Pattern Matching in Compressed Texts and Images

Tim Bell*, Department of Computer Science,
University of Canterbury, New Zealand

Don Adjeroh, Lane Department of Computer Science
and Electrical Engineering, West Virginia University,
Morgantown, WV 26506-6109, USA

Amar Mukherjee†, School of Electrical Engineering
and Computer Science, University of Central Florida,
Orlando, FL. 32816, USA

May 23, 2001

Draft of May 23, 2001

Abstract

This paper provides a survey of techniques for pattern matching in compressed text and images. Normally compressed data needs to be decompressed before it is processed, but if the compression has been done in the right way, it is often possible to search the data without having to decompress it, or at least only partially decompress it. The problem can be divided into lossless and lossy compression methods, and then in each of these cases the pattern matching can be either exact or inexact. Much work has been reported in the literature on techniques for all of these cases, including algorithms that are suitable for pattern matching for various compression methods, and compression methods designed specifically for pattern matching. This work is surveyed in this paper. The paper also exposes the important relationship between pattern matching and compression, and proposes some performance measures for compressed pattern matching algorithms. Ideas and directions for future work are also described.

Keywords: compressed pattern matching, text compression, image compression, performance measures, searching

*Supported by a grant from the Hitachinaka Techno Center Inc, Japan.

†This work has been partially supported by a grant from the National Science Foundation IIS-9977336.

CONTENTS

1 INTRODUCTION
2 COMPRESSION FOR TEXT AND IMAGES
3 SEARCH STRATEGIES
4 RELATIONSHIP BETWEEN SEARCHING AND COMPRESSION
5 SEARCHING COMPRESSED DATA: PERFORMANCE MEASUREMENT
6 SEARCHING COMPRESSED DATA: LOSSLESS COMPRESSION
7 SEARCHING COMPRESSED DATA: LOSSY COMPRESSION
8 DIRECTIOSN FOR FURTHER RESEARCH
9 CONCLUSION

1 Introduction

Computers are increasingly being used to process text, digitised images, digital video, and various other types of data. However, the data required to represent an image generally requires large amounts of storage space, and operations on the data such as pattern matching are time consuming. Even the amount of textual information typically available to the ordinary user has witnessed a tremendous growth due to a lot of factors, such as improvements in storage and communication technologies, widespread deployment of digital libraries, improved document processing techniques, the world wide web, electronic mail, wireless communication, etc. Apart from the problem of sheer size, the huge amounts of data involved also pose problems for efficient search and retrieval of the required information from the stored data.

Since the digitized data is usually stored using some compression technique, and because of the problem of efficiency (in terms of both storage space and computational time), the trend now is to keep the compressed data in its compressed form for as much time as possible. That is, operations such as search and analysis on the data (be it text or images) are performed directly on the compressed representation, without decompression, or at least, with minimal decompression. Intuitively, compared to working on the original uncompressed data, operating directly on the compressed data will require the manipulation of less data, and hence should be more efficient. This also avoids the often time-consuming process of decompression, and the problem of storage space that may be required to keep the decompressed data. The need to search data directly in its compressed form is even being recognized by new international compression standards such as MPEG-4 (Sikora, 1997; MPEG-4, 2000) where part of the requirement is the ability to search for objects directly in the compressed video.

This paper surveys techniques that solve the two basic problems of efficiency (in storage space and computational time) at the same time. That is, the digitised image or text is stored and searched in a compressed format. It might seem that compression and searching work against each other, since a simple system would have to decompress a file before searching it, thus slowing down the pattern matching process. However, there is a strong relationship between compression and pattern matching, and this can be exploited to enable both tasks to be performed efficiently at the same time.

