variable B is defined by [26]:

$$\begin{aligned} H(A|B) &= \sum_{i=1}^{N}\sum_{j=1}^{M} P\big(A=a_i, B=b_j\big)\, I\big(a_i|b_j\big) \\ &= -\sum_{i=1}^{N}\sum_{j=1}^{M} P\big(A=a_i, B=b_j\big) \log_2 P\big(A=a_i|B=b_j\big) \end{aligned} \tag{2.10}$$

Using Equation (2.7),

$$H(A,B) = H(A|B) + H(B) = H(A) + H(B|A) \tag{2.11}$$

2.1.4.5. Average Mutual Information

Extending the definition of the mutual information between two events, the average mutual information $I(A;B)$ between two random variables A and B is given by [26]:

$$\begin{aligned} I(A;B) &= \sum_{i=1}^{N}\sum_{j=1}^{M} P\big(A=a_i, B=b_j\big)\, I\big(a_i; b_j\big) \\ &= -\sum_{i=1}^{N}\sum_{j=1}^{M} P\big(A=a_i, B=b_j\big) \log_2 \frac{P\big(A=a_i|B=b_j\big)}{P(A=a_i)} \end{aligned} \tag{2.12}$$

$$I(A;B) = H(A) - H(A|B) = H(A) + H(B) - H(A,B) \tag{2.13}$$

2.1.4.6. Redundancy

For a DMS the source entropy is given by Equation (2.4). The entropy is maximum if the symbols are equiprobable. For a source with N symbols , the upper bound for the entropy is given by [3, 26]:

$$H_{MAX} = \log_2 N \tag{2.14}$$

In the same manner, for a source with memory (symbols are dependent on each other), the entropy can be computes as [26]:

$$H(A) = \lim_{m\to\infty} \frac{1}{m} H_m(A) \tag{2.15}$$

where $H_m(A)$ is the entropy per group of m symbols.

The redundancy of the source R characterizes the difference between the entropy of information of the source and the entropy of a source with equiprobable symbols. Thus [26],

$$R = 1 - \frac{H(A)}{H_{MAX}} \tag{2.16}$$

R ranges from 0 (the source symbols are independent and equiprobable) to 1 (the entropy of the source is zero).

2.2. Modeling of Text Sources

Modeling is the way to extract information about any redundancy that exists in the source and describe the redundancy in the form of a model [1]. Thus, the model is the representation of only the important information in the source. Modeling text sources devolves in some way to modeling of natural languages which are sources of written text. The DMS model cannot be an accurate model for text sources because it assumes symbols (words) are independent of each other, which is not the case in a language structure.

The redundancy in languages has two reasons: unequal probability of occurrence of symbols (characters) and dependence of symbols on each other also known as context. Source models for written languages are broadly classified as Morphological and Statistical [11]. Morphological methods relay on grammars which are defined based on linguistic knowledge [11]. Statistical language models (SLMs) use corpus to generate probability models based on frequency of occurrence of symbols and the correlation between symbols [3, 11]. Most statistical models are based on one of the following approaches [10]:

- **Frequency**: the model assigns probabilities to the symbol based on their frequencies of occurrences;
- **Context**: the model considers the context of a symbol when assigning its probability.

2.2.1. Context Models

A totality of texts written in a certain language can be considered as a set of realizations of a random process generated by a certain source [17]. Claude E. Shannon modeled written English texts as the output of a source that generates a sequence of symbols from a finite alphabet $\mathcal{A}$ according to certain probabilities [5]. This random process is assumed to be ergodic and in the special case when the probability of occurrence of the next symbol in the text depends on the previous symbols or its context it is called a Markov process [5, 17]. Though, it is by no means a finite-order Markvoian process, a description of written language as a Markovian process of high order can be used as useful approximation [17].

A finite-context model uses the context provided by characters already seen to determine the encoding of the current character. The idea of a context consisting of a few previous characters is based on Markov model. Markov models are used to represent statistical dependence between sequences of symbols on each other [1]. For models used in lossless source encoding of discrete sources, a specific type of Markov process called discrete time Markovian chain is used.

Let $\{x_n\} = \{x_1, x_2, \dots, x_n\}$ be a sequence of observations. The kth-order Markov model for this sequence is given by [1]:

$$P(x_n|x_{n-1}, \dots, x_{n-k}) = P(x_n|x_{n-1}, \dots, x_{n-k}, \dots) \tag{2.17}$$

In this model , knowledge of the past k symbols is equivalent of the entire past history of the process. Markov models are particularly useful in modeling of text source, where the probability of the next letter is heavily influenced by the preceding letters. Knowing the context of a symbol being encoded often makes it possible to obtain an accurate probability model for the symbol. A major difficulty with the idea of using contexts to

determine a probability model is that the number of contexts for a symbol grows exponentially with the length of the context [1, 2]. It would not be practical to encode using probabilities with respect to all possible high-order contexts. To see the extent of this problem consider an alphabet of size N, the number of first-order contexts is N, the number of second-order contexts is N^2, and so on. In order to encode a sequence from an alphabet of size say 256 using contexts of order-5, $256^5 = 1.09951 * 10^{12}$ probability distributions need to be calculated. Consequently, almost all practical context modeling approaches are adaptive in nature.

If a sequence of symbols being encoded does not consist of independent occurrences of the symbols, then the knowledge of which symbols have occurred in the neighborhood of the symbol being encoded will give a much better idea of the value of the symbol being encoded. Given the context, some symbols will occur with much higher probability than others which makes probability distribution more skewed. If the context is known to both encoder and decoder, this skewed distribution can be used to perform the encoding, thus increasing the level of compression [1, 2]. The decoder can use its knowledge of the context to determine the distribution to be used for decoding.

Adaptive context models predict successive symbols taking into account the context provided by symbols already seen. What the term predict refers is that the frequency values used in encoding the current symbol are determined by its context. The frequency distribution used to encode a symbol determines the number of bits it contributes to the binary representation. When symbol a_k occurs with context C and if the model for context C does not include a priori frequency for a_k , then context C fails to predict a_k. This is known as the "*zero frequency problem*" [1, 2]. Depending on the modeling type a lower order model or a default fallback model is used to code the symbol.

There are two ways in which a context model uses previously occurred characters. It may use a fixed number of previous characters in its predictions or combine predictions from contexts of several lengths.

- **Fixed order model:** A model that uses k previous characters to predict the current character is called a pure order-k context model. When $k = 0$, no context

is used and the text is simply coded one character at a time. When $k = 1$, the previous charter is used in encoding the current character, $k = 2$, the previous two characters, and so on.

- **Blended model:** Blending uses a number of models with different values of k in consort. In a blended model prediction of several contexts of different length are combined in to a single overall probability. For example, if a model based on the previous three characters, when the three-character context fails to predict, the model tries to predict using a reduced order-2 context model.

A blended model is composed of two or more sub-models. It is called *fully-blended* if it contains sub-models for the maximum-length context and all lower-order contexts. As such a fully-blended order-3 context model bases its predictions on models of orders $3, 2, 1, 0$ and -1. The order (-1) model is the default fallback model where each member of the alphabet is equiprobable. A *partially-blended* model uses some, but not all, of the lower-order contexts.

2.2.2. Static, Semi-adaptive and Adaptive Models

Modeling can be either static or dynamic. A model is static if the information of which it consists remains unchanged throughout the encoding process. The drawback of this scheme is it will give unboundedly poor compression whenever the text being coded does not correspond to the model [2]. Semi-adaptive modeling addresses this problem by using a different model for each text encoded. Before performing the compression, the text (or a sample of it) is scanned, and a model is constructed from it. The model must be transmitted to the decoder beforehand for decoding. Despite the extra cost of transmitting the model, the strategy will generally pay off because the model is well suited to the text [2].

A dynamic or adaptive model modifies its representation of the input as encoding proceeds and it gathers information about the characteristics of the source. Both encoder and decoder track the changes in the model and adapt to it. Adaptive (or dynamic) modeling avoids the overhead of transmitting the model as follows; Initially, both the encoder and decoder assume some default model, such as all characters being

equiprobable. The encoder uses this model to transmit one symbol, and the decoder uses it to decode the symbol. Then both the encoder and decoder update their models in the same way (e.g., by increasing the probability of the observed symbol). The next symbol is transmitted and decoded using the new model and serves to update the models again. Coding continues in this manner, with the decoder maintaining an identical model to the encoder since it uses exactly the same algorithm to update the model, providing no transmission errors occur. The model being used will soon be well suited to the text being compressed, yet there was no need for it to be transmitted explicitly.

2.3. Coding of Text Sources

Most of the time the term "Coding" is often used in a broader sense to refer to the whole source encoding (compression) process (including modeling). There is a difference between coding in the wide sense (the whole process) and coding in the narrow sense which is the production of a binary sequence (bit-stream) given a model. In this section the term "coding" always refers to the later. The set of binary sequences is called a code, and the individual elements of the set are called codewords.

2.3.1. Source Coding Theorem

The relationship between probabilities and codes was established in Shannon's source coding theorem.

Theorem 1: Shannon's Source Coding Theorem

A source with entropy (or entropy rate) H can be encoded with arbitrarily small error probability at any rate R $(bits/source\ output)$ as long as $R > H$. Conversely if $R < H$, the error probability will be bounded away from zero, independent of the complexity of the encoder and decoder employed [4, 5].

According to the theorem, entropy(H) of the source gives a lower bound in the rate at which the source can be compressed for reliable reconstruction. It is possible to design a code at rates above the entropy, whereas at rates below entropy such a code doesn't exist. Shannon's theorem shows that a symbol that is expected to occur with probability

p is best represented in $-\log_2 p$ bits. In this manner a symbol with a high probability of occurrence is coded with fewer number of bits as compared to unlikely (low probability) symbols. Such assignment of codes produces variable-length codes.

2.3.2. Variable Length Codes (VLCs)

A variable-length code for lossless source encoding is a code in which the source symbols can be completely reconstructed without distortion. In order to achieve this goal, the source symbols have to be encoded unambiguously in the sense that any two different source symbols (with positive probabilities) are represented by different codewords. Codes satisfying this property are called non-singular codes.

In practice, the encoder often needs to encode a sequence of source symbols, which results in a concatenated sequence of codewords. If any concatenation of codewords can also be unambiguously reconstructed, then the code is said to be uniquely decodable. Theorem 2 states the necessary condition for unique decodabiltiy.

Theorem 2: The Kraft-McMillan Inequality

A uniquely decodable code C with binary code alphabet {0,1} and with M codewords having lengths $l_0, l_1, l_2, \dots, l_{M-1}$ must satisfy the following inequality [1],

$$\sum_{m=0}^{M-1} 2^{-l_m} \leq 1 \tag{2.18}$$

In addition to unique decodability, VLCs should allow instantaneous decoding as soon as the binary sequence representing the source is fully received. A particular set of uniquely decodable codes called prefix codes allow to decode codes instantaneously without reference to future codewords in the same sequence. A code is said to be a prefix code if no codeword is a prefix to another codeword. The most common prefix VLCs that are used in coding text sources are Huffman and Arithmetic codes which are presented in the following sections.

