M188: A New Preprocessor for Better Compression of Text and Transcription Files

Mete Eray Şenergin

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the Degree of

Master of Science
in
Electrical and Electronic Engineering

Eastern Mediterranean University
November 2014
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Electrical and Electronic Engineering.

Prof. Dr. Hasan Demirel
Chair, Department of
Electrical and Electronic Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Electrical and Electronic Engineering.

Assoc. Prof. Dr. Erhan A. İnce
Supervisor

Examining Committee

1. Prof. Dr. Hasan Demirel

2. Prof. Dr. Hüseyin Özkaramanlı

3. Assoc. Prof. Dr. Erhan A. İnce

ABSTRACT

Compression of natural language text files is worthwhile for communities such as Project Gutenberg in terms of their storage space and even for text messaging applications' bandwidth efficiency. Thus, there has been extensive research on preprocessing techniques. The thesis proposes a new word-based preprocessor named METEHAN188 (M188). The proposed method provides better compression of text and transcription files when concatenated with some well known data compression algorithms. M188 and state-of-the-art preprocessors; starNT, WRT, ETDC, SCDC and RPBC are compared while concatenated with PPMD and PPMonstr. M188 differs from the other methods; it has larger dictionary which provides coverage of more words, the disadvantage is that it slows down the process; it has longer alphabet which gives M188 the opportunity of assigning shorter codewords; it does not code space and punctuation characters which speeds up M188 also output a more predictable scheme. During experiments, Wall Street Journal, Calgary, Canterbury, Large, Gutenberg and Pizza & Chili corpora are used. For the files in Calgary corpus the experimental results yield that M188 can overcome all other preprocessing techniques in terms of compression effectiveness. For the files selected from the project Gutenberg and Canterbury corpora WRT+PPMonstr has 1.22% gain in over M188+PPMonstr on the average. The results showed that best two preprocessors for compression effectiveness are M188 and WRT and for timing performance ETDC and SCDC are the fastest preprocessors.

Keywords: LIPT, StarNT, WRT, Universal Preprocessor, PPMonstr, M188, ETDC, SCDC, RPBC, PPM, Data Compression.

ÖZ

Gutenberg projesi gibi toplulukların veri depolama alanlarını ve hatta metin mesajlaşma uygulamalarının bant genişliğini kazanımı için metin sıkıştırma kayda değer bir uygulamadır, araştırmalar önişlemcilerin kayda değer kazanç sağladığını göstermiştir. İş bu tez, metin dosyaları için sıkıştırılma oranını en iyileştirmeye yönelik yeni bir önişlemciyi önermektedir. Bu önişlemciyi Metehan 188 ya da M188 olarak adlandırmış bulunuyorum. M188 ile LIPT, StarNT, WRT, ETDC, SCDC, RPBC önişlemcileri PPMonstr ve PPMD sıkıştırma algoritmalarına önişlem yapacak şekilde kullanılmış daha sonrasında zaman ve sıkıştırma başarımı açısından kıyaslanmıştır. Diğer metotlara göre; M188 daha büyük bir sözlüğe sahiptir bu da kodlama kapsamını genişletmiştir; ayrıca, M188 kodlarını daha uzun bir alfabeden yararlanarak yaratmaktadır, bu sayede daha kısa kodlar atayabilmektedir. Son olarak M188 boşluk ve noktalama işaretlerini kodlamamaktadır bu da zamanlamada kazanç sağlamakta olup sıkıştırma algoritmalarına daha tahmin edilebilir bir yapı sağlamaktadır. Deneylerde; Wall Street Journal, Calgary, Canterbury, Large, Gutenberg ve Pizza & Chili metin derlemelerinden alınan dosyalar kullanılmıştır. Calgary dosyalarında M188 diğer tüm önişlemcilerden daha iyi sıkıştırma sağlamıştır. Gutenberg ve Canterbury dosyalarında ise WRT+PPMonstr ikilisi M188+PPMonstr 'ye göre yüzde 1.22 daha iyi sıkıştırma başarımı sağlamıştır. Sonuç olarak sıkıştırma başarımları en iyi olan iki algoritma M188 ve WRT olarak belirlenmiştir. En hızlı iki algoritma ise ETDC ve SCDC olarak belirlenmiştir.

Anahtar Kelimeler: LIPT, StarNT, WRT, Evrensel Önişlemci, PPMonstr, M188, ETDC, SCDC, RPBC, PPM, Veri Sıkıştırma.

To my beloved wife Meltem.

ACKNOWLEDGEMENT

I would like to thank Assoc. Prof. Dr. Erhan A. İnce for his utterly supportive guidance in the declaration of this study and for his time. Also, I would like to thank my family respectively N. Sibel Şenergin (my mother) for giving me the idea of Radix system in this study, Meltem Şenergin (my wife) for her refutations on my initial design and İ. Giray Şenergin (my dad) for making me believe that the problem is not about the compiler or the computer, it was about the code. Special thanks to Dr. Antonio Farina, Dr. Miguel A. Martinez-Prieto, Dr. Gonzalo Navarro and Dr. Susana Ladra Gonzales for their kind cooperation with us.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Since the beginning of time information sharing has been a need of various societies inhabiting our planet. Among the earliest ways of communication text was appropriated most. Today, there are numerous visual multimedia alternatives however for majority of the people text is still the preferred way of communicating. With the computer era the text is digitized and standardized e.g. ASCII. This has allowed us to optimize the space and time efficiency of this textual information flow. Communication systems which are not memoryless needs to store the data and the actual physical memory needed for storage can be quite costly based on the size of the data. Also quick retrieval of information stored on a far-away server should not take too long. Hence source compression has become an important research area.

The main principle of source (data) compression is to represent the source signal with minimum redundancy such that the number of bytes one needs for storage will be smaller than the size of the original data. Data compression algorithms can be classified in two groups: (i) Lossless and (ii) Lossy compressors.

Lossless data compression guarantees identical reconstruction of the original data (referred to as raw text throughout this thesis) and lossy data compression on the other hand aims to keep the information not the exact data. A well known technique for lossy compression is the SMS language. In SMS language receiving party can

understand the message even the words are not typed properly. Such as; the word 'before' is encoded as 'B4', the word 'your' is encoded as 'ur'. So, there is a loss of the original data but the information can be extracted.

The algorithms which are employed to compress the data are called as data compression algorithms (DCA). The aim of this thesis is to propose a new source coding algorithm that provides gain in compression to the lossless DCAs, when it is used as a frontend processor (preprocessor). The idea behind preprocessing is to change the representation of the data in a form that redundancy is more visible for the DCAs. In order to give details on preprocessors DCAs should be mentioned first. There are numerous lossless DCAs in the literature and all DCAs process the data in blocks; the block can be a bit sequence, byte or a string of characters (word). Huffman DCA uses characters (bytes) as the symbols to be compressed according to the probability distribution of the source symbols. On the other hand, word based DCAs takes the words as the symbols to be processed. So, DCAs have different methodologies among themselves and those methods can be categorized as i) statistical methods and ii) dictionary based methods.

1.1 Statistical Methods

Statistical compression methods are known to employ variable-length codes and are based on a model. The model is used by the compression algorithm to map input data to bit sequences in such a way that probable (frequently encountered) data will produce shorter outputs in comparison to improbable data. The quality of compression is based on the model adopted. Static, semi-static and adaptive models are among the well known models. A static model is a fixed model that is known by both the compressor and the de-compressor and does not depend on the data that is

being compressed. A semi-static model on the other hand is a fixed model that is constructed from the data to be compressed and must be included as part of the compressed data. An adaptive model changes during the compression. At a given point in compression, the model is a function of the previously compressed part of the data. Since that part of the data is available to the de-compressor there is no need to store the model. Huffman coding [1], adaptive Huffman coding [2], arithmetic coding (AC) [3], Prediction by Partial Matching (PPM) [4] and PAQ [5] , Plain Huffman (PH)[6], Tagged Huffman (TH)[6], End-Tagged Dense Codes (ETDC)[7], (s; c)-Dense Coding [8] and Restricted Prefix Byte Coding ([9],[11]), are examples of statistical methods. Processing for statistical two pass techniques are as follows: in the first pass these algorithms gather statistics about the list of source symbols (vocabulary) and construct a model of the text and in the second pass each symbol is substituted by a codeword. It has been stated in [12] that Dense Codes offer some advantages over byte-oriented Huffman encoding based compression methods. Some of their advantages are that they can be build faster, require about the same search time as Tagged Huffman and can achieve better compression rates.

1.1.1 Prediction by Partial Matching Family

Prediction by Partial Matching (PPM) ([4], [28]), is an adaptive statistical data compression technique which uses context modeling and prediction. The context is defined as the finite sequence of symbols preceding the current symbol. The length of the sequence is also known as the order of the context. PPM makes use of these previous symbols in the uncompressed symbol stream to predict the next symbol in the stream. [4], was then developed into PPMC [28] by Alistair Moffat. PPMC [28], is a hybrid combination of Methods A and B described in [4]. Performance of these compression methods is based on the escape probabilities (the probability of new

symbols occurring in the context). There are many versions of the PPM since the calculation of the escape probabilities is done in an ad-hoc manner. PPMD+ [29], PPMd [30], PPM* [31] and Monstereous PPMII.J (PPMonstr) [32] are some other variants of the prediction by partial matching algorithm. Compressors like Durilca and Durilca Light [33] are based on Shakarin's PPMd [30] and PPMonstr [32]. mPPM described in [34], is a two stage compressor. The first stage maps words into two byte codewords using a limited length dictionary, and in the second stage conventional PPM is used to encode codewords or new words. DMC [25], is a lossless compression algorithm developed by Cormack and Horspool [25]. It uses predictive arithmetic coding, similar to PPM, except that the input is predicted one bit at a time rather than one byte at a time.

1.2 Dictionary Based Methods

Dictionary based DCAs gets the strings as their symbols. The dictionary can be static or dynamic. For static dictionaries, the dictionary is generated with the help of training corpora which is better if gigantic in size and each word in the corpora takes place in the dictionary only at once, those words can be ordered by their frequency, their length or lexicographically. Then the codeword assignment is done according to the dictionary. Each word has its unique codeword according to its position in the dictionary. Static dictionary has a disadvantage of optimum codeword assignment. Since, the probability distribution of the source symbols may not be related with the probability distribution of the words in the training corpora. For optimum codeword assignment the dictionary can be compiled from the source (data) itself and this is called as dynamic dictionary. However, a dynamic dictionary must be a part of the encoded data. Thus overhead is the disadvantage of using dynamic dictionary. Dynamic dictionary based methods are pretty effective for files containing small

variety of words. Examples for dictionary based methods include LZ77, LZ78, LZW ([13],[14]) and DEFLATE [15]. Length Index Preserving Transformation (LIPT) ([17]-[18]), Star New Transform (StarNT) [19], Word Replacement Transformation (WRT) [21] and Improved Word Replacement Transform (IWRT) [22] are examples of preprocessing techniques that make use of a static dictionary.

1.2.1 Lempel Ziv Family

LZ77 discussed in [13] was introduced during 1977 by Abraham Lempel and Jakob Ziv. It is based on a rule for parsing strings of symbols from a finite alphabet into sub-strings that are shorter in length. Lempel Ziv Welch (LZW) is a variation on the LZ77 due to the introduction of a dictionary and variable-rate coding. LZW was widely used till after 1986. Afterwards, the more efficient DEFLATE algorithm replaced it. DEFLATE algorithm which combines LZ77 and a Huffman coder was first proposed by Phil Katz.