In fact, pattern matching can be regarded as the basis of compression. For example, a *dictionary* compression system might identify English words in a text, and replace these with a reference to the word in a lexicon. The main task of the compression system is to identify patterns (in this example, words), which are then represented using some sort of compact code. If the type of pattern used for compression is the same as the type being used during a later search of the text, then the compression system can be exploited to perform a fast search. In the example of the dictionary system, if a user wishes to search the compressed text for words, then they could look up the word in the lexicon, which would immediately establish whether a search will be successful. If the word is found, then its code could be determined, and the compressed text searched for the code. This will considerably reduce the amount of data to be searched, and the search will be matching whole words rather than a character at a time. In one sense, much of the searching has already been performed off-line at the time of compression.

The potential savings are significant. Text can be compressed to less than a half of its original size, and images are routinely compressed to a tenth or even a hundredth of the size of the raw data. These factors indicate that there is considerable potential to speed up searching, and indeed, systems exist that are able to achieve much of this potential saving. For instance, compressed domain indexing and retrieval is the preferred approach to multimedia information management (Ahanger and Little, 1996; Mandal et al., 1999a), where orders of magnitude speedup has been recorded over operations on uncompressed data (Yeo and Liu, 1996; Adjeroh and Lee, 1997).

In this paper, we survey compression methods, especially identifying the search techniques that they use, and how they could be exploited for searching the compressed text later. We then look at pattern matching methods for uncompressed data, to set the scene for more sophisticated systems. This is followed by a discussion of performance measurement for compressed pattern matching. The next two sections of the paper survey techniques that have been developed for searching compressed data, which is sometimes called *compressed-domain pattern matching*. The first section looks at methods for lossless compression, where the original data is stored in a way that it can be reconstructed exactly. The second section surveys methods where the data has been compressed using lossy methods, which are usually used for images. A number of techniques have been developed for these tasks; some adapt existing compression methods, while others propose new methods that are well suited to searching. The paper concludes with speculation on the likely directions of future work in the area.

2 Compression methods for text and images

We begin with a brief survey of compression methods, focusing particularly on the aspects that make compressed-domain pattern matching particularly easy or difficult. More complete introductions to the topic of compression are

available (Cohn, 1994; Bell, 2000), and the reader is referred to textbooks for more details (Bell et al., 1990; Salomon, 1998; Witten et al., 1999). There is also a conference series on data compression - see (Storer and Reif, 1991; Storer and Cohn, 1992; Storer and Cohn, 1993; Storer and Cohn, 1994; Storer and Cohn, 1997; Storer and Cohn, 1998).

Compression methods are generally classed as *lossless* or *lossy*. Lossless methods enable the original data to be recovered exactly, and are important for general purpose situations. Lossless compression, sometimes called *text compression*, is typically used for text, and to a lesser extent on images, such as in medical imaging where exact reconstruction of the original image is important. In contrast, lossy methods allow some deterioration in the original data, and are generally applied in situations where the data has been digitised from an analogue source (such as images and audio). Usually the level of deterioration is near-imperceptible, yet considerable compression improvement can be achieved because the system is not storing unnecessary detail. Many lossy methods include a lossless method as a sub-component. For example, an image compression system might transform the image to a frequency domain, and then encode only the lower frequency components using a lossless method. Thus techniques for lossless compression can be of relevance to lossy systems, such as image compression systems.

Figure 1 shows a general model of the data compression process. The data transformation stage transforms the input data into some form that will expose the redundancies or repetitions in the data. The encoding stage (also called the coding stage) codes the data to remove the exposed redundancies. The quantization stage is used to reduce some other forms of redundancy in the data (for instance due to limitations of the human eye), at the expense of accuracy in the data representation. The decompression involves doing the reverse operations of decoding, inverse quantization and inverse transformation. The operations before and after the quantization stage are generally reversible and hence do not introduce any loss or artefacts in the compression. Quantization, however, is not reversible and thus introduces some error in the compression process. In effect, from the viewpoint of compression models, the major difference between lossless and lossy compression is the quantization stage: lossless compression does not involve any quantization. The quantization stage also accounts for the huge compression ratios often achievable in lossy data compression.