2.3.3. Huffman Codes

The basic idea of behind Huffman coding is to assign each symbol of the alphabet $\mathcal{A}$ a sequence of bits roughly equal to the amount of information conveyed by the symbol

in question. If we can map each source output of probability p_i to a code word of length approximately $\log_2 \frac{1}{p_i}$ and at the same time ensure unique decodability, we can achieve an average code word length of approximately $H(A)$. The end result is a source code whose average codeword length approaches the fundamental limit set by the entropy of the source. Huffman codes are uniquely decodable instantaneous codes with minimum-average code word length which makes them optimum for a given set of probabilities in a model [4].

The observations on optimum prefix codes that are the base for the Huffman procedure are; Given a source with alphabet $\mathcal{A} = \{a_1, a_1, \dots, a_N\}$ and probability $\{p_1, p_2, \dots, p_N\}$. Let l_i be binary codeword length of symbol a_i. Then there exists an optimal-uniquely decodable variable-length code satisfying

i. $p_i > p_j$ implies $l_i \leq l_j$;

ii. The symbols that occur least frequently will have the same length;

iii. The two longest codewords differ only in the last bit correspond to the least-frequent symbols.

Huffman Encoding Algorithm:

1. List all possible symbols with their probabilities
2. Locate the two symbols with the smallest probabilities
3. Replace these by a single set containing both symbols whose probability is the sum of the individual probabilities
4. Repeat until the list contains only one member
5. Go backward by splitting one of the two (combined) symbols into two original symbols, and the codewords of the two split symbols is formed by appending 0 for one of them and 1 for the other from the codeword of their combined symbol. Repeat this step until all the original symbols have been recovered, and have obtained a codeword.

Length of Huffman Codes

For a source with alphabet $\mathcal{A} = \{a_1, a_1, \dots, a_N\}$ and probability $\{p_1, p_2, \dots, p_N\}$, the average codeword length $\bar{L}$ is given by [1]:

$$\bar{L} = \sum_{i=1}^{N} p_i \, l_i \tag{2.19}$$

where l_i is binary codeword length of symbol a_i.

For Huffman code the average codeword length $\bar{L}_H$ corresponding to the source output A, satisfies [1]:

$$H(A) \leq \bar{L}_H < H(A) + 1 \tag{2.20}$$

If the code is designed for sequence of source symbols of length m(m^{th} extension of the source given by $\mathrm{A^m}$), the upper and lower limits of average codeword length become [1]:

$$H(\mathrm{A^m}) \leq \bar{L}_{H_m} < H(\mathrm{A^m}) + 1 \tag{2.21}$$

where $\bar{L}_{H_m}$ denotes the average code word length for the extended alphabet(number of bits per m group of symbols). The number of bits required per symbol $\bar{L}_H$ is given by [1]:

$$\bar{L}_H = \frac{1}{m} \bar{L}_{H_m} \tag{2.22}$$

Thus,

$$H(A) \leq \bar{L}_H < H(A) + \frac{1}{m} \tag{2.23}$$

Comparing Equations (2.23) and (2.20), encoding the output of the source in longer blocks of symbols guarantees a rate closer to the entropy. For discrete sources with memory, $\bar{L}_H$ approaches the entropy rate of the source [4].

2.3.4. Arithmetic Coding

Arithmetic coding is a direct consequence of Shannon's source coding theorem (Section 2.3.1) which established that the number of bits per symbol needed to encode the output of a source is arbitrarily close to the entropy of the source [1]. In order to prove

his theorem, Shannon considered the following abstract coding scheme: List all likely sequences of symbols and assign to each a binary code with the more probable sequences being assigned shorter codewords. The practical implementation of this coding method is called Arithmetic coding. Arithmetic coding is especially useful when dealing with sources with small alphabets, such as binary sources, and alphabets with highly skewed probabilities (a good example can be written text sources).

Arithmetic coding is able to work most efficiently in the largest number of circumstances and stands out in terms of elegance, effectiveness, and versatility [22]. Some its most desirable features are

- When applied to independent and identically distributed (i.i.d.) sources, the compression of each symbol is provably optimal.
- It simplifies automatic modeling of complex sources, yielding near-optimal or significantly improved compression for sources that are not i.i.d.
- It is effective in a wide range of situations and compression ratios. The same Arithmetic coding implementation can effectively code all the diverse data created by the different processes such as modeling parameters, transform coefficients, and signaling.
- Its main process is Arithmetic, which is supported with ever-increasing efficiency by all general-purpose or digital signal processors (DSPs).

Arithmetic coding assigns integer codes to the individual symbols by assigning one (normally long) code to the entire input file. The method starts with a certain interval, it reads the input file symbol by symbol, and it uses the probability of each symbol to narrow the interval. Specifying a narrower interval requires more bits, so the number constructed by the algorithm grows continuously. To accomplish compression, the algorithm is designed such that a high-probability symbol narrows the interval less than a low-probability symbol, with the result that high-probability symbols contribute fewer bits to the output.

Essentially Arithmetic coding, codes not individual symbols but sequences of symbols. In order to distinguish a sequence of symbols from another sequence of symbols a

unique identifier tag is used. One possible set of tags for representing sequences of symbols, are the numbers in the unit interval [0,1). Because the number of numbers in the unit interval is infinite, it should be possible to assign a unique tag to each distinct sequence of symbols. Mapping of sequences of symbols in to the unit interval is achieved using the cumulative distribution function (cdf) of the random variable associated with the source output.

2.3.4.1. Arithmetic Encoding and Decoding

In Arithmetic coding, a unique identifier or tag is generated for the sequence to be encoded. This tag corresponds to a binary fraction, which becomes the binary code for the sequence. In practice, the generation of the tag and the binary code are the same process.

2.3.4.2. Generating a Tag

Let Ω be a discrete source that outputs symbols a_k form a finite alphabet $\mathcal{A} = \{a_1, a_2, \dots, a_N\}$ with probability $\mathcal{P} = \{p_1, p_2, \dots, p_N\}$.

The random variable X maps the source output symbols(letters) to numbers:

$$X(a_k) = k,\ a_k \in \mathcal{A} \tag{2.24}$$

The probability density function (*pdf*) of random variable X can be defined as

$$P(X = k) = P(a_k) = p_k \tag{2.25}$$

and the cumulative distribution function (*cdf*) of random variable X is

$$F_X(k) = \sum_{i=1}^{k} P(X = i) = \sum_{i=1}^{k} p_i \tag{2.26}$$

The procedure for generating the tag works by reducing the size of the interval in to subintervals in which the tag resides as more and more elements of the sequence are generated. The minimum and maximum value of the *cdf* correspond to the unit interval [0,1). The subinterval $\big[F_X(k-1), F_X(k)\big)$, is associated with the symbol a_k.

In the beginning the unit interval [0,1) is divided in into subintervals of the form $\big[F_X(k-1), F_X(k)\big),\ k = 1, \dots, N$. One of this subintervals is then partitioned in exactly

the same proportions as the original interval depending on the next symbol of the sequence. For example if the first symbol was a_n, the tag interval is taken as $\left[F_X(n\ -\ 1), F_X(n)\right)$, and if symbol a_m follows symbol a_n in the particular sequence of symbols $a_n a_m$, the interval containing the new tag becomes:

$$\left[F_X(n\ -\ 1) + F_X(m\ -\ 1) * \left(F_X(n\ ) - F_X(n\ -\ 1)\right), F_X(n-1) + F_X(m\ ) * \left(F_X(n\ ) - F_X(n\ -\ 1)\right)\right)$$

Each succeeding symbol causes the tag to be restricted to a subinterval that is further partitioned in the same proportions. The interval containing the tag value for a given sequence is disjoint from the intervals containing the tag values of all other sequences. This implies that it is only sufficient to compute upper and lower limits of the interval containing the tag and any value in this interval would be a unique identifier or tag. The upper (u^i) and lower (l^i) limits of the tag interval can be calculated recursively as follows [1]:

For any sequence of values of the random variable $\boldsymbol{x} = x_1 x_2 \dots x_M$: $x_i = X(a_k) = k$, $a_k \epsilon\ \mathcal{A}$ and $k = 1,2, \dots, N$

$$l^i = l^{i-1} + \left(u^{i-1} - l^{i-1}\right)F_X(x_i\ -\ 1) \tag{2.27}$$

$$u^i = l^{i-1} + \left(u^{i-1} - l^{i-1}\right)F_X(x_i\ ) \tag{2.28}$$

where $l^0 = 0$, $u^0 = 1$ and $i = 1,2 \dots, M$

The division of intervals in to sub intervals creates a precision problem when the length of the sequence m becomes large. To solve this problem encoding is performed incrementally, i.e. to transmit portions of the code as the sequence is being observed, rather than waiting for the entire sequence. Incremental encoding is implemented through rescaling operations. There are three types of rescaling operations.

i. If the interval is entirely confined to the lower half of the unit interval $[0, 0.5)$
   Rescale: $[0, 0.5) \rightarrow [0, 1)$; $l^i = 2l^i, u^i = 2u^i$

ii. If the interval is entirely confined to the lower half of the unit interval $[0.5, 1)$
   Rescale: $[0.5, 1) \rightarrow [0, 1)$; $l^i = 2(l^i - 0.5), u^i = 2(u^i - 0.5)$

iii. If the interval straddles the midpoint of the unit interval $[0.25, 0.75)$
   Rescale: $[0.25, 0.75) \rightarrow [0, 1)$; $l^i = 2(l^i - 0.25), u^i = 2(u^i - 0.25)$

2.3.4.3. Generating Binary Code

Subdividing and rescaling the unit interval [0,1) is not an attractive way to implement Arithmetic coding in software since it requires handling floating-point numbers. An alternative approach is to use integer Arithmetic to generate upper and lower limits of the interval. The binary code can be generated from the binary representations of l^i and u^i at each step together with rescaling operations.

The first step in integer implementation is to find the binary word length m that is sufficient to represent a sequence $\boldsymbol{x} = x_1 x_2 \dots x_M$ of length $M = Total\_Count$ [1]:

$$m = 2 + \lceil \log_2 M \rceil = 2 + \lceil \log_2 Total\_Count \rceil \tag{2.29}$$

where $\lceil x \rceil$ defines the next highest integer greater than x.

Given the word length m, interval [0,1) can be mapped to the range of 2^m binary words. The point 0 is mapped to $(\overbrace{00\dots0}^{m\ times})_2$ and 1 is mapped to $(\overbrace{11\dots1}^{m\ times})_2$. The midpoint of the interval 0.5 gets mapped to $(1\overbrace{00\dots0}^{m-1\ times})_2$.