1.3 Others

Other data compression algorithms that do not directly classify in the former two groups include run-length encoders (RLE) [23], Burrows-Wheeler transformation [24], Dynamic Markov Compression (DMC) [25] and Bzip2 [26].

1.3.1 Bzip2

The Bzip2 compressor by Julian Seward [26], is based on the Burrows-Wheeler Transform (BWT). Previous work [27], has reported that when an input file is transformed by BWT the output file would be slightly larger in size than the source. However, it has also been shown that the BWT would sort the file in such a way that the output would have many redundant bytes and become highly suitable for effective compression. Bzip2 compressor would apply four different transformations back to back. These are BWT, a global structure transformation (GST), run length

encoding (RLE) and the entropy coding (EC) stages. A typical representative of the GST is the Move-to-Front (MTF) transformation and for EC Huffman or Arithmetic coding can be employed.

1.3.2 PAQ Project

The PAQ Project ([50],[52]) is an open source project which gave numerous versions from numerous contributors and it is quite successful on many benchmarks. The reason, PAQ is not classified under statistical methods or not under dictionary based methods is because PAQ has both properties in some versions. It uses context mixing and has similarities with PPM. As PPM PAQ also has predictor part with an arithmetic coder as the main mechanism. But the difference is about mixing the contexts, which is about allowing contexts to be arbitrary functions of the history [49]. The model used is context mixing model. In this thesis the latest version PAQ8l which is developed by M. Mahoney in 2007 is used to compress M188's EOL flags since PAQ8l has the best compression rates on many benchmarks.

The organization of the thesis is as follows: Chapter 2 provides a brief summary of some well-known preprocessing techniques, namely LIPT, StarNT, WRT, ETDC, SCDC and RPBC. Chapter 3 introduces the encoding and decoding processes for M188 in details with a detailed example. Chapter 4 summarizes experimental results obtained by using text files from four corpora Calgary [35], Gutenberg [42], Canterbury [43], Large [51], Pizza and Chili [44] and the Wall Street Journal (WSJ) archive obtained from TREC-project [45]. The experiments can be divided into four sets as follows; set one is Calgary files; set two is files from Gutenberg, Large and Canterbury; set three contains comparatively big files depicted from Gutenberg, Pizza Chili and the Wall Street Journal and the fourth set is timing set which contains files from Calgary corpus details will be provided in Chapter 4. Firstly, the

compression achieved by M188 in stand-alone mode is compared against some well known compressors (AC, LZW , Gzip, 7z, Repair coupled with a minimum redundancy Huffman coder, Bzip2, PPMD, PPMonstr, PAQ8). Secondly, M188 and other preprocessors are used prior to PPMD and PPMonstr and bpc values for files selected from Calgary, Gutenberg, and Canterbury corpora are provided. Thirdly, M188 and WRT are compared with word-based byte-oriented preprocessors such as ETDC, SCDC and RPBC. Source files used were selected from Gutenberg corpora, Pizza and Chili corpora and the Wall Street Journal archive. Chapter 4 also provides comparative bar graphs for the time complexity of the M188 encoder/decoder pair and other pre and post-processors. Finally, Chapter 5 delivers a discussion and concludes the thesis.

Chapter 2

PREPROCESSING TECHNIQUES

A preprocessing algorithm tries to exploit different properties of textual data by applying a reversible transformation to the source before it is passed on to a standard DCA. The main aim is to make the redundancy more visible to the post-compressor so that the overall compression rate can be improved. Preprocessing techniques using a static dictionary would replace words in a given text file by a character encoding that represents a pointer to encoded word in the dictionary. Semi-static techniques on the other hand do not assume any data distribution and learn it during a first pass in which the model is built. After the creation of the model, text can be encoded by replacing each symbol with a fixed codeword assigned in accordance with the model. The sub-sections below summarize details of some well-known preprocessing algorithms. Namely: LIPT, StarNt, WRT, ETDC, SCDC and RPBC.

2.1 Preprocessors Derived from the Star-Transform

Star Transform [16], has been proposed by M. R. Nelson in 2002. The main idea behind this transformation is to define a unique signature for each word by replacing the letters of the word by a special character (*) and to use a minimum number of characters to identify each specified word. Subsections 1-3 below are examples of algorithms that have been derived from the basic star-transform.

2.1.1 Length Index Preserving Transformation (LIPT)

Word based preprocessing techniques are known to make use of an English language dictionary. The dictionary is needed for two reasons: firstly it is used to replace

frequently occurring words by corresponding character encoding, secondly it is used at the receiver for decoding the codeword in the compressed file. Given a compiled dictionary, the LIPT algorithm [17], would first create many disjoint dictionaries based on word lengths. All words of length *i* would be placed in dictionary D_i and then sorted according to the frequency of the word in the corpus being compressed. The algorithm will then carry out mapping to encode words in each disjoint dictionary D_i. A word in position *k* in dictionary D_i is denoted as $D_i[k]$. Based on *k* value the encoded word can be written as $*c_{len}$, $*c_{len}[c]$, $*c_{len}[c][c]$ *or* $*c_{len}[c][c][c]$ where c_{len} denotes a character from the alphabet [a-z, A-Z] and *c* cycles through [a-z, A-Z]. If $k = 0$, the encoding is c_{len}. For $k > 0$, encoding can assume three different forms based on the range of values *k* can assume as in formula (1) below.

$$\begin{array}{cc} 1 < k < 52 & *c_{len}\ c \\ 53 < k < 2756 & *c_{len}\ c\ c \\ 2756 < k < 140608 & *c_{len}\ c\ c\ c \end{array} \quad (1)$$

For example, when LIPT is encoding the 4th word of length 6 in dictionary *D*, the codeword will be *fd. For decoding, LIPT uses the length block indicator that comes after the '*' symbol to locate the length block in dictionary D. The characters that come after the length block indicator are used to compute an offset from the beginning of the length block previously chosen. The word at this location in the original dictionary would be the decoded word.

2.1.2 Star New Transformation (StarNT)

Realizing that more than 82% of the words in the English texts had lengths which are greater than three characters Mukherjee, Sun and Zhang concluded that if they re-

code each English word with a representation that is less than three symbols, a certain pre-compression could be achieved. This was the starting point before they proposed a new star transformation called StarNT [19]. This transform differs from the earlier versions of star family of transforms [20] with respect to the usage of '*'. In earlier transformations the '*' denoted the beginning of a codeword but in starNT it implies that the following word does not exist in the dictionary. This change was adopted in order to minimize the encoding/decoding time of the backend compressor. '~' appended to the transformed word implies that the first letter of the word is capital and when ' ‘ ' is appended this would mean that all the letters of the word are capital. For encoding the starNT uses a dictionary where the first 312 words (the most frequently occurring words in English) appear at the top in decreasing order of their frequencies and the remaining words are sorted according to their lengths. For encoding letters [a ... z, A ... Z] are used. The first 26 words in dictionary are assigned 'a', 'b', ..., 'z' as their code words. The next 26 words are assigned 'A', 'B', ..., 'Z'. The 53rd word is assigned 'aa' and 54th 'ab' etc. Using this approach the transform dictionary can support a total of 143,364 entries.

2.1.3 Word Replacement Transformation (WRT)

The word replacement transform (WRT) ([21], [37]) has been proposed by Grabowski and is a variation of the starNT with some improvements like capital conversion, word ordering in the dictionary, q-gram replacement and end of line (EOL) coding. The idea behind WRT is the following. If a word from the source file exists in the static dictionary then since the codewords are shorter than words the encoded file would be smaller than the source itself. The position of the word in the external dictionary determines which codeword to use while encoding. Since WRT is the method M188 is competing it is necessary to give further details about its

process. Firstly, the capital conversion is a well known technique for preprocessing and it is quite obvious from its name capital conversion (CC). CC is converting uppercase letters in a word into lowercase letters with adding a one-byte flag *f* for decoding part to know about this conversion. Actually, there are at least two different one-byte flags, f_1 is for first-upper words and f_2 is for all-upper words. Hence, there is no need of flag for all-lower words. StarNT has capital conversion but in WRT it is improved as follows; the word 'Capital' is a word first-upper case so, it is converted into to 'capitalf_1' in StarNT. Then, Skibinski realized that, when the flag is appended in front of the word instead of end of the word as '$f_1$capital' gives better results on context modeling DCAs by providing longer contexts. Even better results can be obtained when a space is added between the word and the flag [21]. M188 also uses the CC method as 'f_1_capital' with a space between the flag and the word. Secondly, WRT uses three sub alphabets of lengths 43, 43 and 42 respectively so, it can store up to 79,550 words. Thirdly, *q*-gram replacement which is another widely used technique that WRT has adopted. It is based on partial encoding of unknown words. For example, if the word 'whatchamacallit' is encountered and it is not existing inside WRT's dictionary. Then; up to four letters which means q=4, WRT can encode any substring which exists the dictionary eg. 'what' is a substring of the word 'whatchamacallit' so, it can be encoded as '$chamacallit'. Which '$' is the codeword of the word 'what'. WRT also has an improved dictionary. The dictionary of StarNT is also replaced with Aspell's English dictionary level 65 and the ordering is no just based on the frequency as in StarNT. The dictionary is first sorted according to the frequency which is measured with the help of a 3GB size training corpora taken from the Gutenberg Project then, it is sorted in small groups in lexicographical order of suffixes. The last method is end of line coding (EOL) which is replacing the end of

line characters with space characters. The end of line characters can be thought as artificial and by replacing them with space characters DCAs can process larger blocks. WRT at this point chose to replace end of line characters only which are surrounded by lowercase letters, those end of line characters are replaced by space characters. In order decoding part to recover those end of lines, there are binary flags are written and compressed with an arithmetic coder.

2.2 Semi-Static Word Based Byte Oriented Preprocessors

Semi-static word-based byte-oriented preprocessors are known to deliver compression ratios of 30-35 %. Using bytes instead of bits may slightly worsen the compression ratio however both the encoding and decoding processes will speed up. Byte-oriented preprocessors also provide the flexibility to carry out direct pattern search on the compressed text since they are self-synchronized codes. Subsections below provide details about the End-Tagged Dense Coding, (s; c)-Dense Coding and Restricted Prefix Byte Coding (RPBC) techniques.

2.2.1 End Tagged Dense Coding (ETDC)

End-Tagged Dense Coding (ETDC) [7] is a word-based byte-oriented compression method. To compute the codeword of each source word, ETDC uses a semi-static model that is simply the vocabulary (list of source symbols) ordered by frequency. One byte codewords are given to the first 128 words in the vocabulary. Words in positions 128 to $128 + 128^2 - 1$ are sequentially assigned two-byte codewords and the three byte codewords are given to the remaining words. ETDC has been inspired from the Tagged Huffman code [10], and has been obtained through a very simple change. Rather than marking the beginning of each codeword the most important bit of every byte has been used to mark their end. Hence whenever a given byte is the last byte of a codeword the highest bit is set to 1 otherwise it must be set to 0. In

ETDC the flag bit is enough to ensure that the code is a prefix code regardless of the contents of the other 7 bits. Therefore there is no need to use Huffman coding over the remaining 7 bits.