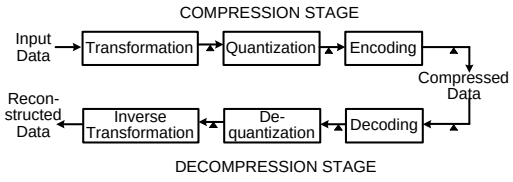

Figure 1: General compression model. Small markers show the points where compressed pattern matching can take place.

In the following sections we first look at methods that are primarily lossless, and then lossy image compression methods are surveyed.

2.1 Text compression

Text compression is generally lossless, but results in only a modest reduction in the data size. The general approach is to exploit different forms of redundancy — repetitions, correlations, etc. that may occur in the text. The methods can be grouped generally into three classes: dictionary methods (also called pattern substitution), statistical methods, and methods based on sorted contexts. These methods are discussed in detail below.

2.1.1 Dictionary methods

Currently the most widely used lossless compression methods are based on *Ziv-Lempel* coding, which represents a variety of methods from the general class known as *dictionary* coders. A dictionary method simply maintains some dictionary (also known as a codebook or lexicon), and data is compressed by replacing strings in the text with a reference to the string in the dictionary. The many variants of dictionary coding arise from different choices for how the dictionary is constructed, the algorithm used to search the dictionary, the rules for which substrings in the text are replaced with a reference, and what bit pattern will be used to represent the reference.

Static dictionary coders work with a pre-constructed dictionary. The problem of constructing an optimal dictionary is NP-complete (Storer and Syzmanski, 1982), although suitable heuristics are available. However, a more elegant solution to the problem is to construct the dictionary *adaptively*, and this is the main idea in Ziv-Lempel coding.

There are two main variants of Ziv-Lempel coding, those based on a 1977 paper (Ziv and Lempel, 1977) and those based on a 1978 paper (Ziv and Lempel, 1978), sometimes called LZ77 (or LZ1) and LZ78 (or LZ2) respectively.

LZ77 methods use the simple idea that text prior to the current coding point is the dictionary, so a greedy search is performed of recent text to see if the next few characters have occurred before, and they are then replaced with a reference that defines how far back the match is, and how long it is.

Limpel-Ziv coding often involves some form of factorization on the input text and the use of a suffix tree. With respect to the block diagram of Figure 1, the transformation stage could involve the initial construction of the suffix trees or the factorization of the input text to expose the repetitions in the text. The coding stage essentially involves the replacement of a factor that has appeared previously in the text with a pointer to where it appeared.

An LZ77 *decoder* is very fast; it maintains an array of recent text, and simply looks up the reference in the array and copies the characters. The *encoder* tends to be slower, as the previous text (usually just the last few thousand characters) must be searched for a match. This is usually done using a hash table to index the text, or sometimes a binary tree, linked list, or lookup table. Normally

LZ77 coders use a greedy algorithm and match the longest string that they find. The searching for encoding could be used to save work in compressed-domain matching by forcing the units being matched to be of relevance to later pattern matching searches. For example, the matches could be forced to start and/or finish on word boundaries to simplify word-based searches. This only affects the encoding algorithm; the same decoder could be used without modification, which has the nice property that such files could still be viewed and processed without any changes to the viewing software.

The LZ77 approach has been used by a number of popular compression methods, including the ZIP methods (a general purpose compression system used in products like PKZIP and GZIP), and the system has been adapted for an image graphics standard called PNG (*portable network graphics*), which is intended to replace the widely used GIF (*graphics interchange format*) standard. The better LZ77 methods use a form of Huffman coding (see Section 2.1.2) to encode the pointers in their output.

The other branch of the Ziv-Lempel family is the LZ78 type methods. These methods also base the dictionary on previously coded text, but the text is broken up into substrings according to a simple heuristic (usually by concatenating one character to a previously coded substring). This limits the range of strings available for substitution, but makes encoding simpler. LZ78 methods are waning in popularity because the LZ77 methods are now both faster and give better compression, and also there are patent issues surrounding the use of some LZ78 variants. However, it may be possible to adapt LZ78 methods for compressed-domain searching again by restricting compression pattern matches to correspond to the kind of match that will be required later. Also, the dictionary contains a lot of implicit information about patterns, and pointers to similar patterns, which could be exploited.