If symbol a_k occurs n_k times in the sequence $\boldsymbol{x}$,the *cdf* $F_X(k)$ can be estimated by [1]:

$$F_X(k) = \frac{\sum_{i=1}^{k} n_i}{M} = \frac{Cum\_Count(k)}{Total\_Count} \tag{2.30}$$

Using Equation (2.30), Equations (2.27) and (2.28) can be written as [1]:

$$l^i = l^{i-1} + \left\lfloor \frac{\left(u^{i-1} - l^{i-1}\right) * Cum\_Count(x_i - 1)}{Total\_Count} \right\rfloor \tag{2.31}$$

$$u^i = l^{i-1} + \left\lfloor \frac{\left(u^{i-1} - l^{i-1}\right) * Cum\_Count(x_i)}{Total\_Count} \right\rfloor - 1 \tag{2.32}$$

where $\lfloor x \rfloor$ defines the largest integer less than or equal to x.

The decoding procedure recovers the original symbols in the same sequence they were encoded. A tag t is computed using the first m bits of binary stream . If the tag value lies with a specific tag interval of a symbol, then the symbol is decoded. The lower and upper limits are updated by the symbol's cumulative count using Equations (2.31) and

(2.32). The tag is also updated using the next sequence of bits from the bit stream. The encoding and decoding algorithms can be summarized as follows:

MSB stands for Most Significant Bit and LSB stands for Least Significant Bit

ENCODER

Initialize $l = 0,\ u = 2^m,\ s = 0$

Get symbol

$$u \leftarrow l + \left\lfloor \frac{(u - l + 1) * Cum\_Count(x)}{Total\_Count} \right\rfloor - 1$$

$$l \leftarrow l + \left\lfloor \frac{(u - l + 1) * Cum\_Count(x\ -\ 1)}{Total\_Count} \right\rfloor$$

while(MSB of u and l are both equal to b or second MSB of $u = 0$ and $l = 1$)

```
if(MSB of u and l are both equal)
      send b
      shift l to the left by 1 bit and shift 0 into LSB
      shift u to the left by 1 bit and shift 1 into LSB

      while(s > 0)
            send complement of b
            decrement s

if(second MSB of u = 0 and l = 1)
      shift l to the left by 1 bit and shift 0 into LSB
      shift u to the left by 1 bit and shift 1 into LSB
      complement (new) MSB of l and u
      increment s
```

DECODER

Initialize $l = 0,\ u = 2^m,\ k = 0$

Read the first m bits of the received bitstream into tag t.

while $\left(\left\lfloor \frac{(t - l + 1) * Total_{Count} - 1}{Total\_Count} \right\rfloor \geq Cum\_Count(k) \right)$

$k \leftarrow k + 1$

Decode symbol x

$$u \leftarrow l + \left\lfloor \frac{(u - l + 1) * Cum\_Count(x)}{Total\_Count} \right\rfloor - 1$$

$$l \leftarrow l + \left\lfloor \frac{(u - l + 1) * Cum\_Count(x - 1)}{Total\_Count} \right\rfloor$$

```
while(MSB of u and l are both equal to b or second MSB of u = 0 and l = 1)
      if(MSB of u and l are both equal)
            shift l to the left by 1 bit and shift 0 into LSB
            shift u to the left by 1 bit and shift 1 into LSB
            shift t to the left by 1 bit and read next bit from received
            bitstream into LSB
if(second MSB of u = 0 and l = 1)
            shift l to the left by 1 bit and shift 0 into LSB
            shift u to the left by 1 bit and shift 1 into LSB
            shift t to the left by 1 bit and read next bit from received
            bitstream into LSB
            complement (new) MSB of l,u, and t
```

2.3.4.4. Length of Arithmetic Codes

The average length of an Arithmetic code $\bar{L}_{A_m}$ for a sequence $\mathrm{X^m}$ of length m is bounded by [1]:

$$H(\mathrm{X^m}) \leq \bar{L}_{A_m} < H(\mathrm{X^m}) + 2 \tag{2.33}$$

The length per symbol $\overline{L_A}$, using Equation (2.22) becomes [1]:

$$H(X) \leq \bar{L}_A < H(X) + \frac{2}{m} \tag{2.34}$$

Similar to Huffam codes, a code rate close to the entropy can be guaranteed by increasing the length of the sequence.

2.3.5. Comparison of Huffman and Arithmetic Codes

For both Huffam and Arithmetic coding it is more efficient to generate codewords for groups or sequences of symbols rather than to generate a separate codeword for each symbol in a sequence. However, this approach becomes impractical when one tries to obtain Huffman codes for long sequences of symbols. In order to find the Huffman codeword for a particular sequence of length m, it requires to generate codewords for all possible sequences of length m. This fact causes an exponential growth in the size of the codebook. In contrast, a unique Arithmetic code can be generated for a sequence of length m without the need for generating codewords for all sequences of the same

length. In practice, we can make m large for the Arithmetic coder and not for the Huffman coder. This means that for most sources rates get closer to the entropy using Arithmetic coding than by using Huffman coding [1].

The Huffman method is simple, efficient, and produces the best codes for the individual symbols. However, the only case it is ideally optimal (codes whose average length equals the entropy) is when the symbols have probabilities of occurrence that are negative powers of 2 ($p_k = 2^{-k}$). This is because the Huffman method assigns a code with an integral number of bits to each symbol in the alphabet. From Shannon's source coding theorem a symbol with probability 0.4 should ideally be assigned a $-\log_2 0.4 \approx 1.32$-bit code. The Huffman method, however, normally assigns such a symbol a code of 1 or 2 bits. Arithmetic coding overcomes the problem of assigning integer codes to the individual symbols by generating one long code for a sequence of symbols [10].

Finally, it is much easier to adapt Arithmetic codes to changing input statistics. The estimate of the probabilities of the input alphabet can be computed by simply keeping a count of the symbols as they are coded. There is no need to generate a code a priori, as in the case of Huffman coding. This property allows to separate the modeling and coding procedures in a manner that is not very feasible with Huffman coding. The separation permits greater flexibility in the design of compression systems, which can be used to great advantage [1].

2.4. Lossless Source Encoding Performance Measures

The following measures are commonly used to express the performance of source encoding (compression) algorithms [1];

2.4.1. Compression Ratio

The compression ratio (CR) is ratio of the number of bits required to represent the data before compression to the number of bits required to represent the data after compression [1, 3];

$$Compression\ Ratio\ (CR) = \frac{Uncompressred\ Size}{Compressed\ Size} \tag{2.35}$$

2.4.2. Compression Rate

The compression rate is the average number of bits required to represent a certain sample. Its measure is always depending on how the sample units are represented. For example, for text source the rate is how many bits are required per each symbol ($bits/symbol$) and for image sources it is given in terms of pixels ($bits/pixel$).

$$Compression\ Rate = \frac{Compressed\ Size}{Number\ of\ Sample\ Units} \tag{2.36}$$

2.4.3. Space Savings

The average amount of space saved (percentage reduction in file size) after compression is given by [1, 3]:

$$Space\ Savings = \left(1 - \frac{1}{CR}\right) * 100\% \tag{2.37}$$

2.4.4. Compression and Decompression Speed

Speed is evaluated in terms of uncompressed bits processed per second.

$$Compression\ Speed = \frac{Uncompressed\ Bits}{Seconds\ to\ Compress} \tag{2.38}$$

$$Decompression\ Speed = \frac{Uncompressed\ Bits}{Seconds\ to\ Decompress} \tag{2.39}$$

3. Prediction by Partial Match Context-Modeling Algorithm

Huffman and Arithmetic coding which are presented in the previous chapter are known as entropy coders. These methods generate binary codes very efficiently in accordance with the source coding theorem. The average length of a code is lower bounded by the entropy of the source. The length of codeword assigned to a particular symbol is a function of its probability of occurrence. As mentioned in Section 2.2 there are two ways of assigning probabilities: frequency and context. If an external model with low entropy can be built external to the coding algorithms compression can be greatly improved. This chapter focuses on one of the best modelers especially for multiple alphabet sources, like written text sources, called prediction by partial match (PPM).

3.1. PPM Basic Principles

PPM is a compression method first proposed by John Cleary and Ian Witten [8, 10], with extensions and implementation by Alistair Moffat [9, 10]. It uses a partially blended finite-context model in combination with Arithmetic coding. Not only it can be a good approximation to the entropy but also helps to incrementally improve coding which makes it a good fit with Arithmetic coding.

Context models try to capture the correlation between successive symbols and provide an approximate model to text sources. Using higher-order Markov model provides with an improved compression ratio. But in adaptive coding, initially coding efficiency suffers because not enough statistics is available to build higher-order models. To solve this problem, PPM uses a "*partial match*" strategy, where a higher-order model is formed but used for lower order predictions in cases when high-order ones are not yet available [2, 8].

During the adaptive process of estimating symbol probabilities, especially at the beginning, symbols that have not been seen before for any context are encountered:

zero frequency problem [1, 2]. In order to handle this situation in PPM the source coder alphabet contains a special "*escape (esc)*" symbol, which is used to signal that the symbol to be encoded has not been seen in the current context.

3.1.1. PPM Blending Algorithm

PPM utilizes partial blending mechanism where higher-order predictions are preferred. The basic algorithm initially attempts to use the longest available context. The order (length) K of the longest context is predetermined. When the symbol S to be encoded has not previously been encountered in the current context C of *order-*K, an escape symbol is encoded and the PPM encoder switches to a next shorter context $C'$of *order-*$(K-1)$. C' is a string consisting of the rightmost $K-1$ symbols of C. This process continues until either a context that has a probability estimate of symbol S is found or at a point where an estimate is not found in any sub-context C' (*order-*$(K-1)$… *order-*0) of the current context C. In the later case, the default order minus one (*order-*(-1)) model is used encode each symbol with probability $1/M$, where M is the size of the source alphabet.

3.1.2. Escape Symbol

While assigning probability to symbols in a certain context PPM reserves some probability space to novel symbols that haven't occurred in the context. This allows to help continue encoding since Arithmetic coding requires for all symbols to have non-zero probability. In addition, the PPM decoder can follow the encoder while looking for a symbol in the perspective context. When the encoder decides to switch to a shorter context, it first encodes the escape symbol. The decoder can decode the escape symbol, since it is encoded in the present context. After decoding an escape, the decoder also switches to a shorter context.

