The data compression field has always been an important part of computer science and it is becoming increasingly popular and important today. Although computers become faster and data storage becomes less expensive and more efficient, the increased importance of usage of data necessitates the use of at least a small measure of data compression due to vast storage and transmission requirements. The question in many applications is no longer whether to compress data, but what compression method should be applied.

In the last decade, there has been an unprecedented explosion of textual information flow over the internet through electronic mail, web browsing, digital library and information retrieval systems, etc. Given the continued increase in the amount of data that needs to be transformed or archived, the importance of data compression is likely to increase in the near future. So, it is often necessary to reduce the size of files to reduce proportionally their transmission times. Data Compression is a process that reduces the data size, removing the excessive information. In addition to reducing the transmission time, the other advantages in compressing files are related to two connected issues, namely, integrity of data and security. Of the two related issues,
integrity of data is easily accomplished through redundancy checks during the decompression phase. The second issue namely security requires data compression before applying cryptography [63].

There are many known methods for data compression. They are based on different ideas, suitable for different types of data, and produce different results, but they are all based on the same principle, namely, compression of data by removing the redundancy from the original data. Any non-random collection data has some structure, and this structure can be exploited to achieve a smaller representation of the data, which is discernible [28].

Data Compression techniques can be divided into two major families, namely, Lossy and Lossless. Lossy data compression concedes a certain loss of accuracy in exchange for greatly increased compression. Lossy compression proves effective when applied to graphic images and digitized voices. By their very nature, these digitized representations of analog phenomena are not perfect to begin with, and hence the idea of output and input not matching exactly is a little more acceptable. Most lossy compression techniques can be adjusted to different quality levels, gaining high accuracy in exchange for less effective compression [3]. Until recently, lossy compression has been primarily implemented using dedicated hardware. During the past few years, powerful lossy compression programs have been moved to desktop CPUs, but even then the field is still dominated by hardware implementations. The lossy compression methods typically yield much better compression ratios than lossless algorithms and hence if we can accept some distortions of data, these methods can be used. There are however, situations in which the lossy methods must not be
used to compress picture data. In many countries, the medical images can be compressed only by the lossless algorithms because of its importance.

Lossless compression techniques are used when storing database records, spreadsheets or word processing files. In these applications, the loss of even a single bit could be catastrophic. Lossless compression techniques guarantee the generation of an exact duplicate of the input data stream after a compress-expand cycle. All the algorithms analyzed in this thesis are lossless.

When discussing compression algorithms it is important to make a distinction between two components: the model and the coder. The model component somehow captures the probability distribution of the messages by knowing or discovering something about the structure of the input. The coder component then takes advantage of the probability biases generated in the model to generate codes. It does this by effectively lengthening low probability messages and shortening high-probability messages. The models in most current real-world compression algorithms, however, are not so sophisticated, and use more mundane measures such as repeated patterns in text. Although there are different ways to design the model component of compression algorithms and a huge range of levels of sophistication, the coder components tend to be quite generic in current algorithms. They are almost exclusively based on either Huffman [3] or arithmetic codes [13].

One of the main strategies in developing compression methods is to prepare a specialized compression algorithm for the data one has to transmit or store. The compression algorithm developed by analyzing the nature of input data it takes for
Random access and preprocessing algorithms for text data compression

compression, would provide better compression ratio. In general, it is not possible to prepare a specialized compression method for each type of data. The main reasons are, it would result in a vast number of algorithms and the cost of developing a new compression method could surpass the gain obtained by the reduction of the data size.

The lossless compression algorithms can be classified into Statistical methods and Dictionary methods. Statistical compression methods known as entropy coders use a statistical model of the data, and the quality of compression depends on how good that model is. An entropy coder is a method that assigns a code to every symbol from the alphabet depending on the probability of symbol occurrence. The symbols that are more probable to occur get shorter codes than the less probable ones. The codes are assigned to the symbols in such a way that the expected length of the compressed sequence is minimal. The most popular entropy coders are Huffman coder and an Arithmetic coder. The Huffman coder [3] serves as the basis for several popular programs used on personal computers. The Huffman coder is optimal in the class of methods that assign codes of integer length, while the arithmetic coder is free from this limitation. The Huffman method produces best results only when the symbols have probabilities of occurrence that are negative powers of 2 [93]. This is because this method assigns a code with an integral number of bits to each symbol in the alphabet. Arithmetic coding [14] overcomes this problem by assigning one code to the entire input stream, instead of assigning codes to the individual symbols.