One of the best known variants of the LZ78 family is the *LZW* method (Welch, 1984), which developed into a popular public domain system called COMPRESS, which for many years was the de-facto standard for lossless compression. The LZW method is also the basis of the compression component of the GIF image standard, which are widely used on the World Wide Web, amongst other places.

There are also various other proposed variants of the LZ78 family, such as the LZMW (Miller and Wegman, 1984; Miller and Wegman, 1985), the LZAP (Storer, 1988), etc. (See (Salomon, 1998) for more variants of the LZ77 and LZ78 compression family). A hybrid approach between the LZ77 and LZ78 which has specific properties particularly tuned for searching on the compressed data has also been proposed (Navarro and Raffinot, 1999). The variants differ mainly in how they factorize the input text, how they reference the previous occurrence of the text segments, or how they limit the amount of memory that may be taken up by the dictionary.

More recently, a dictionary-related compression method based on *antidictionaries* has been proposed (Crochemore et al., 2000). Here, rather than using the words that appear as factors in the text (i.e. the dictionary) to provide compression, the words that did not appear (the antidictionary) are used. Compressed pattern matching for text compressed with antidictionaries have been

also been explored (Shibata et al., 1999b).

2.1.2 Statistical (symbolwise) compression methods

Statistical methods take a different approach to compression. Each symbol in the input (typically a character, byte, or pixel) is coded based on its probability of occurrence. According to Shannon's noiseless source coding theorem (Shannon, 1948; Shannon and Weaver, 1949), a symbol with probability p is optimally coded in $-\log_2 p$ bits.

The classic method for symbolwise coding is *Huffman* coding (Huffman, 1952), a method for optimally allocating short codes to more probable strings, and longer codes to the less probable. Huffman coding was regarded for a long time to be optimal, but this optimality only applies to codes that use a whole number of bits. In the 1980s a method called arithmetic coding (Langdon, 1984; Witten et al., 1987) was developed that obtained improved compression by "splitting the bit." Nevertheless, Huffman coding is still very important because in many situations it produces very good compression, with high compression speed.

Huffman and arithmetic coding provide a method for generating a representation for symbols given their probability distribution. However, the task of finding an appropriate probability distribution for a given text is something of an art, and is discussed below. The probability distribution is called a model, since it abstracts important features of the data. In a sense, the models used to determine/generate the probability distributions perform a kind of transformation, for instance transforming the input symbols into some simple frequency counts. Hence, for statistical methods, the transformation stage in Figure 1 captures the modelling part, while the encoding stage determines the appropriate codes for the data using the symbol probabilities. First we describe Huffman and arithmetic coding, which serve as the back-end for most symbolwise and lossy compression methods. Then we look at how models can be constructed to provide good probability distributions.

Huffman's method for coding is based on the binary tree. An example is given in Figure 2. This tree is for an alphabet of five characters, each with some estimated probability of occurrence. The codeword for a particular character is given by the path from the root to the corresponding leaf node. For example, an "a" is coded as 000, and an "e" as "11". The tree is generated using a simple algorithm: the two least probable leaves are paired to form an aggregate node which replaces the two leaves, and this pairing is applied until there is just one aggregate node, the root. This greedy algorithm is usually implemented using a heap to find the node with the smallest value.

The set of codewords thus forms a kind of vocabulary for the symbols in the text. Clearly, such a vocabulary can be exploited in performing compressed domain search on the text.