For each context C of length $k \leq K$, PPM allocates a probability $\mathrm{P}_k(esc|C)$ for all symbols that did not appear after the context C and the remaining $1-\mathrm{P}_k(esc|C)$ is distributed among all other symbols that have non-zero counts for this context. The escape probability, estimate of the probability that a particular context will be followed by a character that has never before followed it, determines how much code space

should be allocated to the possibility of shifting to the next smaller context. Thus, the probability estimate should decrease depending on the characters observed in that context. Suppose that context C was seen ten times in the past and was always followed by symbol S. This shows that the likelihood of same context will be followed by the same symbol S in the future is high, so the encoder will only rarely have to switch down to a lower context. The escape symbol can therefore be assigned the small probability. However, if every occurrence of context C in the past was followed by a different symbol (suggesting that the text varies a lot), then there is a good chance that the next occurrence will also be followed by a different symbol, forcing the encoder to switch to a lower context (and thus to emit an escape) more often. The escape is therefore assigned a higher probability here.

3.1.3. PPM Symbol Probabilities

Let Ω be a text source with a discrete alphabet

$$\mathcal{A} = \{a_1, a_2, \dots, a_M\} \tag{3.1}$$

consisting M symbols with $M \geq 2$

Define: $\alpha^n -$ a sequence of n symbols taken from some part (sample) of the source(text)

$$\alpha^n = \{\alpha_1 \alpha_2 \dots \alpha_n\} : \ \alpha_i = \ a_m \in \ \mathcal{A}, m = 1,2, \dots, M \ \& \ i = 1,2, \dots, n$$

K – the order of the longest context (maximum order of a Markov model)

$C_k -$ a context with length $k \leq K$

Let's assume the first $j-1$ symbols $\alpha_1 \alpha_2 \dots \alpha_{j-1}$ of the sequence α^n are already processed.

The probability of the next symbol α_j in the sequence α^n with respect to context C_k,

$$C_k = \alpha_{j-k} \alpha_{j-k+1} \dots \ \alpha_{j-2} \alpha_{j-1} \tag{3.2}$$

which contains the previous k symbols before α_j, is [18]:

$$\mathrm{P}(\alpha_j | C_k) = \begin{cases} \mathrm{P}_k(\alpha_j | C_k) & \text{if } C_k \alpha_j \text{ appeared in the text before} \\ \mathrm{P}_k(esc | C_k) \mathrm{P}(\alpha_j | C_{k-1}) & \text{Othewise} \end{cases} \tag{3.3}$$

where

$$\mathrm{P}_k(\alpha_j|\, C_k) = \frac{\mathbb{C}_{C_k}(\alpha_j)}{\mathbb{T}_{C_k} + \mathbb{C}_{C_k}(esc)} \tag{3.4}$$

$$\mathrm{P}_k(esc|C_k) = \frac{\mathbb{C}_{C_k}(esc)}{\mathbb{T}_{C_k} + \mathbb{C}_{C_k}(esc)} \tag{3.5}$$

$$\mathbb{T}_{C_k} = \sum_{a_m \in \mathcal{A}_{C_k}} \mathbb{C}_{C_k}(a_m) \tag{3.6}$$

$$\mathcal{A}_{C_k} = \{a_m : \mathbb{C}_{C_k}(\alpha_m) > 0, m = 1,2 \dots, M\} \tag{3.7}$$

$\mathrm{P}_k(\alpha_j|C_k)$: symbol α_j's probability in context C_k

$\mathrm{P}_k(esc|C_k)$: escape probability in context C_k

$\mathbb{C}_{C_k}(\alpha_j)$: is the count of symbol α_j in context C_k (number of times $C_k\alpha_j$ appeared)

$\mathbb{C}_{C_k}(esc)$: is the count of the escape symbol (depends on the PPM Variant)

$\mathbb{T}_{C_k}$: total count of symbols in context C_k

$\mathcal{A}_{C_k}$: alphabet containing distinct symbols that followed context C_k

C_{k-1}: is the next smaller sub-context of C_k length $k-1$

PPM encoder uses frequency counts ($\mathbb{C}_{C_k}$) of symbols in each context to provide cumulative probability to an Arithmetic encoder. As an example, Table-3.1 shows the assignment of symbol probabilities $\mathrm{P}_k(\alpha_j|C_k)$ for the phrase "ለምለም ፈለፈለች" (particularly chosen because of repetition of symbols). The space between words is represented by "␣". The longest context order is assumed to use the previous two symbols ($K = 2$). The escape symbol is assigned a count equal to the number of symbols that appeared in the respective context. This particular way of calculating escape symbol probability is called Method C.

3.2. The Exclusion Principle

The basic idea behind Arithmetic coding is the division of the unit interval into subintervals, each of which represents a particular symbol. The smaller the subinterval,

Table-3.1 : PPM model for string "ለምለም␣ፊለፊለች"
(a) Order-2 (b) Order-1 (c) Order-0

Order $k=2$			
context	symbol	$\mathbb{C}_{C_2}$	P_2
ለም	ለ	1	1/4
	␣	1	1/4
	esc	2	2/4
ምለ	ም	1	1/2
	esc	1	1/2
ም␣	ፊ	1	1/2
	esc	1	1/2
␣ፊ	ለ	1	1/2
	esc	1	1/2
ፊለ	ፊ	1	1/4
	ች	1	1/4
	esc	2	2/4
ለፊ	ለ	1	1/2
	esc	1	1/2

(a)

Order $k=1$			
context	symbol	$\mathbb{C}_{C_1}$	P_1
ለ	ም	2	2/7
	ፊ	1	1/7
	ች	1	1/7
	esc	3	3/7
ም	ለ	1	1/4
	␣	1	1/4
	esc	2	2/4
␣	ፊ	1	1/2
	esc	1	1/2
ፊ	ለ	2	2/3
	esc	1	1/3

(b)

Order $k=0$			
context	symbol	$\mathbb{C}_{C_0}$	P_0
	ለ	4	4/15
	ም	2	2/15
	␣	1	1/15
	ፊ	2	2/15
	ች	1	1/15
	esc	5	5/15

(c)

the more bits are required to distinguish it from other subintervals. If the number of symbols to be represented is reduced, the number of subintervals goes down as well. This in turn means that the sizes of the subintervals increase, leading to a reduction in the number of bits required for encoding. The exclusion principle is a smoothing technique used in PPM to provide this kind of reduction in rate. Exclusion enhances the escape estimation by removing symbols that occurred in higher-order contexts in lower-order probability estimation calculations.

Let's assume the string "ለምለም␣ፊለ"is already encoded in the previous example. Symbol probabilities are given in Table-3.2. The next symbol to be coded in the sequence is "ፊ". With the assumption that the longest context length $K = 2$, the encoder looks for context "ፊለ" in order-2 entries in Table-3.2a . Context "ፊለ" has not occurred previously

so the encoder jumps to the next smaller context "ለ" (order-1, Table-3.2b). Even if context "ለ" occurred already symbol "ፈ" hasn't occurred with it. The encoder now encodes an escape here and switches to the next lower order context (order-0, Table-3.2c). Here "ፈ" is found with a previous count of 1. While calculating the probability of symbol "ፈ", using the exclusion principle the count of symbol "ም" can be excluded

Table-3.2 : PPM model for string "ለምለም␣ፈለ" without exclusions
(a) Order-2 (b) Order-1 (c) Order-0

Order $k=2$			
context	symbol	$\mathbb{C}_{C_2}$	P_2
ለም	ለ	1	1/4
	␣	1	1/4
	esc	2	2/4
ምለ	ም	1	1/2
	esc	1	1/2
ም␣	ፈ	1	1/2
	esc	1	1/2
␣ፈ	ለ	1	1/2
	esc	1	1/2

(a)

Order $k=1$			
context	symbol	$\mathbb{C}_{C_1}$	P_1
ለ	ም	2	2/3
	esc	1	1/3
ም	ለ	1	1/4
	␣	1	1/4
	esc	2	2/4
␣	ፈ	1	1/2
	esc	1	1/2
ፈ	ለ	1	1/2
	esc	1	1/2

(b)

Order $k=0$			
context	symbol	$\mathbb{C}_{C_0}$	P_0
	ለ	3	3/11
	ም	2	2/1
	␣	1	1/11
	ፈ	1	1/11
	esc	4	4/11

(c)

(set count to zero) since it appeared in the order-1 context "ለ". The new probability estimations with exclusions for order-0 are given in Table-3.3.

Using the new probability estimates of Table-3.3, according to the source coding theorem, the codeword length for encoding symbol "ፈ" is:

esc in order-1 context "ለ": $-\log_2 \frac{1}{3} = 1.585\ bits$

symbol "ፈ" in order-0 : $-\log_2 \frac{1}{9} = 3.17\ bits$

Total: $1.585 + 3.17 = 4.755\ bits$

Table-3.3 : Order-0 PPM model for string "ለምለም_ፈለ" with exclusions

Order $k = 0$			
context	symbol	$\mathbb{C}_{C_0}$	P_0
	ለ	3	3/9
	ም	1	1/9
	_	1	1/9
	ፈ	1	1/9
	esc	4	4/9

For the case without exclusions using Table-3.2,

esc in order-1 context "ለ": $-\log_2 \frac{1}{3} = 1.585\ bits$

symbol "ፈ" in order-0 : $-\log_2 \frac{1}{11} = 3.46\ bits$

Total: $1.585 + 3.46 = 5.045\ bits$

Comparing the above two results, using exclusions, symbol "ፈ" can be encoded with 4.585 $bits$ instead of the original 5.045 $bits$ saving 0.29 $bits$. The cumulative effect of this exclusion of symbols from contexts on a temporary basis can result in significant improvements in final compression performance.

Exclusion is also useful while updating models. This type of exclusion known as *update exclusion* doesn't increment the count for a predicted symbol if it is already predicted by higher-order context. The effect of update exclusions is to use the uniqueness of a symbol in a particular context for prediction instead of its frequency. Update exclusions haven been found to improve compression ratio by 2% [2].

3.3. PPM Variants

There are four variants of PPM based on the assignment of escape counts. There is no theoretical basis for choosing the escape probability optimally [2]. The methods have been selected based on vast experience that developers had with data compression.

3.3.1. PPMA

Method A simply assigns a count of one to the escape symbol in each context ($\mathbb{C}_{C_k}(esc) = 1$). This gives the escape probability [2]:

$$\mathrm{P}_k(esc|C_k) = \frac{1}{\mathbb{T}_{C_k} + 1} \tag{3.8}$$

The probability assigned to the other symbol α_j (different from the escape symbol) in context C_k becomes [2]:

$$\mathrm{P}_k(\alpha_j|C_k) = \frac{\mathbb{C}_{C_k}(\alpha_j)}{\mathbb{T}_{C_k} + 1} \tag{3.9}$$

3.3.2. PPMB

Method B treats the appearance of a new symbol in a context as a rare event when a few symbol(s) followed a particular context beforehand [2]. It refrains from predicting characters unless they have occurred more than once in the present context. This is done by subtracting one from all counts and giving the count to the escape symbol.