2.2.2 (*s, c*) - Dense Coding (SCDC)

(s, c)-Dense Coding [8] is a more sophisticated variant of word-based byte-oriented text compressors. End-Tagged Dense Codes use 128 target symbols for the bytes that do not end a codeword (continuers), and the other 128 target symbols for the last byte of the codeword (stoppers). An (*s, c*)-Dense Code on the other hand adapts the number of stoppers and continuers to the word frequency distribution of the text, so that *s* values are used as stoppers and $c = 256 - s$ values as continuers. SCDC assigns the one-byte codewords from *0* to *s-1* to the first s words of the vocabulary. Words in positions *s* to $s + sc - 1$ are sequentially given two-byte codewords. Three-byte codewords are for words from $s+sc$ to $s + sc + sc^2 - 1$ The encoding and decoding algorithms are the same as those of ETDC. One only needs to change the 128 value of stoppers and continuers by *s* and *c* respectively.

2.2.3 Restricted Prefix Byte Coding (RPBC)

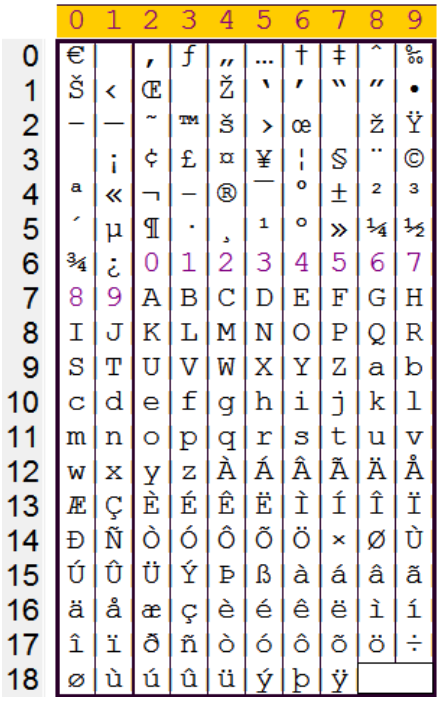
Restricted Prefix Byte Coding (RPBC) technique was first proposed in [9]. Unlike the (*s, c*) - dense codes which use an infinite tuple of numbers, the RPBC uses a finite tuple where the numbers in the tuple refer to the initial digit ranges in the radix-R code. It can be said that the code is restricted since $v1 + v2 + v3 + v4 \leq R$. Under RPBC the first byte of each codeword is used to describe the length of the codeword and additional bytes use the remaining code space. While using RPBC codeword lengths are not as variable as in an unrestricted radix-256 Huffman code, however the loss in compression effectiveness compared to a Huffman code is less. For encoding with a 4-tuple (v_1, v_2, v_3, v_4) the code has *v1* one-byte codewords, Rv_2

two-byte codewords, R^2v_3 three-byte codewords and R^3v_4 four-byte codes. It is required that $v_1 + Rv_2 + R^2v_3 + R^3v_4 \leq n$, where n represents the cardinality of the source alphabet.

Chapter 3

THE PROPOSED PREPROCESSOR: M188

This section provides details about the proposed preprocessor, M188. This new preprocessor uses 1-3 bytes long codewords while encoding text documents. The codewords are composed of characters drawn on the basis of a radix-188 numbering system from a 188 characters long alphabet which has been provided in Fig. 1.



	0	1	2	3	4	5	6	7	8	9
0	€		‚	ƒ	„	…	†	‡	ˆ	‰
1	Š	‹	Œ		Ž	‘	’	“	”	•
2	–	—	˜	™	š	›	œ		ž	Ÿ
3		¡	¢	£	¤	¥	¦	§	¨	©
4	ª	«	¬	-	®	¯	°	±	²	³
5	´	µ	¶	·	¸	¹	º	»	¼	½
6	¾	¿	0	1	2	3	4	5	6	7
7	8	9	A	B	C	D	E	F	G	H
8	I	J	K	L	M	N	O	P	Q	R
9	S	T	U	V	W	X	Y	Z	a	b
10	c	d	e	f	g	h	i	j	k	l
11	m	n	o	p	q	r	s	t	u	v
12	w	x	y	z	À	Á	Â	Ã	Ä	Å
13	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
14	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù
15	Ú	Û	Ü	Ý	Þ	ß	à	á	â	ã
16	ä	å	æ	ç	è	é	ê	ë	ì	í
17	î	ï	ð	ñ	ò	ó	ô	õ	ö	÷
18	ø	ù	ú	û	ü	ý	þ	ÿ		

Figure 1: The alphabet of M188

The value 188 was obtained as follows: Realizing that most of the space and punctuation characters are each 1-byte and the smallest codeword length is also 1-byte for M188, it is quite rational to leave the space and punctuation characters as they are in the encoding process. From the 256 characters extended ASCII set, this

would leave only 191 which classify otherwise. Anticipating that some words that is needed to encode may not be in the dictionary; the 127th ASCII character was reserved for encoding of such words. Also, for the capital conversion process there are two more flags are reserved, which are 143rd and 144th ASCII characters for flagging the first-upper words and all-upper words respectively. Which all CC flags are chosen from unseen characters so those will be denoted as unknown word flag f_{uw}, first-uppercase flag f_{fu} and all-uppercase flag f_{au} throughout the text, hence a total of 188 characters would remain. With three bytes and the extended character set, it is possible to represent up to 6,644,671 different words. M188 dictionary (M188DICT) contains 168,797 words and is 1.49MB in size. M188DICT has been compiled by using; Webster's Unabridged dictionary, some text files from the Project Gutenberg, a name dictionary, various computer transcriptions and various internet sources. The sources used in the compiling of this dictionary sums up to 46.24MB. The dictionary has been created as follows: The compiled text file is scanned sequentially all uppercase characters are converted into lowercase and words are ordered based on their occurrence frequencies. The dictionary is then created by sorting the frequencies in descending order and writing one copy of each word in a text file at the position dictated by this ordering. Besides the regular words, M188DICT also contains some characters or short abbreviations. These characters and abbreviations come about due to the use of various computer transcription files while compiling the dictionary. It can be thought a dictionary larger than 1.49MB is possible nevertheless, locating the position of a particular word in a larger dictionary would require more time such an action would lead to a slowdown in the encoding process. The basic order of processing for M188 preprocessor can be summarized as follows: (i) Encoding represents punctuation marks and separators as they are,

applies capital conversion with flags f_{fu}, f_{au} (ii) Unknown words are escaped with f_{uw}, and represented in plain form. (iii) Words in the dictionary are given a codeword depending on their position in M188DICT using a radix-188 number. M188 encoder uses one byte for the encoding of the first 188 words, the following 188^2-188=35,156 words are presented by two bytes and 3 bytes are used for what remains.

Though not implemented in this study, it is possible to re-design M188DICT to include words from other languages so that it will be capable to encode non-English text files. However, expanding the dictionary this way would mean slower encoding speed.

3.1 M188 Encoder

The encoder for the M188 preprocessor does not replace the space and punctuation characters by codewords and would only administer word encoding. Justification for this is that; the smallest codeword M188 would assign is 1-byte, space and punctuation characters also require 1-byte and for PPM family those are easy to predict. M188 encoding process should be discoursed in details. Capital conversion, end of line coding, search methodology, unknown words and of course the radix-188 numbering system should be mentioned. Therefore, the following subsections would give details.

3.1.1 Capital Conversion

The encoder for M188 starts by scanning the input file character wise and every time a word is encountered (which means after some alphanumerical character(s), a non alphanumerical character is encountered), first thing is categorization of the word's uppercase class. If the word contains uppercase letter(s) those are taken as lowercase letters into the string. If this uppercase letter is only at word's zero index flag for

first-uppercase words f_{fu} plus a space is written into the encoded file instantly. Else, if the word is made up of all-uppercase characters then flag for all- uppercase f_{au} plus a space is written into the encoded file instantly. When the space between the flag and the unknown word is put it gave even better results as stated by Skibinski [21]. The only other case is the word containing only lowercase letters which mostly likely occurs, directly passes through the search phase. In flagged cases the word passes to the search phase in all-lowercased form, right after the flag specified is put.

3.1.2 End of Line Coding

Although end of line coding (EOL) is an optional function of M188 it provides better results on files which have significant space distribution on its lines. M188's EOL technique differs from WRT's in such manner; WRT replaces end of line characters which are surrounded by lowercase and uses binary flags then compresses those flags with an arithmetic coder; M188 replaces (with space character) end of line characters which has 0, 8, 9, 10, 11, 12, 14 and 15 space characters at the same line also flags are ASCII characters; space (32), tab (9) and most frequent 5 letters in English 'e', 't', 'a', 'o', 'i' [48] and compressed with PAQ8l [52]. This technique is based on the analysis of the probability distribution function of number of space characters on a line (see Figure 2) for the Corpora used. When all the end of line characters were replaced by space characters the DCAs gives their best performance (excluding flags file) so, the strategy should be replacing as many as end of line characters but with the optimum size of flag overhead. Then it can be seen from the Figure 2 that those values 0,8-15, covers the majority of the end of lines signed with 8 predictable flags on a separate stream.

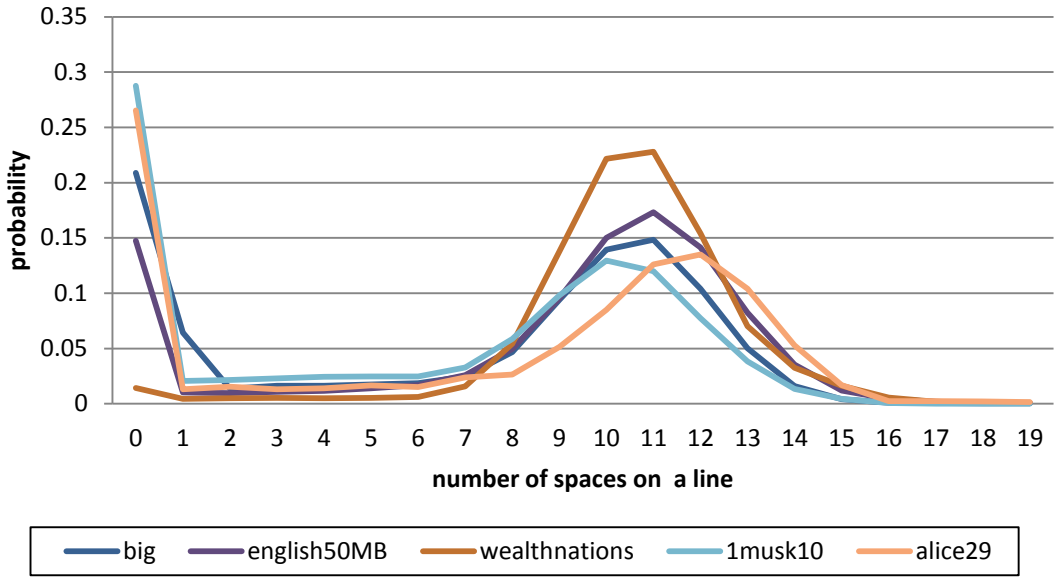


Figure 2: Probability distribution function of number of space characters on a line

The files in the figure above gave better results with EOL coding since, most of their end of line characters are covered. However, as it was stated before some files which have less significant distribution of number of spaces on a line did not gave better results. So, EOL coding is left as optional in M188.