Other variants of the Huffman approach with special relevance to searching have recently appeared in the literature. The byte-oriented word-based Huffman coding scheme (Moura et al., 2000; Ziviani et al., 2000) uses words and

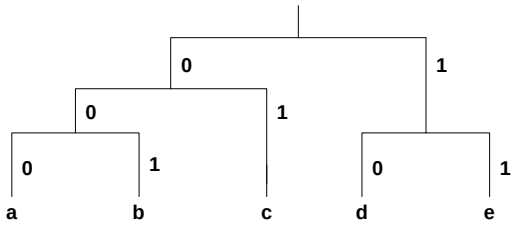

Figure 2: Huffman coding tree for five sumbols with symbol probabilities: $p(a) = 0.13, p(b) = 0.12, p(c) = 0.32, p(d) = 0.21, p(e) = 0.22$

text separators (rather than individual characters) as the basic symbols in the alphabet. Then a sequence of bytes(rather than the usual bits) are assigned to each word or separator. The *plain Huffman code* uses all the bits in the bytes, while the *tagged Huffman code* uses the highest order bit to tag the first byte of each codeword. The tagging was found to be useful in searching directly on the compressed data, because it provides random access. We also notice that the same symbol in the Huffman code could have different bit patterns assigned to it, as the assignment depends on how the tree is structured. This could be a problem for searching, unless we restrict the tree structure. In practice, a variation of the code is used called *canonical* Huffman codes (Hirschberg and Lelewer, 1990). These have the same length as the traditional Huffman code, but have their bit patterns chosen in a way that makes them very fast to encode and decode using a lookup table rather than a tree.

Arithmetic coding also represents input characters using bits relative to a given probability distribution. However, in arithmetic coding the representation of consecutive symbols may overlap. One analogy is that Huffman coding can be regarded as generating the output bit-stream by performing a SHIFT operation on the code (to take it to the end of the current output string) and then an OR operation (to add it to the bit-stream). In contrast, arithmetic coding uses multiplication to move the code across, and addition to incorporate it in the bit-stream. Thus one bit in the output may contain information about two different input symbols, and the boundary between two symbols is not likely to have a corresponding boundary between two output bits. The reader is referred to the literature for the details of how arithmetic coding works (Langdon, 1984; Witten et al., 1987; Bell et al., 1990).

With arithmetic coding, the same symbol can have quite different representations each time it is coded. Further, because of the possible overlap in the coding of two different symbols, decompression cannot normally start in the middle of the compressed file. This means that it will be difficult to provide random access to the compressed data. Therefore, arithmetic coding is generally viewed as not being very suitable for compressed-domain searching (Baeza-Yates and Ribeiro-Neto, 1999).

Arithmetic coding certainly offers quite a challenge, because a particular

pattern need not have a particular encoding in the output, and to interpret a particular part of the encoded file, all of the prior part of the file must be decoded first. It is possible for particularly unusual models (such as skew probability distributions) that the output will be predictable, but it is not clear that this will be of use in practice. This is an area that needs to be investigated.

We now turn to models that are used for symbolwise methods. These models provide the probability distribution that Huffman and arithmetic coders use to create a bit-stream.

One of the simplest models is one that simply counts symbol frequencies, and uses the relative frequency of a symbol to estimate its probability. For example, if the symbol "u" accounts for 4 out of 200 characters then we might estimate its probability to be 2%, and according to Shannon's entropy formula this should be coded in about $-\log_2 0.02 \approx 5.6$ bits. This kind of model is often used in conjunction with Huffman coding, and is particularly effective when the symbols being used are English words. For example, a word-level Huffman code is used in the *mg* full-text retrieval system (Witten et al., 1999), as it gives very good compression, and also the unit of coding (words) matches the unit that will be used for searching the text later.

The simple model just described can be improved by observing that the probability of a symbol is influenced by its context. For example, suppose whenever the letter "q" has occurred in a text that it is followed by the letter "u" 98 times out of 100. We can then estimate the probability of a "u" in this context to be 98%, which should be coded in about $-\log_2 0.98 \approx 0.03$ bits — considerably less that the 5.6 bits with the simple model. This is a particularly extreme example, but considerable gains can be made by using contextual information. This type of model is called a *finite context model*. The *order* of the model is the number of symbols used as context; the example above used an order-1 finite context model. Even better compression is achieved using higher order models, but if the context size is too large then too much memory is required to keep track of all the contexts, and also large contexts will not have occurred very many times, and so probability estimates will be based on unreliable samples (if any are available!)