Let n_{C_k} be the number of distinct symbols that have occurred in context C_k. The escape probability using Method B is [2]:

$$\mathrm{P}_k(esc|C_k) = \frac{n_{C_k}}{\mathbb{T}_{C_k}} \tag{3.10}$$

Before allowing code space for escape symbol, the probability of each symbol α_j is [2]:

$$\mathrm{P}_k(\alpha_j|C_k) = \frac{\mathbb{C}_{C_k}(\alpha_j) - 1}{\mathbb{T}_{C_k} - n_{C_k}} \tag{3.11}$$

After allowing for the escape symbol, the code space allocated for symbol α_j is [2]:

$$\mathrm{P}_k(\alpha_j|C_k) = \frac{\mathbb{C}_{C_k}(\alpha_j) - 1}{\mathbb{T}_{C_k}} \tag{3.12}$$

3.3.3. PPMC

In Method C, the count assigned to the escape symbol is the number of symbols that have occurred in that context ($\mathbb{C}_{C_k}(esc) = n_{C_k}$), similar to Method B. The difference comes in the fact that, instead of subtracting this from the counts of individual symbols, the total count is inflated by this amount. The escape and symbol probabilities respectively become [2, 22];

$$\mathrm{P}_k(esc|C_k) = \frac{n_{C_k}}{\mathbb{T}_{C_k} + n_{C_k}} \tag{3.13}$$

$$\mathrm{P}_k(\alpha_j|C_k) = \frac{\mathbb{C}_{C_k}(\alpha_j)}{\mathbb{T}_{C_k} + n_{C_k}} \tag{3.14}$$

3.3.4. PPMD

In this implementation of the PPM called Method D (Double): symbol α_j's frequency is incremented by 2 in a certain context C_k when $\mathbb{C}_{C_k}(\alpha_j) > 0$,otherwise this 2 increment is divided between the probability of the current symbol and the probability of the escape symbol.

The escape symbol gets a probability [22]:

$$\mathrm{P}_k(esc|C_k) = \frac{n_{C_k}}{2\mathbb{T}_{C_k}} \tag{3.15}$$

and probability of symbol α_j is [22]:

$$\mathrm{P}_k(\alpha_j|C_k) = \frac{2\mathbb{C}_{C_k}(\alpha_j) - 1}{2\mathbb{T}_{C_k}} \tag{3.16}$$

There is no obvious method of assigning counts to the escape symbol. There are some variations in performance depending on the data being encoded. For text sources PPMD and PPMC seem to provide the best compression ratio [2 ,9, 15, 22].

3.4. PPM Data Structure

During the encoding process PPM maintains a data structure where all contexts (orders-0 to K) of every symbol read for the text are stored. The structure is a special type of tree known as *trie*. A trie is an ordered prefix tree used to store partial string contexts. Each node in a trie is labeled with a symbol and stores the number of times the symbol occurred. Using a trie, it is possible to very quickly access and update the longest context seen so far which contains the current character by descending the tree until we hit a leaf node. Looking up a key of length ℓ takes at most $O(\ell)$ time, as opposed to a binary search tree which is $O(log\ n)$ if n counts have been stored in the

tree. For a large number of short strings, tries are space efficient because the contexts are not stored explicitly and nodes are shared between contexts.

Figure-3.1 shows a trie constructed for the string "ለምለም_ፈለፈለች" assuming $K = 2$. The tree grows in width but not in depth. Its depth remains $K + 1 = 3$ regardless of how much input data has been read. Its width grows as more and more symbols are input, but not at a constant rate.

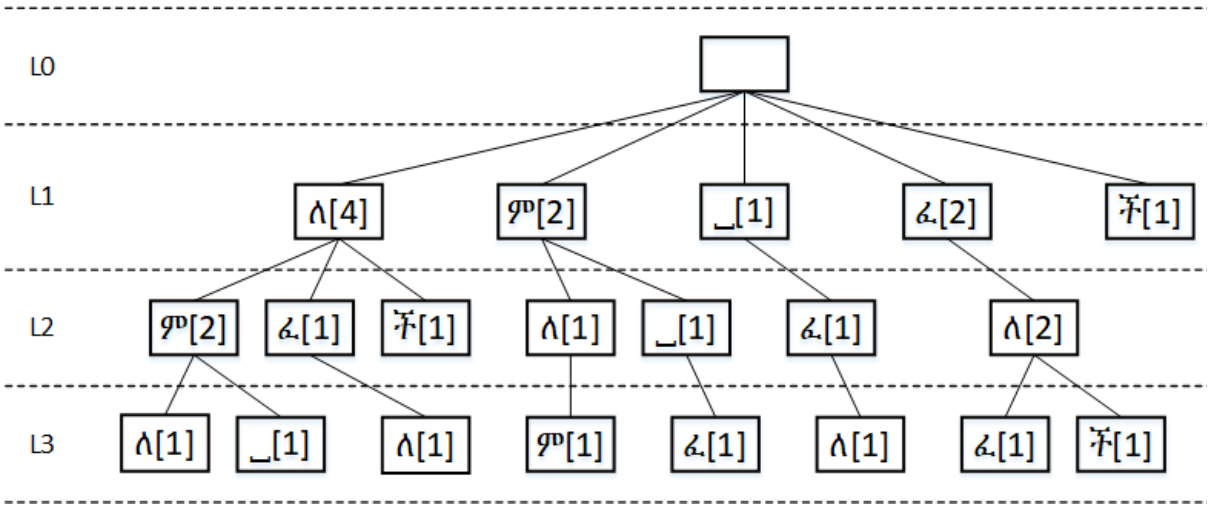

Figure-3.1 : Context trie for string "ለምለም_ፈለፈለች"

The root node at the tip(apex) of the trie in *level*-L0 represents the order-0 context. Symbols that appear at *level*-L1 are the children of the root node. They represent the order-1 contexts and their counts are used to obtain the order-0 probability model. The order-2 contexts can be obtained by moving from the root node to its child, and from there to its appropriate grandchild. Thus the combination of *level*-L1 and *level*-L2 define an order-2 context. The order-1 symbol probability estimation can be calculated using counts in *level*-L2. In the same manner counts in *level*-L3 contribute to the order-2 symbol probabilities.

4. Results and Discussions

4.1. Experiment Setup

PPMC and PPMD versions of the PPM context modeling algorithm are implemented using *python*. The final coding is performed using integer implementation of Arithmetic coding. Eight books written in Amharic language are taken as test files which are presented in Table-4.1. The books are composed of fictions and non-fictions and have variable sizes.

Table-4.1 : Test Files

No	Book	Size in bytes
1	Tsehay ፀሀይ	154,453
2	Agere አገሬ	367,740
3	Tenbit ትንቢት	699,320
4	Yegazetegnaw Mastawesha የጋዜጠኛው ማስታወሻ	841,068
5	Fiker Eske Mekaber ፍቅር እስከ መቃብር	857,349
6	Yederasiw Mastawesha የደራሲው ማስታወሻ	1,033,812
7	New Testament አዲስ ኪዳን	1,370,536
8	Bible መጽሐፍ ቅዱስ	4,475,512

Python is chosen for its simplicity and strong support for Unicode string processing. Because it is dynamically typed it suffers from computational delay, thus results in slow compression and decompression speed. To account for this delay a C optimized implementation is used using *Cython*. Cyhton has syntax similar to python with some modifications. The Cyhton implementation reduced the execution time taken for compression and decompression by a factor of three.

4.2. Results

4.2.1. Compression Performance of PPMC and PPMD

The compressed file sizes starting from order 0 to 6 are presented in the following tables.

Table-4.2 : Compressed file size for PPMC with Exclusions

No.	Book	Compressed Size (bytes)						
		Order(*k*)						
		0	1	2	3	4	5	6
1	Tsehay ፀሀይ	39,702	28,492	**22,920**	22,952	23,225	23,353	23,462
2	Agere አገሬ	103,223	77,806	63,173	**61,593**	62,195	62,771	63,162
3	Tenbit ትንቢት	202,445	148,109	115,302	**111,418**	112,336	113,390	114,218
4	Yegazetegnaw Mastawesha የጋዜጠኛው ማስታወሻ	242,927	191,533	158,227	**154,090**	155,477	156,812	157,646
5	Fiker Eske Mekaber ፍቅር እስከ መቃብር	228,045	172,759	139,754	**138,089**	140,925	143,322	144,997
6	Yederasiw Mastawesha የደራሲው ማስታወሻ	298,569	234,288	193,669	**188335**	190,162	192,016	193,178
7	New Testament አዲስ ኪዳን	380,374	276,103	209,923	197,230	**195,735**	196,539	197,533
8	Bible መጽሐፍ ቅዱስ	1,255,487	903,296	667,145	612,418	598,596	**595,701**	596,823

Table-4.3 : Compressed file size for PPMD with Exclusions

No.	Book	Compressed Size (bytes)						
		Order						
		0	1	2	3	4	5	6
1	Tsehay ፀሀይ	39,662	28,400	22,787	**22,719**	22,931	23,078	23,176
2	Agere አገሬ	103,167	77,473	62,725	**61,107**	61,651	62,186	62,564
3	Tenbit ትንቢት	202,372	147,666	114,594	**110,568**	111,368	112,318	113,104
4	Yegazetegnaw Mastawesha የጋዜጠኛው ማስታወሻ	242,861	191,103	157,651	**153,403**	154,637	155,881	156,692
5	Fiker Eske Mekaber ፍቅር እስከ መቃብር	227,976	172,326	139,079	**136,964**	139,264	141,516	143,094
6	Yederasiw Mastawesha የደራሲው ማስታወሻ	298,495	233,662	192,618	**187,221**	188,890	190,651	191,766
7	New Testament አዲስ ኪዳን	380,304	275,619	209,124	196,157	**194,384**	194,709	195,420
8	Bible መጽሐፍ ቅዱስ	1,255,403	902,604	665,202	608,964	594,149	**589,741**	589,918

Figure-4.1 and 4.2 present a histogram plot of the above tables.