3.1.3 Search Methodology

The searching methodology is designed for M188DICT. It is build up from about half million lines of code, it narrows the searching zone by three phases. After the word's case categorization is made or the case flag is written. The string which is in all-lowercase form passes to a switch which selects the word's length hence, there left only same length words in the scope and this is the first phase the search. After eliminating all other words which are shorter or longer than the word we are searching, the word's first letter passes to another switch; then the case fit is found as the second phase. End of second phase there are words which are in the same length and starts with the same letter in the scope. On the third phase, the word's last letter pass to the third switch then case is found thus the scope is reduced to the list of

words; which have the same length, same first letter and the same last letter with the word is searched. After this phase the search process continues with the linear search of a short list. It can be thought; ternary search, binary search or any well-known search method could be applied. However, M188DICT is not ordered alphabetically or lexicographically it is not the best condition for those well-known search methods, the search method designated has the best performance in comparison with ternary search and binary search.

3.1.4 Unknown Words

The search result can end in two different states; first state is 'the word is found in the dictionary' or second state is 'the word is an unknown word'. If the state is unknown word then unknown word flag f_{uw} is put in the encoded file instantly. Then the word is written in all-lowercase form since the case flag was put before. The flag f_{uw} M188 uses is the 127th ASCII character (DEL). So, the other state should be well explained, if the word is found in M188DICT. Next subsection gives detail of this state.

3.1.5 Radix-188 Numbering

If the word search is successful then its position in M188DICT is known so, the codeword can be written into the encoded file. Radix-188 numbering system is simply represents the position found which is an integer number between [1, 168797] in one to three byte long strings which are made up of the character drawn from the M188's alphabet as in Figure 1. The arithmetic is very simple, if the value is in the range 1-187 then one character of the ordered extended alphabet (see Figure 1) that corresponds to this word’s position in the dictionary. For example, the word 'was' which happens to be in the 14th position in the dictionary will be encoded as ' Ž '. If the word's position is in the range 188-35,343 then two characters from the alphabet

will be used to encode the particular word. For example, 'world' which is at the 459th position will be encoded as 'L,' which 'L' is at the position 83 (see Figure 1) and ',' is at position 2 ($83*188^0 + 2*188^1$). Similarly if the position is between 35,344 and 6,644,671 then three characters would be the codeword. The arithmetic is given in the formula (2).

$$i_1 := p_{dict} mod 188 \xrightarrow{then} c_1 := a_{i_1}$$

$$i_2 := \frac{(p_{dict} - i_1)}{188} mod 188 \xrightarrow{then} c_2 := a_{i_2}$$

$$i_3 := \frac{(p_{dict} - 188 i_2 - i_1)}{188^2} mod 188 \xrightarrow{then} c_3 := a_{i_3} \quad (2)$$

$$\mathbb{C}(p_{dict}) = \begin{cases} c_1, & 1 \le p_{dict} < 188 \\ c_1 c_2, & 188 \le p_{dict} < 188^2 \\ c_1 c_2 c_3, & 188^2 \le p_{dict} < 188^3 \end{cases}$$

Where $i_{1,2,3}$ is the code letters' index in the alphabet a and c is the code letter and $\mathbb{C}$ is the codeword which is a function of p_{dict} position found in the dictionary.

3.2 M188 Decoder

M188 decoding process is robust and it is actually nothing but a simple table look-up there is no searching of any kind of data in this process. Data again read byte wise sequentially. If there is no flag of capital conversion (f_{au} or f_{fu}) is not read then the codeword is converted to the line number of the word by using the formula (4).

$$p_{dict} := i_1 \times 188^0 + i_2 \times 188^1 + i_3 \times 188^2 \quad (3)$$

Simply by putting the line number as an index to the data structure, the word is captured. If there were CC flags read necessary modifications are done according to capital conversion flags after word is captured. Then the captured word is written into the decoded file. Note that, spaces after CC flags are ignored. If f_{uw} is read the characters read are put as they are (if no capital conversion is necessary) until a non alphanumerical character is read. If there is no flag read, simply the codeword is get,

line number is calculated from the formula (3) if the character is a space or a punctuation (space after f_{au} and f_{fu} is excluded) put into the decoded file as it is. M188 decoding process does not require computationally complex operations such as encoder's nested switches of order 3. Hence, there is not much work done on M188 decoder for the sake of timing performance enhancement. Timing performance is enhanced only with embedded dictionary into the executable consequently there is no need to read/load the dictionary in the decoding process.

3.3 M188 Demonstration by Encoding and Decoding a Quote

The quote is one of the famous Albert Einstein quotes which is '*Once you stop learning, you start dying. - Albert Einstein.*'. This quote contains 3 first-uppercase words 6 all-lowercase words, 4 punctuation characters, 9 space characters and 1 unknown word. M188 encoder reads the quote in the Raw_Text file character by character so, in this case the first input read is 'O' which stimulates the flags f_{au} and f_{fu} thus 'o' is copied into the search string. Then, the second input read is 'n' copied into the search string therefore CC flag is determined as the first-uppercase flag f_{fu} and it is instantly written (with a space appended to its end) into the Encoded_Text file. Reading process continues with 'c', 'e'. Afterwards, a space character is encountered so, end of string character is put into the search string (which means a word is captured) and this string passes to the search switches containing the word 'once', this word is found in the M188DICT at 375th position as showed in Figure 3.

```
wordstruct[375].myWord="once";
wordstruct[20].myWord="you";
wordstruct[593].myWord="stop";
wordstruct[1909].myWord="learning";
wordstruct[822].myWord="start";
wordstruct[5196].myWord="dying";
wordstruct[16435].myWord="albert";
```

Figure 3: Dictionary line numbers of the words in the quote

Then its codeword as in the Figure 4 is written into the Encoded_Text file. The space character which has ended the string is put into the Encoded_Text file as is.

```
wordstructoe4[0].myWord ="once";wordstructoe4[0].code="ÿ ";
wordstructyu3[0].myWord ="you";wordstructyu3[0].code="–";
wordstructsp4[0].myWord ="stop";wordstructsp4[0].code="Ÿƒ";
wordstructlg8[0].myWord ="learning";wordstructlg8[0].code="ŸŠ";
wordstructst5[1].myWord ="start";wordstructst5[1].code="8„";
wordstructdg5[2].myWord ="dying";wordstructdg5[2].code="w ";
wordstructat6[20].myWord ="albert";wordstructat6[20].code="HP";
```

Figure 4: Codewords of the words in the quote

The process is completely same until the word 'einstein' is captured. This word is not existing inside M188DICT so f_{uw} is put then the word is printed into Encoded_Text file as is. In Figure 5 original screenshots of the Raw_Text, Encoded_Text, Decoded_Text files are presented. Decoding process can be traced with the help of figures provided.

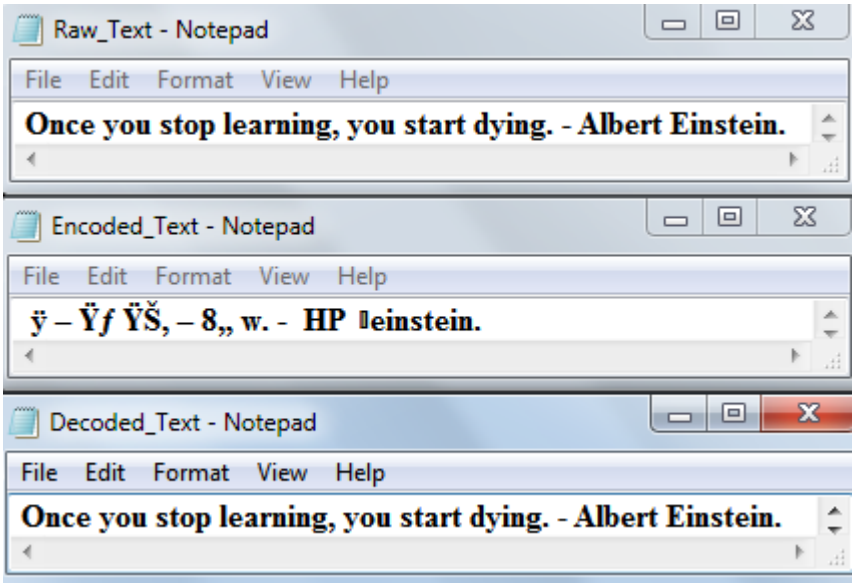


Figure 5: The quote, its M188 encoding and decoding

In order to expose the unseen flag character Figure 6 is created which the red characters are the flags and the space after the CC flags are denoted by '_' character in red.

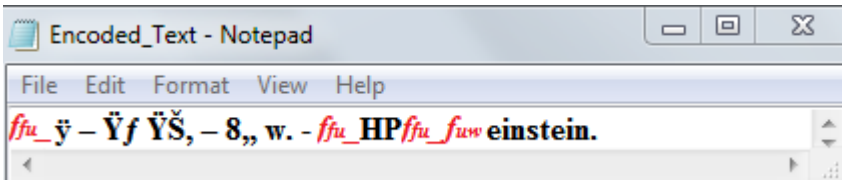


Figure 6: The quote, its M188 encoding and decoding with exposed flags

Chapter 4

PERFORMANCE EVALUATION

In this chapter the focal point is to compare the compression effectiveness provided by preprocessors to the well established DCAs in terms of bit per character values (bpc). Also, timing performance of those algorithms are compared. Bpc value yields the compression ratio such as, a non-compressed extended ASCII character has bpc of 8 since each character is has a 1 byte ASCII value. In a compressed file which has non-compressed size of 100 bytes and compressed file size is 20 bytes then the bpc value of the compressed file is 1.6 referring to formula (4).

$$\frac{Compressed\ File\ Size\ (bytes)}{Non{-}Compressed\ File\ Size(Bytes)} \times 8 \qquad (4)$$

This chapter presents all the experiments carried out; the tools in those experiments can be classified into three groups as; the corpora, preprocessing algorithms and data compression algorithms. All three groups' elements and their references are provided in Table 1. The corpora used in four different combinations which are called as the experiment sets defined in Table 2; the sets 1, 2 and 3 are categorized according to the corpus they belong to and their size. The set 1 is covers files from the Calgary corpus. In set two there are larger files and their source is other than the Calgary corpus. Third set which is the last set for compression experiments is derived from gradually larger files up to 200 MB. The timing set is selected from the Calgary corpus containing more files than set 1. All details of experiment set are provided in Table 2.