Another statistical compression method, a Dynamic Markov Coder (DMC), was invented by Cormack and Horspool [17] in 1987. Their algorithm assumes that the data to compress are output of some Markov source class, and during the
compression it tries to discover this source by better estimating the probability of occurrence of the next symbol. Using this probability, the codes for symbols from the alphabet are assigned by making use of an arithmetic coder. This algorithm also needs a lot of memory to store the statistics of symbol occurrences and runs rather slowly.

In 1995, an interesting compression method, a Context Tree Weighting (CTW) algorithm, was proposed by Willems et al. [18]. The authors introduced the concept of a context tree source class. In their compression algorithm, it is assumed that the data are produced by some source of that class, and relating on this assumption the probability of symbol occurrence is estimated.

Dictionary based compression methods [28] do not use a statistical model. Instead, they select strings of symbols and encode each string as a token using a dictionary. The dictionary holds strings of symbols and it may be static or dynamic (adaptive). The former is permanent, sometimes allowing the addition of strings but no deletions, whereas the later holds strings previously found in the input stream, allowing for additions and deletions of strings as new input is being read. A number of dictionary based lossless data compression algorithms were proposed. Nowadays they are widely used. Ziv and Lempel [15, 16] in 1977–78 proposed to search the data to compress for identical parts and to replace the repetitions with the information where the identical subsequences appeared before. This task can be accomplished in several ways. Ziv and Lempel proposed two main variants of their method: LZ77 [15], which encodes the information of repetitions directly, and LZ78 [16], which maintains a supporting dictionary of subsequences appeared so far, and stores the indexes from this dictionary in the output sequence. The main advantages of these methods are high speed and ease of implementation.
Another compression method proposed in 1994 is a block-sorting compression algorithm, called Burrows–Wheeler Compression Algorithm (BWCA) [19]. The main concept of their algorithm is to build a matrix, whose rows store all the one-character cyclic shifts of the compressed sequence, to sort the rows lexicographically, and to use the last column of this matrix for further processing. This process is known as the Burrows–Wheeler Transform (BWT). The output of the transform is then handled by a move-to-front transform [20], and in the last stage, compressed by an entropy coder, which can be a Huffman coder or an arithmetic coder. As the result of the BWT, a sequence is obtained, in which all symbols appearing in similar contexts are grouped. The important features of the BWT-based methods are their high speed of running and reasonable compression ratios [59].

**Thesis contribution**

All the existing lossless compression methods are adaptive models which generate variable-bit-length codes or only generate variable-bit-length code that must be decoded sequentially from beginning to end. In this case, random access to large data sets can be provided only by dividing the data into smaller files or blocks and compressing each chunk separately. Even if an application is sophisticated enough to decompress separate blocks of data, it still must begin decompressing each block at the start of that particular block. In this thesis attention is focused on developing these types of random access compression algorithms. All modern lossless compression algorithms allow decompression only from the beginning of the compressed text. If a byte of an input to the decompressor is corrupted by transmission errors, the entire file is affected and cannot be decoded. If a file is compressed by a Random Access Data
Compression algorithm, then random access to a compressed file is possible. To access a specific group of bytes the entire compressed file need not be decompressed. Identify the position and decompress only the portion, where we want to refer/access. That is, the decompression may start from any place in the compressed file, and not necessarily from the beginning. Random Access Data Compression algorithms compress a file in such a way that the decompression process need not be initiated from the beginning. This provides fast access in a compressed file which contains large volume of data. Unlike modern compression algorithms, if a byte of an input to the decompressor is corrupted by transmission errors, only a similar fraction of output bytes is affected. Thus Random Access Data Compression algorithms are more suitable to real time mission-critical and data transmission applications. These algorithms are also used for managing compressed data base file. Based on these, separate area on managing compressed data bases may be evolved.

For example, there is a simple text compression trick that allows random access in the compressed file. It employs the unused higher order bit in ASCII characters to indicate that the preceding space character has been removed. This technique in effect removes all single spaces and reduces runs of consecutive spaces to half length, compressing typical text by atleast 15 percent [21]. Another simple approach is to encode common words as one byte using the fixed dictionary of the most frequently used words [38]. But schemes such as these are not for general purpose and have limited usefulness.