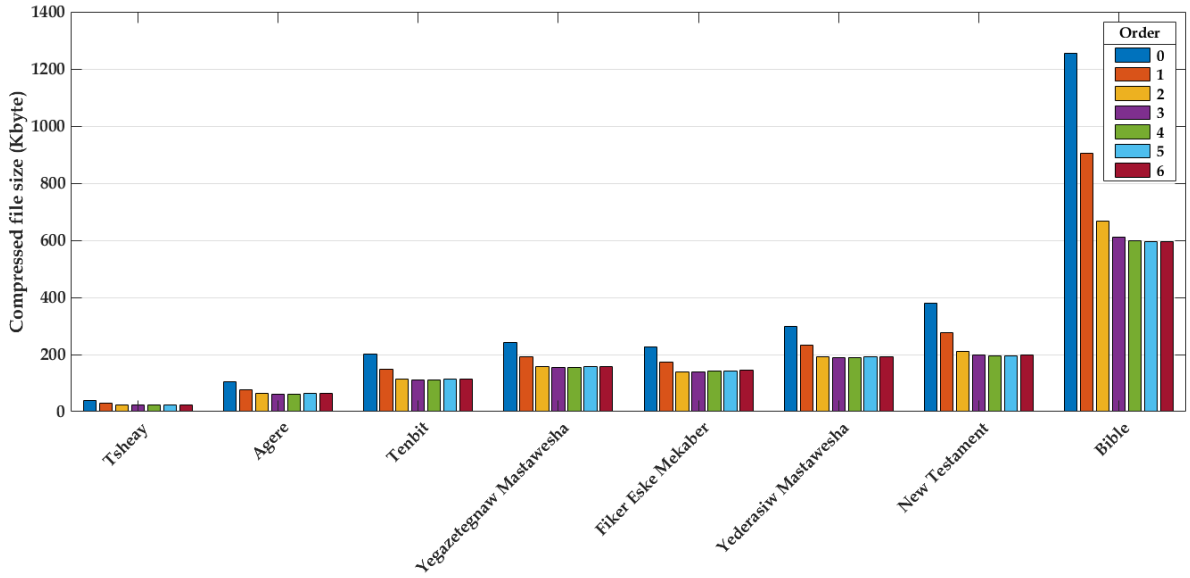

Figure-4.1 : PPMC Compression Performance

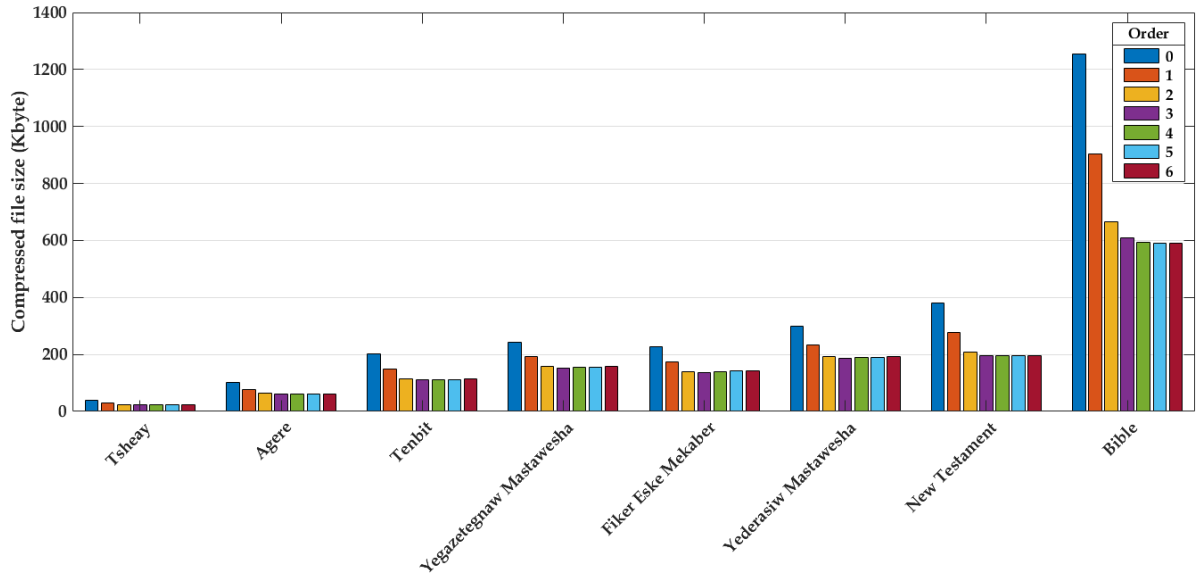

Figure-4.2 : PPMD Compression Performance

4.2.2. Average Compression Performance of PPMC and PPMD

Average compression performance of PPMC and PPMD over the eight books for orders 0 up to 10 is presented below:

Table-4.4 : PPMC with Exclusions Average Compression Performance

Performance Metrics	Order										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Compression Ratio	3.60	4.83	6.09	**6.315**	6.29	6.25	6.21	6.185	6.17	6.155	6.15
Space Savings (%)	72.22	79.3	83.58	**84.16**	84.10	83.99	83.89	83.83	83.78	83.75	83.73
Compression Rate (bits/symbol)	5.74	4.29	3.41	**3.29**	3.32	3.34	3.36	3.38	3.39	3.392	3.398

Table-4.5 : PPMD with Exclusions Average Compression Performance

Performance Metrics	Order										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Compression Ratio	3.60	4.84	6.13	**6.36**	6.345	6.31	6.275	6.25	6.23	6.22	6.21
Space Savings (%)	72.22	79.33	83.67	**84.27**	84.23	84.14	84.06	84	83.94	83.92	83.89
Compression Rate (bits/symbol)	5.735	4.28	3.39	**3.28**	3.29	3.31	3.33	3.345	3.35	3.36	3.362

4.3. Discussions

4.3.1. Optimum Order

Order-3 contexts provide best compression ratio on average. As shown in Figure-4.1, for both algorithms compression ratio rises sharply up to order-3 and falls slightly for increasing order. Thus, order-3 is the *optimum* order for efficient compression for Amharic text sources. The decrease in compression performance is expected which is also the case for other languages: for example, in English order-4 or order-5 is taken as the optimum order [1, 2]. The reason for this slight degradation is for relatively small orders (1 up 4) repetitions of contexts is much more likely than higher order contexts. Consequently, using higher order contexts requires coding of the escape symbol frequently due to the uniqueness(non-repetition) of contexts. If the context size grows indefinitely to large orders, the use of escape symbol to skip contexts becomes an overhead which makes incremental Arithmetic coding inefficient. This additional overhead creates the degradation in performance by increasing the size of the compressed file.

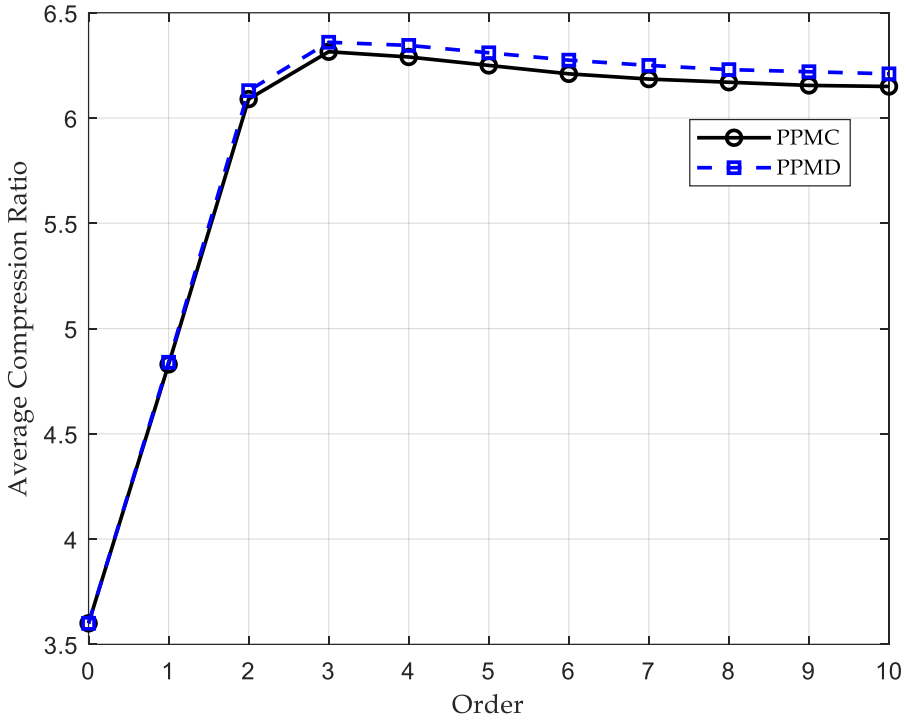

Figure-4.3 : Average Compression ratio with Order for PPMC and PPMD.

4.3.2. Compression Rate, Entropy and Redundancy

Compared to UTF-8 representation of Amharic symbols which uses 16 $bits/symbol$, at the optimum order-3 PPMC requires 3.29 $bits/symbol$ (Table-4.4) and PPMD requires 3.28 $bits/symbol$ (Table-4.5). There is a significant gap (around 12.7 $bits$) which shows the advantage gained by using context-modeling algorithms to code Amharic text sources. The results also show around 84.2% reduction in file size is obtained by using the PPM context modeling algorithm. This results are relatively improved over previous approaches implemented in [3] using N-gram with Huffman and Arithmetic coding obtaining 75.98% and 79.55% space savings for $N = 2$ and $N = 3$ respectively [3].

The other lens to look at the above results is in light of the theoretical entropy and redundancy estimations of Amharic language. Tesgamlak and Dereje estimated a lower bound entropy of the language to 1.074 $bits/symbol$ [3]. Figure-4.2 shows the average compression rate for both PPMC and PPMD in $bits/symbol$. At the optimal order-3, both algorithms require around 3.3 $bits/symbol$ which uses 2.23 $bits$ in excess of the estimated entropy of the language.

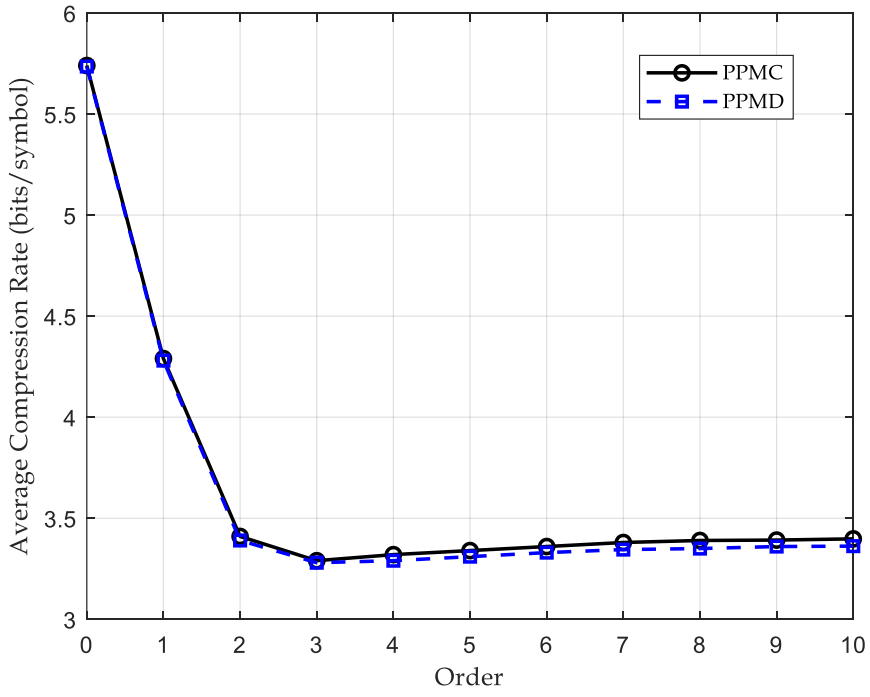

Figure-4.4 : Average Compression Rate with Order for PPMC and PPMD.