Table 1**:** Corpora and source codes used in experiments

Data Compression Algorithms (DCAs)	
Gzip	http://www.Gzip.org
p7zip	http://www.7-zip.org
re-pair	http://www.cbrc.jp/ rwan/software/restore.html
mrhc	http://ww2.cs.mu.oz.au/ alistair/mr coder/shuff-1.1.tar.gz
Bzip2	Bzip2 under 7zip
PPMD	http://compression.ru/ds/
PPMonstr	http://compression.ru/ds/
PAQ8	http://mattmahoney.net/dc/PAQ8l.zip
mPPM	http://www.infor.uva.es/ jadiego/download.php
Corpora	
Calgary	http://corpus.canterbury.ac.nz/descriptions/
Gutenberg	http://www.promo.net/pg/
Canterbury	http://corpus.canterbury.ac.nz/descriptions/
Pizza and Chili	http://pizzachili.dcc.uchile.d/texts/nlang/
Large	http://corpus.canterbury.ac.nz/descriptions/#large
Preprocessing Algorithms	
StarNT	https://code.google.com/p/starnt/source/
M188	http://students.emu.edu.tr/071384/M188_source.zip
WRT4.6	http://pskibinski.pl/research/WRT/WRT46.zip
ETDC	http://vios.dc.fi.udc.es/codes/files/ETDC.tar.gz
SCDC	http://vios.dc.fi.udc.es/codes/files/SCDC.tar.gz

Table 2: File, size, corpora and sets for experiments

File	Size (bytes)	Corpus	Experiment set: (1,2,3,Timing)
big	6,617,121	Gutenberg	3
dickens	31,457,485†	Gutenberg	3
english200MB	213,802,643†	Pizza and Chili	3
english50MB	53,436,448†	Pizza and Chili	3
warpeace	4,434,670	Gutenberg	3
wealthnations	2,227,424	Gutenberg	3
wsj100	100,037,639	The Wall Street Journal	3
1musk10	1,349,139	Gutenberg	2
alice29	152,089	Canterbury	2
anne11	587,051	Gutenberg	2
asyoulik	125,179	Canterbury	2
bible	4,047,392	Large	2
lcet10	426,754	Canterbury	2
bib	111,261	Calgary	1 & Timing
book1	768,771†	Calgary	1 & Timing
book2	610,856	Calgary	1 & Timing
news	377,109	Calgary	1 & Timing
paper1	53,161	Calgary	1 & Timing
paper2	82,199	Calgary	1 & Timing
progc	39,611	Calgary	1 & Timing
progl	71,646	Calgary	1 & Timing
progp	49,379	Calgary	1 & Timing
paper3	46,526	Calgary	Timing
paper4	13,286	Calgary	Timing
paper5	11,954	Calgary	Timing
paper6	38,105	Calgary	Timing
trans	93,695	Calgary	Timing

†Sizes may differ from the source since, some ASCII control characters are removed.

Table 3: Distribution of character types in the sample files

File	Punctuations (%)	Spaces (%)	Remaining (%)
Bib	9.70	17.99	72.31
book1	4.51	18.49	76.99
book2	6.11	16.93	76.97
news	9.83	17.61	72.56
paper1	8.82	16.65	74.53
paper2	4.31	16.92	78.77
progc	16.11	24.37	59.51
progl	20.28	23.59	56.13
progp	14.73	28.31	56.96
1musk10	4.67	18.68	76.65
alice29	5.59	21.89	72.51
anne11	4.00	19.39	76.61
asyoulik	4.01	21.07	74.92
bible	3.02	19.68	77.30
dickens	4.26	18.84	76.90
lcet10	4.28	17.83	77.89

The Table 3 gives percentages of different character types in some of the files used, thus one can observe that spaces and punctuation characters are having approximately 25% of the data. This table can be used in justification of the preprocessing algorithms' performance comparison reminding that M188 does not encode those characters and gives the best results in the files which has more space and punctuation characters, details on the results are given in next the sections.

4.1 Stand-alone Compression Effectiveness of M188

The experiment has studied the stand-alone compression effectiveness of the proposed M188 preprocessor and experiment set 1 is used. Comparisons are made between Huffman coder, Arithmetic coder, LZW, Gzip, 7z, Bzip2, PPMD-o4, PPMonstr, PAQ8, Repair and M188; Table 4 provides the bpc results and for visual easiness of evaluation Figure 7 provides the bar graph of those results.

Table 4: Stand alone compression effectiveness of M188 vs DCAs on set 1

File	Huffman bpc [40]	Artihmetic bpc [40]	M188 bpc	LZW bpc [40]	Gzip -9 bpc	7z bpc	Repair + mhrc bpc	Bzip2 bpc	PPMD -o4 bpc	PPMonstr bpc	PAQ8 -8 bpc
bib	5.31	5.23	6.41	3.87	2.51	2.20	2.27	1.97	1.90	1.64	1.50
book1	4.57	4.55	4.25	4.07	3.25	2.72	2.70	2.42	2.30	2.12	2.00
book2	4.84	4.78	4.13	4.54	2.70	2.22	2.33	2.06	2.01	1.72	1.59
news	5.25	5.19	4.96	4.94	3.06	2.52	2.70	2.52	2.41	2.06	1.90
paper1	5.17	4.98	4.37	4.69	2.79	2.61	2.75	2.49	2.34	2.10	1.97
paper2	4.73	4.63	3.93	4.05	2.89	2.66	2.68	2.44	2.31	2.10	1.99
progc	5.44	5.11	5.39	4.94	2.68	2.55	2.78	2.53	2.39	2.07	1.92
progl	4.91	4.76	5.52	3.96	1.80	1.68	1.94	1.74	1.73	1.32	1.19
progp	5.06	4.89	6.04	3.77	1.81	1.69	1.86	1.74	1.73	1.33	1.15
Average bpc	**5.03**	**4.90**	**5.0**	**4.31**	**2.61**	**2.32**	**2.45**	**2.21**	**2.12**	**1.83**	**1.69**

Results of Gzip are obtained in high compression (Gzip -9) and the grammar based compressor Repair has been concatenated with a minimum redundancy Huffman coder [39]. The average bpc for M188 encoding is 5.0. It is noted that when M188 preprocessor is used in stand-alone mode it provides respective gain of 1% over Huffman and it cannot compete with the other DCAs. Table 4 proves that preprocessing performance is not proportional to stand-alone compression. It is

observed that worsen the stand alone performance may enhance the preprocessing performance according to the numerous trials had implemented for M188.

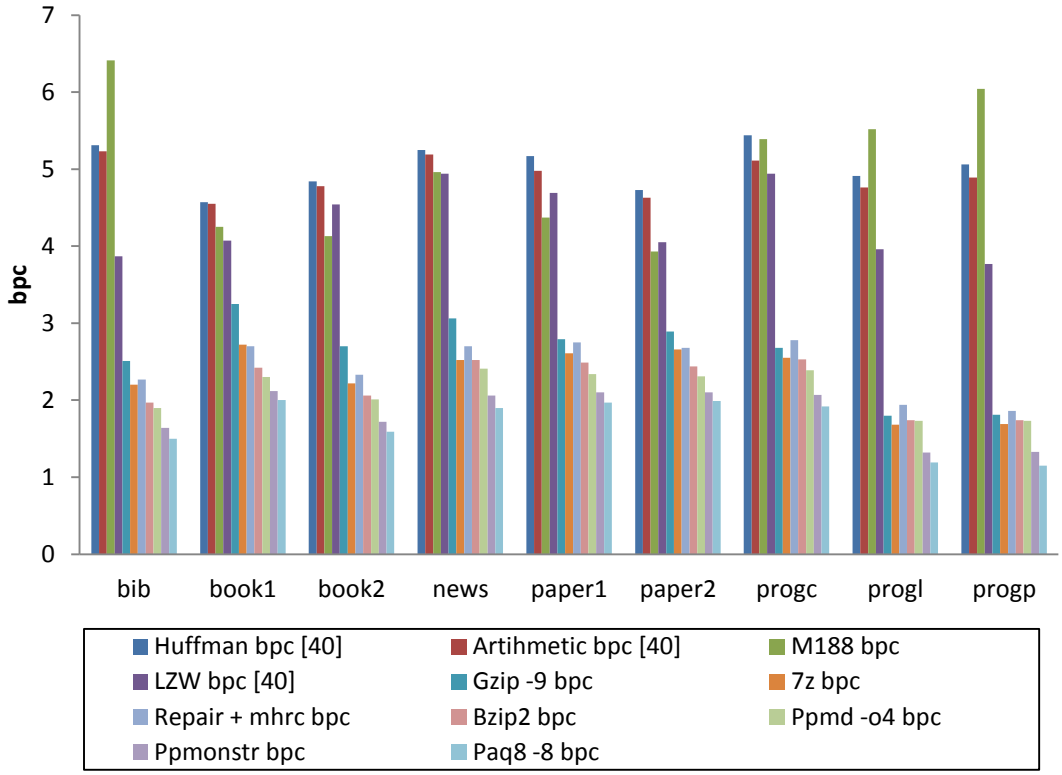


Figure 7: Stand alone compression effectiveness of M188 vs. DCAs on set 1

4.2 Preprocessors Concatenated with PPMD and PPMonstr

This section consists of three subsections each of those studies different experiment sets and provides the results for concatenation of preprocessors with PPMD and PPMonstr. The experimental presentation flow of this thesis is from general to specific by eliminating the worst resultant algorithms from the next experiment, the best resultants are kept for the final. First experiment is studied on the experiment set 1 with all preprocessors considered in the thesis.

4.2.1 Preprocessed PPMD and PPMonstr on set 1

Tables 5 and 6 provide compression effectiveness of LIPT, StarNT, WRT, M188, ETDC, SCDC and RPBC preprocessors when they are used prior to post-processors

such as PPMD and PPMonstr. These experiments consider only a subset of the Calgary corpus [35] which is set 1. Note that column five of the Table 5 also provides results for the Universal Processor of Abel and Teahan [36] concatenated with (PPMD+)[29]. The Universal preprocessor does not require an external dictionary and is known to work for all languages that are Latin based. [36] reports that, the Universal preprocessor makes use of techniques like capital letter and capitalized word conversion, end of line (EOL) coding, word replacement by tokens, replacement of the most frequent bigrams and trigrams and alphabet reordering. Last column of Table 5 has been reserved for compression results with mPPM [34]. Since mPPM first maps words into two byte codewords and then encodes the codewords using conventional PPM (two stage compressor), it is appropriate to compare it in this table with results obtained from other preprocessors concatenated with PPMD. A quick look at Table 5 shows that M188+PPMD, WRT+PPMD and StarNT+PPMD are the three best performing methods among the ones considered. Bpc for M188+PPMD is 1.89, for WRT+PPMD it is 1.92 and for StarNT+PPMD it is 1.93. Table 6 provides bpc values of M188+PPMonstr, LIPT+PPMonstr, StarNT+PPMonstr, WRT+PPMonstr, ETDC+PPMonstr and SCDC+PPMonstr. Experimental results point out that M188+PPMonstr, WRT+PPMonstr and StarNT+PPMonstr provide better compression in comparison with the others. Their respective bpc values are 1.60, 1.66 and 1.80. Thus, M188+PPMonstr has respective gains of 3.61% and 11.11 % over WRT+PPMonstr and StarNT+PPMonstr.