Byte Pair Encoding (BPE) [21] scheme is a universal compression algorithm that supports random access in the compressed file. The global substitution process of BPE produces a uniform data format that allows decompression to begin anywhere in
the data. Using BPE, data from anywhere in a compressed block can be immediately decompressed without having to start at the beginning of the block. This can provide a very fast access for some applications. It is like being able to grab a volume of an encyclopedia off the shelf and open it to the exact page you want without having to flip through all the earlier pages in the volume. ASCII character set uses only bytes from 0 to 127. This method uses 128 to 255 for replacing the most frequently occurring pairs of adjacent bytes. The BPE operates by finding the most frequently occurring pair of adjacent bytes in the text and replacing all instances of the pair with a code from 128 to 255. The compressor repeats this process until no more compression is possible or all codes from 128 to 255 are used for replacements. The algorithm records the details of the replacements in a table, which is stored as a header in the compressed file.

This thesis suggests few random access text data compression algorithms. The algorithm BPE+ is a variation of BPE that uses not only the byte codes from 128 to 255 but also the unused bytes from 0 to 127 of the input file. BPE++ is a multi stage random access text compression algorithm. After the substitution of pairs by pair codes, we found some of the bytes that are used earlier become unused after the substitution. By collecting and using those unused bytes as byte codes, again the process of substituting the frequent pairs is repeated. This process is repeated again and again until all the byte values from 0 to 255 are used in the output file or there is no frequent pair to replace. BTE is another proposed random access text compression algorithm, which replaces the most frequently appearing triplets by codes in between 128 to 255. While replacing, if there are no more frequent triplets, the algorithm replaces the most frequently occurring pairs. Another proposed algorithm BQE is
replacing the most frequently appearing quadruplets by the codes from 128 to 255. While replacing, if there are no more frequent quadruplets, the algorithm replaces the most frequently occurring triplets and pairs. The concepts of BQE and BPE++ are combined together to suggest another algorithm BQE+ for random access text compression. These algorithms compress a file in such a way that the decompression process need not be initiated from the beginning. The decompression process may be started anywhere in the decompressed file. This provides fast access in a compressed file which contains large volume of data.

Another approach, which is taken up for analysis in this thesis, is to perform a lossless, reversible transformation to a source file prior to applying an existing compression algorithm. The transformations are designed to make it easier to compress the source file. The original text file is provided as input to the transformation, which outputs a transformed text. This output is provided to an existing, unmodified data compression algorithm such as LZW, which compresses the transformed text. To decompress, one merely reverses this process, by first invoking the appropriate decompression algorithm, and then providing the resulting text to the inverse transform.

The thesis begins, in Chapter 2, with the formulations of data compression problem. In this chapter, a modern paradigm of data compression, modeling and coding, is described. The statistical methods, such as Huffman and Arithmetic Coding with their implementations are presented. The popular dictionary methods such as LZ77, LZW and their variations are well analyzed. This chapter ends with the BWT compression technique.
Random access text compression algorithms have been discussed in Chapter 3. In the first part of this chapter, the universal method for random access data compression algorithm Byte Pair Encoding is analyzed with illustrations. Six new algorithms for random access text compression are proposed and discussed. The experimental table is presented at the end of the chapter and results are analyzed.

In Chapter 4, the need for preprocessing the input data before applying compression process has been discussed. These preprocessing techniques tend to improve the compression ratio of the back end algorithms better. This chapter also provides a modern paradigm of text preprocessing.

Chapter 5, a few reversible transformation techniques which improve the back-end algorithm’s ability to compress have been proposed. We propose a method, which transforms a text file into intermediate file with minimum possible byte values. An attempt has been made to reduce the number of possible bytes that appeared after every byte in the source file. Some techniques are also proposed to handle the most frequently appearing characters such as space and eoln. Results show that there is significant improvement in the compression ratio after the proposed transformations.

Chapter 6 provides applications of data compression algorithms. The applications of data compression in the areas of Cryptography, Networks and Data Base Management are analyzed. Chapter 7 contains a discussion on obtained results and justification of the thesis.