Similarly, according to Tesga and Dereje theoretically the redundancy of the Amharic language is 86.54% [3]. In this practical implementation of context-modeling, using Equation (2.16), the theoretical redundancy for optimum order-3 is

PPMC: $$R = 1 - \frac{H(A)}{H_{MAX}} = 1 - \frac{3.29}{\log_2 365} = 61.35\%$$

PPMD: $$R = 1 - \frac{H(A)}{H_{MAX}} = 1 - \frac{3.28}{\log_2 365} = 61.46\%$$

where $H_{MAX} = \log_2 N$, for a source with N symbols.

There are a total of 365 distinct symbols observed over the eight books resulting in $H_{MAX} = \log_2 365 = 8.51\ bits/symbol$.

4.3.3. Compression Speed

Though it is not the primary focus of the implementation used in the experiment, computational complexity of context-modeling algorithms highly affects compression and decompression speed. In general, the execution time it takes for compression and decompression increases with file size for all orders. Compression speed is optimal for order-2 and order-3 shown in Figure-4.3.

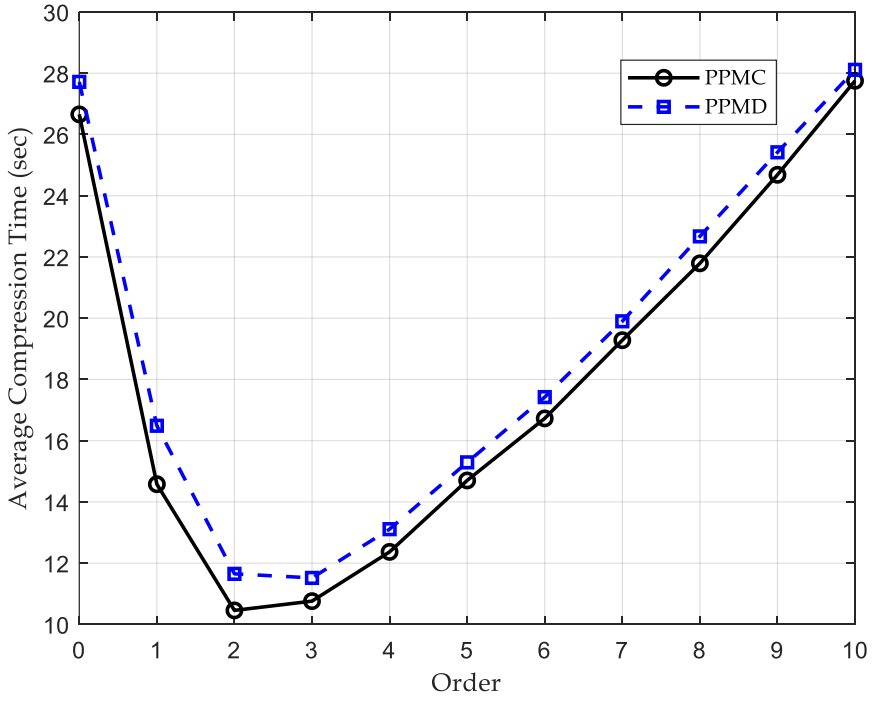

Figure-4.5 : Average Compression Time with Order for PPMC and PPMD.

5. Conclusions and Recommendations

5.1. Conclusions

This thesis aimed at analyzing context-modeling techniques for modeling Amharic language and coding of Amharic text sources. It had also the task of finding out how much compression can be achieved using these techniques. The best versions of the state of the art multiple alphabet bounded context-modeling algorithm PPM: PPMC and PPMD were successfully implemented. The algorithms are smoothed using the exclusion principle for optimal results.

The optimum order (context length) for efficient compression (source encoding) was shown to be order-3. This is lower compared to other languages in Table-1.1 for which the best compression is achieved for order-4 and above. Particularly a comparison can be drawn with the knowledge of the English language in contrast to the other languages given in Table-1.1. English has a small alphabet (26 symbols, 21 consonants and 5 vowels) compared to Amharic with 182 basic symbols (26 consonants with 7 variations). In English repetition of symbols and groups of symbols is more likely because there are only five vowels. In case of Amharic the probability of repetition of a particular sequence of symbols is less likely because each combination of a consonant with a vowel is represented by a unique symbol. The effect of this behavior is reflected in the results for which the best compression is achieved at a lower order (order-3) which implies groups of four symbols are frequently non-unique compared to the other groups in Amharic texts. This observation is a simplistic one and only tells half the story since frequency of symbols and contexts is also dependent on more complex features of languages like word formation and grammar rules which are difficult to measure and compare across different languages.

For the optimal order it was possible to achieve 84.2% reduction in file size which will significantly improve bandwidth utilization for communication applications and reduce resources required for storing Amharic text source files as compared to UTF-8 representation. It is shown in [3], simple adaptive models that are only based on frequency counts of symbols, order-0 model without context, result in slightly worse

results as compared to semi-adaptive models. To overcome this drawback PPM combines several Markov models of different order to estimate symbol probabilities and better capture the statistics of the text source and hence at the optimum order (order-3) a 45% improvement is observed over previous implementations of adaptive encoding for Amharic text sources.

The results obtained from both PPMC and PPMD show a 7.67% improvement in approaching the theoretical redundancy estimations of similar source coding implementations done in [3]. More redundancy can be extracted by modifying the PPM algorithm to learn and adapt speech tags and grammar rules. Although this tasks are computationally intensive and may not be well suited for source coding applications, they are very helpful in speech synthesis, word segmentation and machine translation.

The golden standard which all source coding algorithms strive it to get close to the entropy of the source. With this regard, the results presented in the previous chapter have a 2.23 $bit/symbol$ difference as compared to theoretical entropy estimations of Amharic language. Though a slight improvement over previously reported source coding results the gap shows there is still a room for improvement.

Compression and decompression speed are also optimal for order-3. This shows PPM can be used in moderately delay insensitive applications (e.g. email, SMS) with computationally efficient devices (multiprocessor based computers and smart phones) to save communication resources and reduce cost. PPM is also a very good candidate to archive text source files if decompression speed is not prime concern as in most commercial compression software.

Finally, this thesis presented the advantage that can be gained from employing context-modeling techniques for encoding of Amharic text sources. Mainly an improvement in compression ratio and compression rate are achievable. But there are mirroring drawbacks which come from computational complexity of the algorithms affecting speed of processing.

5.2. Recommendations

Modeling of natural languages is not a simple task. Even if they have their own rules they involve vast numbers of parameters that can be used in ways that introduce various ambiguities, yet can still be understood by other humans. As shown in this thesis, statistical approaches like context-modeling relay on Mrakov models to generate probabilistic models that are able to predict the next symbol in the sequence given the symbols that precede it. This approach is simple enough for source coding applications. But the study here can be extended using Neural Language Models (NLM) for language processing tasks in relation to machine learning such as speech recognition, optical character recognition, handwriting recognition, machine translation, spelling correction, image captioning, text summarization, etc.

While conducting studies related to source coding, it is helpful to establish a common corpora of source files (books or other publications for the case of text sources) in order to compare and justify results. For English language the Calgary, Canterbury and Brown corpus are used to test source encoding and compression performance. Since the work in this thesis and similar previous studies done on Amharic language are likely to be compared with future studies on Amharic or other local languages a common corpus is an ideal solution to close the gap in incompatibilities.

References

[1] K. Sayood, Introduction to Data Compression, Waltham, MA, USA: Morgan Kuffman, 2012.

[2] T. C. Bell, J. G. Cleary and I. H. Witten, Text Compression, Englewood Cliffs, NJ, USA: Prentice-Hall, 1990.

[3] T. Terefe and D. Hailemariam, "Entropy Estimation and Entropy-based Encoding of Written Amharic Language for Efficient Transmission in Telecom Networks," in *IEEE Africon 2017 Proceedings*, 2017.

[4] J. G. Proakis and M. Salehi, Communication Systems Engineering, 2nd ed., New Jersey, NJ: Prentice-Hall, 2002.

[5] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal,* vol. 27, pp. 379-423, July 1948.

[6] C. E. Shannon, "Prediction and Entropy of Printed English," *Bell System Technical Journal,* vol. 30, pp. 50-64, Jan 1951.

[7] D. S. Hirschberg and D. A. Lelewer, *Context Modeling for Text Compression,* Unpublished.

[8] J. G. Cleary and I. H. Witten, "Data Compression Using Adaptive Coding and Partial String Mathcing," *IEEE Transaction on Communications,* Vols. Com-34(4), pp. 396-402, April 1984.

[9] A. Moffat, "Implementing the PPM Data Compression Scheme," *IEEE Transaction on Communications,* Vols. COM-38(11), pp. 1917-1921, November 1990.

[10] D. Salomon and G. Motta, Handbook of Data Compression, 5th ed., London: Springer-Verlag, 2010.

[11] J. Navara, *Compression of natural Czech text,* Master's thesis, Dep. of Theoretical Comp. Sci., Czech Tech. Univ., Prague, Czech, June 2016.

[12] The Unicode Consortium, "The Unicode Standard Version 11," December 2018. [Online]. Available: http://www.unicode.org/charts. [Accessed 28 July 2018].

[13] A. Wimsatt and R. Wynn, "Amharic Language and Cultural Manual," 2011. [Online]. Available: http://languagemanuals.weebly.com/language-manuals-list.html.

[14] M. Yifiru, "Morphology-Based Language Modeling for Amharic," Ph.D. dissertation, Dep. Comp. Sci., Hamburg Univ., Hamburg, Germany, August 2010.

[15] K. M. Alhawiti, "Adaptive Models of Arabic Text," PhD Dissertation, Bangor University, Bangor,Wales, March 2014.

[16] W. J. Teahan, P. Wu and W. Liu, "Adaptive Compression-based Models of Chinese Text," in *IEEE 2014 International Conference on Audio, Language and Image Processing*, July 2014.

[17] L. B. Levitin and Z. Reingold, "Entropy of Natural Languages: Theory and Experiment," *Chaos, Solitons & Fracrals,* vol. 4, no. 5, pp. 709-743, 1994.

[18] R. Begleiter, R. El-Yaniv and G. Yona, "On Prediction Using Variable Order Markov Models," *Journal of Artificial Intelligence Research,* vol. 22, pp. 385-421, December 2004.

[19] A. Moffat, "A Word-based text compression," *Software Practice and Experience,* vol. 19, no. 2, pp. 185-198, February 1989.

[20] T. M. Cover and R. C. King, "A convergent gambling estimate of the entropy of English," *IEEE Transaction on Information Theory,* Vols. IT-24, no. 4, pp. 413-421, Jul y 1978.

[21] P.-N. Cheny and F. Alajaji, *Lecture Notes on Information Theory, Volume I,* August 2005.

[22] K. Sayood, Lossless Compression Handbook, San Dieg, California: Academic Press, 2003.