Table 5: Comparison of preprocessors in concatenation with PPMD on set 1

File	Size (bytes)	LIPT + PPMD order 5 bpc	StarNT + PPMD order 5 bpc	Universal + (PPMD+) bpc [29]	WRT4.6 + PPMD order 4 bpc	M188 + PPMD order 4 bpc	ETDC + PPMD order 4 bpc	SCDC + PPMD order 4 bpc	RPBC + PPMD order 4 bpc	mPPM bpc
bib	111,261	1.83	1.62	1.85	1.69	1.75	2.33	2.32	2.30	1.90
book1	768,771	2.23	2.24	2.20	2.10	2.09	2.57	2.56	2.55	2.23
book2	610,856	1.91	1.85	1.91	1.81	1.81	2.16	2.15	2.14	1.92
news	377,109	2.31	2.16	2.34	2.23	2.13	2.82	2.81	2.78	2.40
paper1	53,161	2.21	2.10	2.28	2.03	2.02	2.86	2.83	2.81	2.46
paper2	82,199	2.17	2.07	2.23	2.03	1.97	2.66	2.63	2.62	2.28
progc	39,611	2.30	2.17	2.32	2.25	2.11	3.04	2.99	2.98	2.58
progl	71,646	1.61	1.51	1.62	1.55	1.49	1.83	1.80	1.80	1.68
progp	49,379	1.68	1.64	1.66	1.67	1.63	1.86	1.83	1.82	1.69
Average bpc		**2.03**	**1.93**	**2.05**	**1.93**	**1.89**	**2.46**	**2.44**	**2.42**	**2.13**

Table 6: Comparison of preprocessors in concatenation with PPMonstr on set 1

File	Size (bytes)	LIPT + PPMonstr bpc	StarNT + PPMonstr bpc	WRT4.6 + PPMonstr bpc	M188 + PPMonstr bpc	ETDC + PPMonstr bpc	SCDC + PPMonstr bpc	RPBC + PPMonstr bpc
bib	111,261	1.81	1.63	1.46	1.35	2.04	2.04	2.03
book1	768,771	2.19	2.07	1.90	1.90	2.34	2.34	2.33
book2	610,856	1.91	1.72	1.58	1.60	1.94	1.94	1.93
news	377,109	2.14	2.05	1.91	1.78	2.48	2.47	2.46
paper1	53,161	2.08	2.00	1.80	1.80	2.59	2.57	2.57
paper2	82,199	2.28	1.97	1.82	1.78	2.44	2.42	2.42
progc	39,611	2.24	2.04	1.94	1.82	2.72	2.69	2.68
progl	71,646	1.59	1.33	1.23	1.18	1.61	1.59	1.59
progp	49,379	1.64	1.41	1.27	1.23	1.59	1.56	1.56
Average bpc		**1.99**	**1.80**	**1.66**	**1.60**	**2.20**	**2.18**	**2.17**

Figures 8 and 9 provide bar graphs for the data presented in Tables 5 and 6. The figures respectively show which preprocessors would excel while compressing the different source files. It can be seen from Figure 8 that when the post-processor is PPMD, M188 attains lower bpc values while compressing files 'news', 'paper1', 'paper2', 'progc', 'progl', 'progp' and 'book2'. WRT provides better gain for 'bib' only.

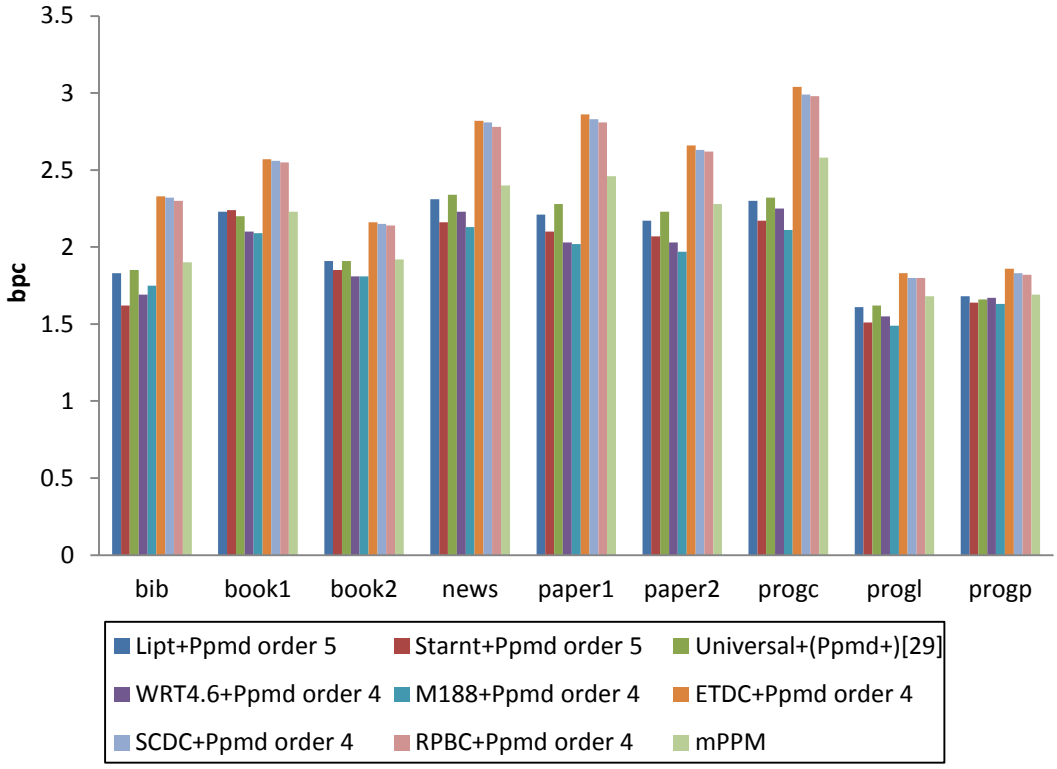


Figure 8: Comparison of preprocessors in concatenation with PPMD on set 1

With PPMonstr as the post-processor (see Figure 9), M188 gets lower bpc values for 'bib', 'news', 'paper1', 'paper2', 'progc', 'progl', 'progp'. WRT provides better gain for 'book2' only.

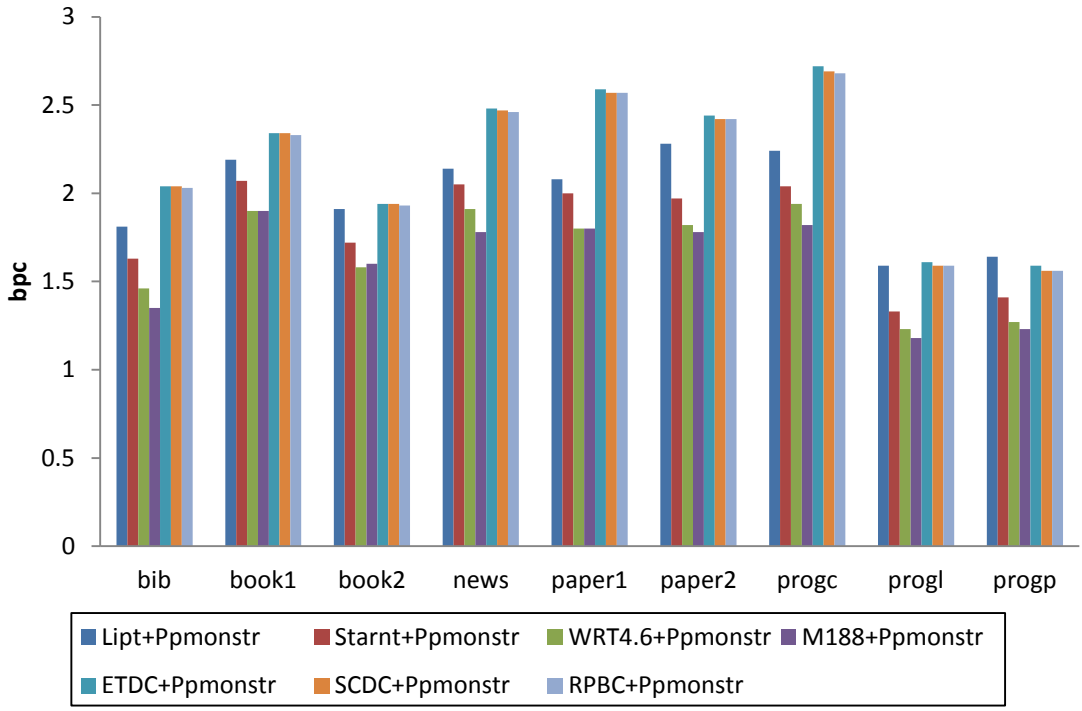


Figure 9: Comparison of preprocessors in concatenation with PPMonstr on set 1

4.2.2 Preprocessed PPMD and PPMonstr on set 2

A new set of experiment were carried out using text files from Gutenberg [42] and Canterbury [43] corpora where different preprocessors have been concatenated with PPMD and PPMonstr. During experiments PPMD with order-4 and PPMonstr with order-8 and memory limit of 256MB was assumed. bpc results while using PPMD and PPMonstr as post-compressor have respectively been provided in Tables 7 and 8. For both experiments WRT concatenated with the DCA would provide the best compression results on the average. For example when the post-processor is PPMD the average bpc values for WRT, M188 and StarNT are respectively 1.83, 1.84 and 1.89. Similarly, when the postprocessor is PPMonstr, the respective average bpc values are 1.62, 1.64 and 1.75. Thus M188+PPMonstr provide 6.29% gain over StarNT+PPMonstr and WRT+PPMonstr has 1.22% gain over M188+PPMonstr. In [21], it is stated that while compiling the dictionary of WRT a training corpus of 3 GB has been taken from the Project Gutenberg. This explains the lower bpc values

when WRT is using the Aspell's dictionary. Since, better training would lead to lower bpc values. Results are also available on the bar graphs Figure 10 and 11.

Table 7: Comparison of preprocessors in concatenation with PPMD on set 2

File	Size (bytes)	LIPT + PPMD order 4 bpc [18]	StarNT + PPMD order 4 bpc	WRT4.6 + PPMD order 4 bpc	M188 + PPMD order 4 bpc	ETDC + PPMD order 4 bpc	SCDC + PPMD order 4 bpc	RPBC + PPMD order 4 bpc
1musk10	1,349,139	1.85	1.82	1.72	1.78	2.03	2.03	2.02
anne11	587,051	2.04	2.01	1.91	1.96	2.27	2.26	2.25
alice29	152,089	2.06	2.00	1.90	1.91	2.37	2.35	2.34
asyoulik	125,179	2.35	2.24	2.24	2.18	2.77	2.75	2.74
lect10	426,754	1.86	1.78	1.72	1.70	2.07	2.06	2.05
bible	4,047,392	1.57	1.47	1.46	1.48	1.52	1.52	1.52
Average bpc		**1.96**	**1.89**	**1.83**	**1.84**	**2.17**	**2.16**	**2.15**

Table 8: Comparison of preprocessors in concatenation with PPMonstr on set 2

File	Size (bytes)	LIPT + PPMonstr bpc	StarNT + PPMonstr bpc	WRT4.6 + PPMonstr bpc	M188 + PPMonstr bpc	ETDC + PPMonstr bpc	SCDC + PPMonstr bpc	RPBC + PPMonstr bpc
1musk10	1,349,139	1.83	1.70	1.56	1.63	1.84	1.83	1.83
anne11	587,051	1.98	1.88	1.71	1.79	2.09	2.08	2.08
alice29	152,089	1.99	1.87	1.70	1.74	2.19	2.17	2.17
asyoulik	125,179	2.21	2.12	1.98	1.94	2.53	2.51	2.51
lect10	426,754	1.77	1.68	1.52	1.53	1.88	1.88	1.87
bible	4,047,392	1.58	1.32	1.25	1.27	1.34	1.33	1.33
Average bpc		**1.89**	**1.75**	**1.62**	**1.64**	**1.98**	**1.97**	**1.96**