[23] J. G. Cleary and W. J. Teahan, "Unbounded length contexts for PPM," *The Computer Journal,* vol. 40, pp. 30-74, February 1997.

[24] W. J. Teahan and J. G. Cleary, "The entropy of English using PPM-based," in *Proceddings of Data Compression Conference*, 1996.

[25] D. A. Shkarin, "Improving the Efficiency of the PPM Algorithm," *Problems of Information Transmission,* vol. 37, no. 3, pp. 226-235, March 2001.

[26] D. L. Ruyet and M. Pischella, Digital Communications 1 Source and Channel Coding, London, UK: ISTE Ltd, 2015.

Appendix

Appendix I: PPMC Compression Ratio

Compression Ratio												
No	Book	Order										
		0	1	2	3	4	5	6	7	8	9	10
1	Tsheay ፀሀይ	3.8902	5.4207	6.7385	6.7288	6.65	6.6136	6.5828	6.5521	6.517	6.4904	6.4744
2	Agere አገሬ	3.5626	4.7264	5.8212	5.9705	5.9127	5.8584	5.8222	5.8	5.7873	5.7797	5.775
3	Tenbit ትንቢት	3.4544	4.7216	6.0651	6.2764	6.2252	6.1673	6.1226	6.0949	6.0783	6.068	6.0608
4	Yegezetengw Mastaesha የጋዜጠኛው ማስታወሻ	3.4622	4.3912	5.3139	5.4583	5.4096	5.3635	5.3352	5.3218	5.315	5.3116	5.3101
5	Fiker eske Mekabir ፍቅር እስከ መቃብር	3.8353	5.0669	6.2635	6.339	6.2115	6.1076	6.037	5.9918	5.9618	5.9405	5.927
6	Yederasiw Mastawesha የደራሲው ማስታወሻ	3.4626	4.4126	5.338	5.4892	5.4365	5.384	5.3516	5.3351	5.327	5.323	5.3211
7	New Testament አዲስ ኪዳን	3.6031	4.9639	6.5288	6.9489	7.002	6.9734	6.9383	6.9123	6.8932	6.8773	6.8671
8	Bible መጽሐፍ ቅዱስ	3.5648	4.9546	6.7085	7.3079	7.4767	7.513	7.4989	7.4771	7.4574	7.4472	7.4402
Average		**3.6044**	**4.832237**	**6.097187**	**6.314875**	**6.290525**	**6.2476**	**6.211075**	**6.185638**	**6.167125**	**6.154713**	**6.146963**

Appendix II: PPMC Compression Rate

Compression Rate (bits/symbol)												
No	Book	Order										
		0	1	2	3	4	5	6	7	8	9	10
1	Tsheay ፀሀይ	4.99	3.58	2.88	2.88	2.92	2.93	2.95	2.96	2.98	2.99	3.0
2	Agere አገሬ	5.86	4.42	3.59	3.5	3.53	3.56	3.59	3.6	3.61	3.61	3.62
3	Tenbit ትንቢት	6.09	4.45	3.47	3.35	3.38	3.41	3.43	3.45	3.46	3.46	3.47
4	Yegezetengw Mastaesha የጋዜጠኛው ማስታወሻ	6.04	4.76	3.94	3.83	3.87	3.9	3.92	3.93	3.94	3.94	3.94
5	Fiker eske Mekabir ፍቅር እስከ መቃብር	5.22	3.95	3.2	3.16	3.23	3.28	3.32	3.34	3.36	3.37	3.38
6	Yederasiw Mastawesha የደራሲው ማስታወሻ	6.05	4.74	3.92	3.81	3.85	3.89	3.91	3.92	3.93	3.93	3.93
7	New Testament አዲስ ኪዳን	5.79	4.2	3.19	3	2.98	2.99	3.01	3.02	3.03	3.03	3.04
8	Bible መጽሐፍ ቅዱስ	5.87	4.22	3.12	2.86	2.8	2.79	2.79	2.8	2.81	2.81	2.81
Average		**5.73875**	**4.29**	**3.41375**	**3.29875**	**3.32**	**3.34375**	**3.365**	**3.3775**	**3.39**	**3.3925**	**3.39875**

Appendix III: PPMC Compression Time

Compression Time (secs)												
No	Book	Order										
		0	1	2	3	4	5	6	7	8	9	10
1	Tsheay ፀሀይ	3.01	1.63	1.46	1.63	1.81	2.05	2.31	2.57	2.82	3.14	3.43
2	Agere አገሬ	7.34	3.91	3.42	3.87	4.48	5.17	6.01	6.72	7.45	8.17	8.91
3	Tenbit ትንቢት	15.31	7.85	6.26	6.92	8.17	9.52	10.91	12.15	13.8	15.54	16.94
4	Yegezetengw Mastaesha የጋዜጠኛው ማስታወሻ	18.08	10.41	8.76	10.28	11.72	13.48	15.13	17.14	18.68	20.86	22.85
5	Fiker eske Mekabir ፍቅር እስከ መቃብር	17.61	10.76	8.34	8.42	9.35	10.9	12.53	14.35	16.42	18.22	19.82
6	Yederasiw Mastawesha የደራሲው ማስታወሻ	23.17	14.2	10.9	12.28	14.43	16.57	19.01	21.47	23.43	27.38	31.06
7	New Testament አዲስ ኪዳን	27.84	13.62	9.71	9.93	11.76	14.03	16.39	18.88	21.76	24.98	30.04
8	Bible መጽሐፍ ቅዱስ	100.89	54.25	34.81	32.75	37.23	45.89	51.53	60.97	69.97	79.17	88.97
Average		**26.6562**	**14.5787**	**10.4575**	**10.76**	**12.3687**	**14.7012**	**16.7275**	**19.2812**	**21.7912**	**24.6825**	**27.7525**

Appendix IV: PPMD Compression Ratio

Compression Ratio											
No	Book	Order									
		0	1	2	3	4	5	6	7	8	9	10
1	Tsheay ፀሀይ	3.8942	5.4385	6.7781	6.7984	6.7356	6.6927	6.6644	6.6351	6.5991	6.5722	6.556
2	Agere አገሬ	3.5645	4.7467	5.8627	6.018	5.9649	5.9135	5.8778	5.856	5.8429	5.8349	5.83
3	Tenbit ትንቢት	3.4556	4.7358	6.1026	6.3248	6.2794	6.2263	6.183	6.1554	6.1385	6.1279	6.1203
4	Yegezetengw Mastaesha የጋዜጠኛው ማስታወሻ	3.4632	4.4011	5.335	5.4827	5.439	5.3956	5.3677	5.3545	5.3476	5.3441	5.3424
5	Fiker eske Mekabir ፍቅር እስከ መቃብር	3.8397	5.0796	6.2939	6.3911	6.2855	6.1855	6.1173	6.0744	6.0451	6.0236	6.01
6	Yederasiw Mastawesha የደራሲው ማስታወሻ	3.4634	4.4244	5.3672	5.5219	5.4731	5.4225	5.39	5.3745	5.3664	5.3621	5.3601
7	New Testament አዲስ ኪዳን	3.6038	4.9726	6.5537	6.9869	7.0507	7.0389	7.0133	6.993	6.9765	6.9618	6.9518
8	Bible መጽሐፍ ቅዱስ	3.565	4.9584	6.728	7.3494	7.5326	7.5889	7.5867	7.5717	7.5557	7.5474	7.5411
Average		**3.60617**	**4.84463**	**6.12765**	**6.35915**	**6.3451**	**6.30798**	**6.27502**	**6.25182**	**6.23397**	**6.22175**	**6.213962**

Appendix V: PPMD Compression Rate

Compression Rate (bits/symbol)												
No	Book	Order										
		0	1	2	3	4	5	6	7	8	9	10
1	Tsheay ፀሀይ	4.98	3.57	2.86	2.85	2.88	2.9	2.91	2.92	2.94	2.95	2.96
2	Agere አገሬ	5.86	4.4	3.56	3.47	3.5	3.53	3.55	3.57	3.57	3.58	3.58
3	Tenbit ትንቢት	6.08	4.44	3.44	3.32	3.35	3.38	3.4	3.42	3.42	3.43	3.43
4	Yegezetengw Mastaesha የጋዜጠኛው ማስታወሻ	6.04	4.75	3.92	3.82	3.85	3.88	3.9	3.91	3.91	3.91	3.92
5	Fiker eske Mekabir ፍቅር እስከ መቃብር	5.22	3.94	3.18	3.13	3.19	3.24	3.27	3.3	3.31	3.33	3.33
6	Yederasiw Mastawesha የደራሲው ማስታወሻ	6.04	4.73	3.9	3.79	3.83	3.86	3.88	3.9	3.9	3.9	3.91
7	New Testament አዲስ ኪዳን	5.79	4.19	3.18	2.99	2.96	2.96	2.97	2.98	2.99	3.0	3.0
8	Bible መጽሐፍ ቅዱስ	5.87	4.22	3.11	2.85	2.78	2.76	2.76	2.76	2.77	2.77	2.77
Average		**5.735**	**4.28**	**3.39375**	**3.2775**	**3.2925**	**3.31375**	**3.33**	**3.345**	**3.35125**	**3.35875**	**3.3625**

Appendix VI: PPMD Compression Time

Compression Time (secs)												
No	Book	Order										
		0	1	2	3	4	5	6	7	8	9	10
1	Tsheay ፀሀይ	3.17	1.82	1.56	1.73	1.94	2.07	2.32	2.68	2.68	3.29	3.34
2	Agere አገሬ	7.28	3.96	3.44	3.83	4.49	5.06	5.79	6.48	6.48	7.51	8.28
3	Tenbit ትንቢት	16.83	8.18	6.52	7.35	8.7	10.22	11.31	12.78	12.78	14.62	15.49
4	Yegezetengw Mastaesha የጋዜጠኛው ማስታወሻ	18.96	11.62	9.41	10.66	12.55	14.37	15.24	17.74	17.74	19.38	21.18
5	Fiker eske Mekabir ፍቅር እስከ መቃብር	18.14	12.28	10.83	9.22	10.08	11.7	14.15	15.96	15.96	17.3	19.94
6	Yederasiw Mastawesha የደራሲው ማስታወሻ	24.73	15.25	12.35	13.79	16.17	18.58	20.91	23.3	23.3	25.6	28.41
7	New Testament አዲስ ኪዳን	29.71	21.13	11.49	11.53	13.64	15.22	18.02	20.53	20.53	23	26.58
8	Bible መጽሐፍ ቅዱስ	102.92	57.66	37.61	34.04	37.31	45.12	51.65	59.72	59.72	70.67	80.12
Average		**27.7175**	**16.4875**	**11.6512**	**11.5187**	**13.11**	**15.2925**	**17.4237**	**19.8987**	**22.6712**	**25.4175**	**28.11**