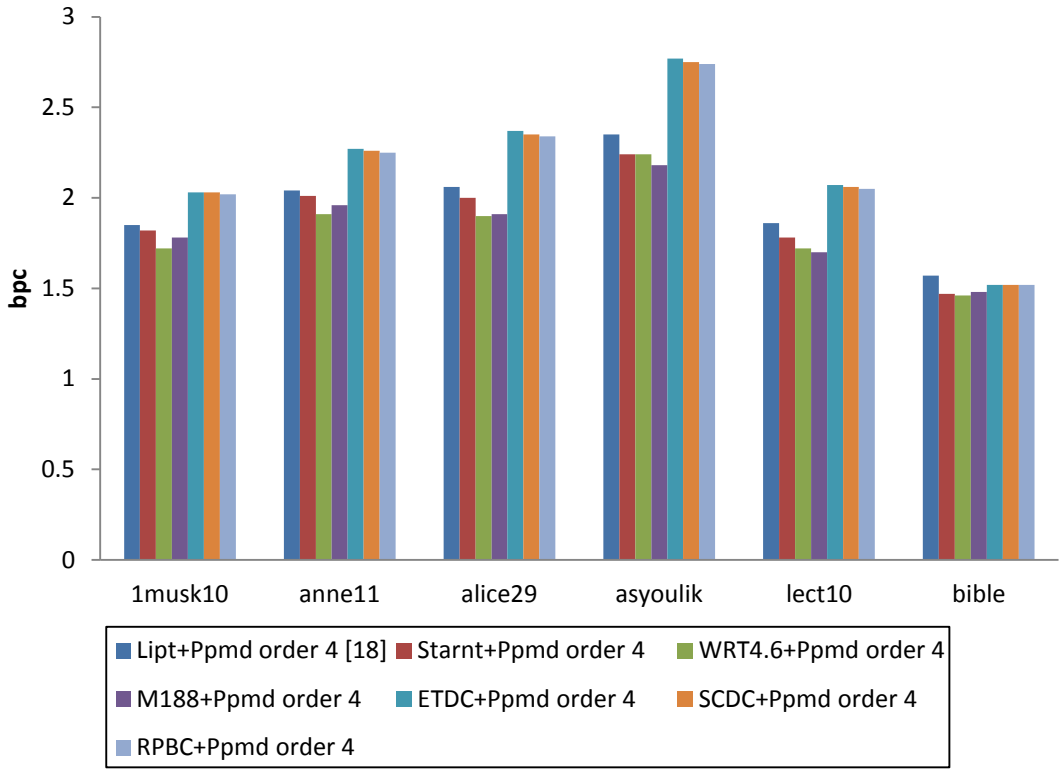


Figure 10: Comparison of preprocessors in concatenation with PPMD on set 2

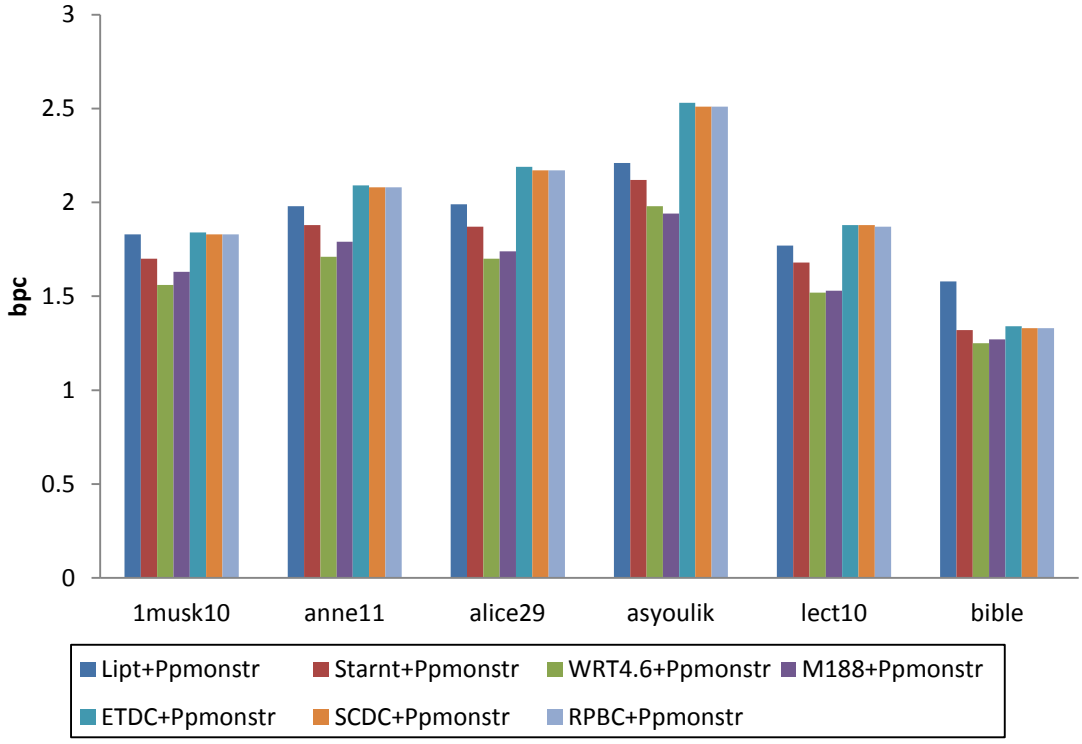


Figure 11: Comparison of preprocessors in concatenation with PPMonstr on set 2

4.3 Best Four Preprocessors Concatenated with PPMonstr on set 3

In this section the dictionary based WRT and M188 are compared against the word-based byte-oriented semi-static methods such as SCDC and RPBC. In this experiment, set 3 which contains seven medium-to-large size text files has been used. The first four files which were taken from the Project Gutenberg had names: *wealthnations, warpeace, big* and *dickens* and they were respectively 2.12, 4.23, 6.3 and 30MB in size. The files named *english50* and *english200* were taken from Pizza and Chili corpus. These files had previously been created by concatenation of English text files selected from etext02 - etext05 of Gutenberg Project, *wsj100* text file that is 100MB in size was taken from the TREC Project archives and this is the only text file not related to the Project Gutenberg . Figure 12 provides a comparative bar graph that shows the bpc values achieved by the algorithms considered when they are concatenated with PPMonstr (PPMD was not considered since earlier experiments showed that concatenating preprocessors with PPMonstr would provide lower bpc values).

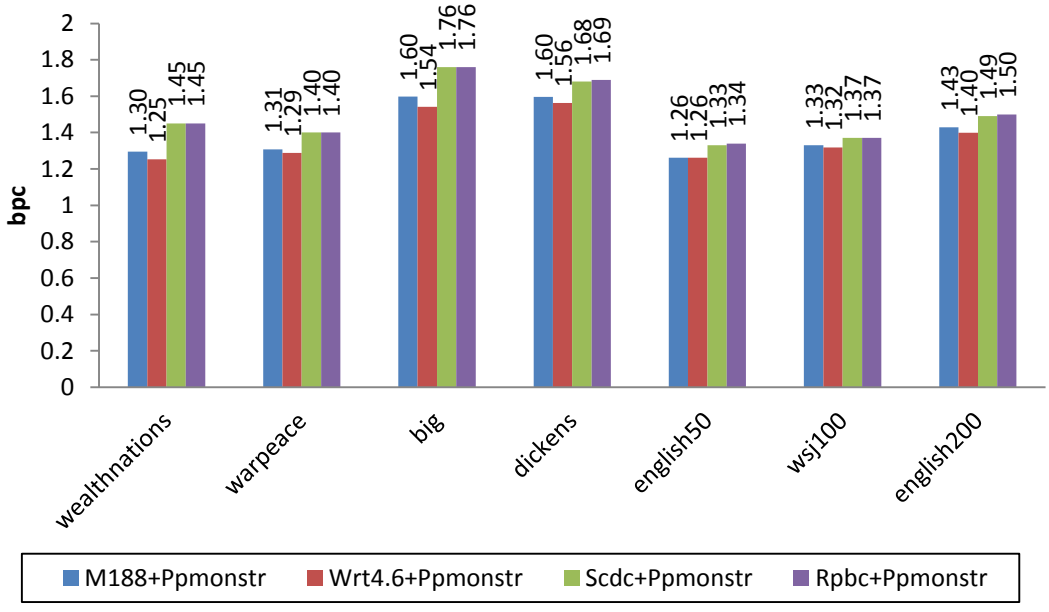


Figure 12: Best four methods concatenated with PPMonstr on set 3

For the set 3 files in Figure 12, the average bpc values for SCDC, RPBC, M188 and WRT were respectively 1.50, 1.50, 1.40 and 1.37. Results clearly show that both WRT and M188 achieve higher average gains than the byte-oriented semi-static methods: SCDC and RPBC. For the 100MB *wsj100* text file which is not from the Gutenberg Project Library the bpc difference between WRT and M188 is 0.01, and this corresponds to 140KB. For the 50MB *english50* text file M188 and WRT have same bpc values. It is also noted that as the file size became larger the difference between dictionary based and semi-static methods would become less significant. However, since most of the time the files one would like to exchange are smaller than 200MB, it is fair to say that for small to moderately large files the dictionary based methods would overcome the semi-static byte-oriented methods.

4.4 Timing Performance of M188

In this section the experiments were carried out along with the timing set which contains Calgary corpus files (Table I of [46]). The results shown in the Figure 13 are ensemble average values of the time measurements of 10 runs for each file. The experiments were carried out on a 2.5 GHz Intel core i5 CPU supported by 3GB of RAM. Encoding times depicted in Figure 13 point out that M188 can encode faster than WRT, RPBC, PPMD -o4, PPMonstr, Bzip2 and of course PAQ8l. M188 and all pre or post-processors are much slower than the semi-static byte-oriented preprocessors, namely: ETDC and SCDC. Similarly Figure 13 shows that M188 decodes faster than PPMonstr and PPMD-o4 and is 0.005 seconds slower than both Bzip2 and WRT. ETDC and SCDC are very fast in comparison to all the other algorithms, particularly in decoding.

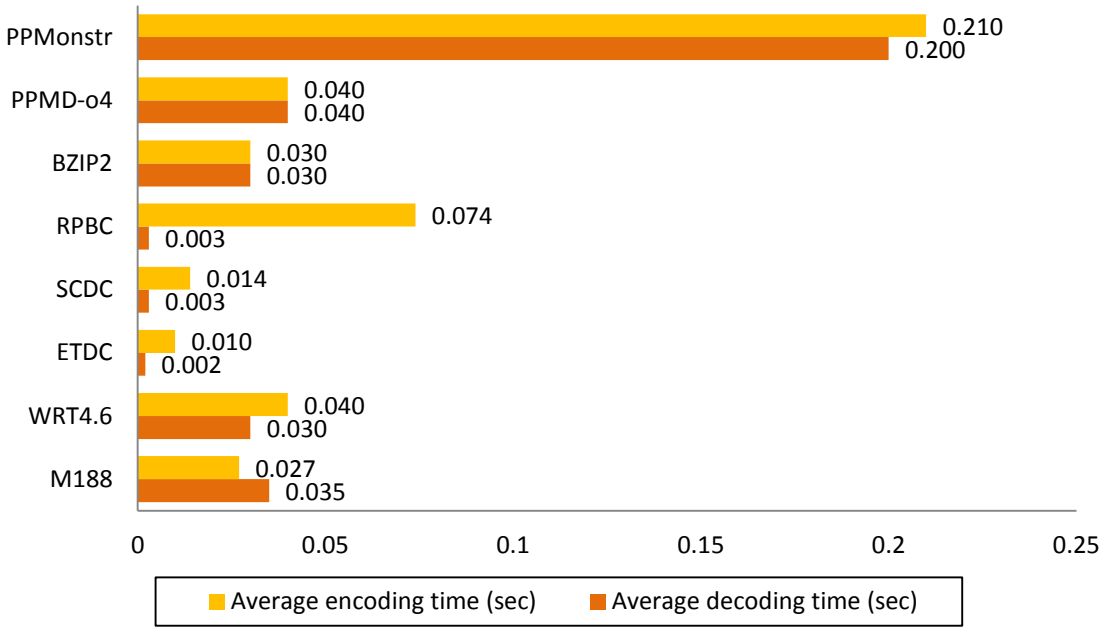


Figure 13: Timing performances of the algorithms

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Conclusions

A new source coding algorithm named Metehan188 is proposed which can be used as a preprocessor for well-known backend compressors. The proposed M188 algorithm has simple logic and compression gains attained in concatenation with different post-processing algorithms indicate that M188 is either better or just as effective as the selected state-of-the-art preprocessors. In different experimental setups M188 and WRT can achieve higher compression when compared to the semi-static word-based byte-oriented methods: namely ETDC, SCDC and RPBC. While using the Calgary corpus M188 outperforms all the other preprocessors when concatenated with PPMD or PPMonstr. In experiments using the Project Gutenberg text files bpc values for M188 are slightly higher than those of WRT but M188 overcomes the remaining algorithms. In the experiment where WRT and M188 have been compared with the semi-static byte-oriented preprocessors using medium to large size text files, both M188 and WRT have provided higher average gains. Among themselves WRT overcomes M188 for Project Gutenberg related files.

5.2 Future Work

This thesis proves that there is room for improvement of M188 such as; enhancing EOL analysis, alphabet re-ordering and dictionary re-ordering can carry M188 as the state-of-the-art preprocessing technique.

REFERENCES

[1] Huffman, D. A., "A method for the construction of minimum-redundancy codes", *In Proceedings of the Institute of Radio Engineers*, Sept 1952, pp.1098-1101.

[2] Gallager, R. G., "Variations on a Theme by Huffman", *IEEE Transactions on Information Theory*, Nov 1978, Vol.24, No.6, pp. 668-674.

[3] Rissanen, J., and Langdon, G. G., "Arithmetic coding", *IBM Journal of Research and Development*, 1979, (28), pp.149-162.

[4] Cleary, J. G., and Witten, I. H., "Data compression using adaptive coding and partial string matching", *IEEE Transactions on Communications*, Apr 1984, 32(4), pp. 396-402.

[5] Mahoney, M, "The PAQ6 data compression program". Retrieved on: September, 2014 . Available: http://www.cs.fit.edu/ mmahoney/compression/paq6v2.exe

[6] Moura, E., Navarro, G., Ziviani, N., and Baeza-Yates, R., "Fast and flexible word searching on compressed text", *ACM Transactions on Information Systems*, 2000, 18(2), pp. 113-139.

[7] Brisaboa, N., Iglesias, E., Navarro, G., and Parama, J.,"An efficient compression code for text databases", *25th European Conference on IR Research*, ECIR 2003, LNCS 2633, pages 468481.

[8] Brisaboa, N., Farina, A., Navarro, G., and Parama, J., "Leightweight natural language text compression", *Information Retrieval,* 10(1), 2007, pp. 1-33.

[9] Culpepper, J.S., Moffat, A., "Enhanced byte codes with restricted prefix properties", *Proc. of 12th Int. Symp. on String Processing and Information Retrieval*, LNCS 3772, Springer-Verlang, 2005, pp. 1-12.

[10] Silva de Mura, E., Navarro, G., Ziviani, N., and Baeza-Yates, R., "Fast and Flexible Word Searching on Compressed Text", *ACM Transactions on Information Systems*, 18(2): 113-139,2000.

[11] Brisaboa, N., Farina, A., Ladra, S., and Navarro, G., "Implicit Indexing of Natural Language Text by Reorganizing Bytecodes", *Information Retrieval,* 15(6), pp. 527-557, 2012.

[12] Brisaboa, N., Farina, A., Navarro, G., and Parama, J. R., "New adaptive compressors for natural language text", *Software Practice and Experience*, 2008, pp. 1-23.

[13] Ziv, J., and Lempel, A., "A universal algorithm for sequential data compression", *IEEE Transactions on Information Theory*, May 1977, IT-23(3), pp. 337-343.

[14] Welch, T.A., "A Technique for High-Performance Data Compression," *Computer*, June 1984, vol.17, no.6, pp.8,19.

[15] Deutsch, P., "Deflate compressed data format specification, version 1.3", *Network Working Group*, 1996.

[16] Nelson, M. R., "Star Encoding", *Dr. Dobb's Journal*, August 2002.

[17] Awan, F. S., Zhang, N., Motgi, N., Iqbal, R. T., and Mukherjee, A., "LIPT: A reversible lossless text transform to improve compression performance", *Data Compression Conference*, Mar 2001, pp. 481-494.

[18] Awan, F., and Mukherjee, A., "LIPT: A lossless text transform to improve compression", *Proc. of Int. Conf. on Information Technology: Coding and Computing*, Apr 2001, pp. 452-460.

[19] Sun, W., Mukherjee, A., and Zhang, N., "A dictionary-based multi corpora text compression system", *Proc. of Data Compression Conference*, Mar 2003, pp. 1-11.

[20] Radescu, R., "Star-derived transforms in lossless text compression", *Int. Symp. on Signals, Circuits and Systems*, Jul 2009, pp. 1-6.

[21] Skibinski, P., Grabowski, S., and Deorowicz, S., "Revisiting dictionary based compression", *Software: Practice and Experience*, Dec 2005, Vol. 35, Issue 15, pp. 1455-1476.

[22] Rexline, S. J., and Robert, L., "IWRT: Improved Word Replacement Transformation in Dictionary Based Lossless Text Compression", *European Journal of Scientific Research*, ISSN 1450-216x, Sept 2012, Vol. 86, No. 2, pp. 193-201.

[23] S. W. Golomb, "Run-length encoding", *IEEE Trans. on Information Theory*,1966,12(3),pp. 337-343.

[24] Burrows, M., and Wheeler, D. J., "A Block-sorting Lossless Data Compression Algorithm", *Digital Systems Research Center, Research Report 124*, 1994.

[25] Cormack, G. V., and Horspool, R.N., "Data compression using dynamic Markow modelling", *The Computer Journal*, Dec 1987, 30(6), pp. 541-550.

[26] Seward, J., "On the performance of BWT sorting algorithms", *Data Compression Conference*, Mar 2000, pp. 173 182.

[27] Effros, M., Visweswariah, K., Kulkarni, S.R., and Verdu, S., "Universal Lossless Source Coding with the Burrows Wheeler Transform", *IEEE Transactions on Information Theory*, Vol.48, No. 5, pp. 1061-1081, May 2002.

[28] Moffat, A., "Implementing the PPM data compression scheme", *IEEE Transactions on Communications*, Vol. 38, No. 11, pp. 1917-1921 ,Nov 1990.

[29] Teahan, W., "Probability Estimation for PPM", *Proc. of the New Zealand Computer Science Research Students' Conference*, University of Waikato, New Zealand, 1995.

[30] Shkarin, D., "PPMD Compressor Ver. J.", Retrieved on: September, 2014, . Available: http://compression.ru/ds/.

[31] Teahan, E. J., and Witten, I. H., "Unbounded length contexts for PPM", *Data Compression Conference*, Mar 1995, pp. 52-61.

[32] Shkarin, D., "Monstrous PPMII compressor based on PPMD var. I.", Retrieved on: September, 2014, Available: http://compression.ru/ds/, 2004.

[33] Shkarin, D.,"The Durilca and Durilca Light 0.4a programs", Retrieved on: September, 2014, Available: http://www.compression.ru/ds/durilca.rar

[34] Adiego, J., Martinez-Prieto, M. A., and Fuente de la P., "High Performance Word-Codeword Mapping Algorithm on PPM", *Data Compression Conference*, 2009, pp. 23-32.

[35] Bell, T. "Calgary corpus", Retrieved on: January, 2012. Available: http://www.data-compression.info/Corpora/CalgaryCorpus/.

[36] Abel, J., and Teahan, W., "Universal Text Preprocessing for Data Compression", *IEEE Transactions On Computers*, May 2005, Vol. 54, No. 5, pp.497-507.

[37] Batista, L., and Alexandre, L. A., "Text pre-processing for lossless compression", *Data Compression Conference*, Mar 2008, pp. 506-516.

[38] Brisaboa, N.R., Farina, A., Navarro, G., and Parama, J.R., "Improving semistatic compression via phrase-based modelling", *Information Processing & Management*, Elsevier Science, Vol. 47, Iss: 4, July 2011, pp. 545-559.

[39] Turpin, A., and Moffat, A., "On the Implementation of Minimum-Redundancy Prefix Codes", *IEEE Transactions on Communications*, 45(10), pp. 1200-1207, Oct 1997.

[40] Robert, L. and Nadarajan, R., "Simple lossless preprocessing algorithm for text compression", *IET Software*, Aug 2009, Vol. 3, Iss. 1, pp. 37-45.

[41] Atkinson, "Spell Checking Oriented Word Lists (SCOWL) Revision 5", 2002, Retrieved on: September, 2014. Available: http://wordlist.sourceforge.net

[42] Project Gutenberg, 19712012, Retrieved on: February, 2013. Available: http://www.promo.net/pg/.

[43] Bell, T., and Powell, M., "The Canterbury Text compression corpora", Retrieved on: January, 2012. Available: http://corpus.canterbury.ac.nz/descriptions/.

[44] Ferragina, P., and Navarro, G., "Pizza and Chili Corpus Compressed Indexes and their Testbeds", Retrieved on: September, 2014. Available: http://pizzachili.dcc.uchile.d/texts/nlang/.

[45] Text REtreival Conference, "Text Research Collection Volume 1 and Volume 2", Retrieved on: September, 2014. Available: http://trec.nist.gov/data.html.

[46] Sun, W., Zhang, N., and Mukherjee, A., "Dictionary-based fast transform for text compression", Proc. of Int. Conf. on Information Technology: Computers and Communications, Apr 2003, pp. 176-182.

[47] Teahan, W. J., and Cleary, J. G., "The entropy of English using PPM-based models", Data Compression Conference, Mar 1996, pp. 53-62.

[48] Letter frequency. (2014, October 19). In Wikipedia, The Free Encyclopedia. from http://en.wikipedia.org/w/index.php?title=Letter_frequency&oldid=630284810

[49] Mahoney, M. V., "Adaptive weighing of context models for lossless data compression.", *Technical Report*, CS-2005-16, 2005.

[50] PAQ. (2014, June 27). In Wikipedia, The Free Encyclopedia. Retrieved 20:27, October 27, 2014, from http://en.wikipedia.org/w/index.php?title=PAQ

[51] Bell, T., "The large compression corpora", Retrieved on: February, 2013. Available: http://corpus.canterbury.ac.nz/descriptions/#large.

[52] Mahoney, M. V., "Data Compression Programs", Retrieved on: September, 2014. Available: http://mattmahoney.net/dc/PAQ8l.zip.