The probabilities can either be estimated statically (by determining them in advance, possibly from the text to be encoded), or adaptively, by keeping count of the character frequencies during encoding, and basing probability estimates on the counts of previously coded characters. In an adaptive coding situation, the decoder must also keep track of the counts so that it is using the same model as the encoder. This is not difficult because the counts are taken only from data that has already been decoded. For adaptive models, in the limiting case, the probabilities they generate will asymptotically approach the ones generated by the static models.

The methods that achieve the best compression are based on finite context models. One of the most widely known is the "prediction by partial matching" (PPM) system (Cleary and Witten, 1984; Moffat, 1990), which uses a variety of context sizes. To encode each symbol PPM looks for the longest context available to code a character. Shorter contexts are needed early in the coding

when few contexts have been seen. Arithmetic coding is used to efficiently code the probabilities, which are normally quite low. The PPM system usually uses a trie-like data structure (Bell et al., 1990) to search for previous occurrences of the context, looking for the longest match possible. This structure must also be maintained by the decoder, which raises the possibility of using the structure for compressed-domain searching. The decoding process would still have to be performed in full (particularly because arithmetic coding is used), but the output need not be stored, and it may be useful to keep the search trie in memory after decoding to use as an index to perform multiple pattern matching operations on the text.

Another kind of model used for text compression is *Dynamic Markov Compression* (DMC), which uses a finite state automata to make probability estimates for symbols (Horspool and Cormack, 1986; Cormack and Horspool, 1987). Bell and Moffat (Bell and Moffat, 1989) have shown that these models are equivalent to finite context models, but are a lot simpler to implement (although they use a lot of resources at runtime). As with PPM, there may be some possibility of using the finite state machine data structure to search for patterns, possibly after a whole text has been decoded.

Klein (Klein, 2000) has suggested to improve the decompression time for static compression schemes by extending the symbol alphabet. The idea is to construct a *meta-alphabet*, which extends the standard alphabet by including frequent words, phrases, or word fragments. The main advantage here is the improved decompression time, even at the possible expense of longer compression time.

2.1.3 Text compression by sorted-contexts

Some compression methods make use of contexts, but do this by permuting the text so that characters that occur in similar contexts occur adjacent to each other. This general technique is called *block sorting*, or the *Burrows-Wheeler transform* (BWT) (Burrows and Wheeler, 1994; Fenwick, 1996a; Fenwick, 1996b). A large block of text is coded at a time. A list is constructed of every character and its (arbitrarily long) context, and then the list is sorted lexically by contexts. The characters are then transmitted in this sorted order. This permutation of the characters has the desirable property that characters in lexically similar contexts will be near each other, and this can be exploited by using a coder that assigns short codes to recently seen symbols. From the viewpoint of the general compression model of Figure 1, the permutation and subsequent sorting correspond to the transformation stage. Remarkably, the permuted stream of characters can be used to reconstruct the original stream. The inverse transformation is based on the observation that the decoder can sort the sequence of characters to get the last character of the sorted context. This provides enough information to reconstruct the order of the original characters.

The block sorted contexts in the encoder would be an excellent index for a pattern matching system, since only a binary search is required to locate a given pattern, and similar patterns will be adjacent. The decoder only has limited

information about the sorted context, but it may be possible to exploit this to perform an initial match on two symbols (a character and its context), and then decode only that part of the text to see if the pattern match continues.

Recently a related technique has become available, called ACB (Associative Coder of Buyanovsky) (Salomon, 1998; Buyanovsky, 1994). This method uses a cross between LZ77 (and LZ78) matching and the BWT sorted contexts. In the ACB implementation, the encoder is able to generate a "context file," which contains dictionary information that is used for coding. It may be possible to exploit this file for compressed-domain searching.

New results and new challenges in lossless compression can be found in a recent special issue of the *Proceedings of the IEEE*, edited by Storer (Storer, 2000).

2.2 Image compression

The methods described in the previous section are lossless. Although some lossless methods have been used for image compression (see a recent comparative study (Memon et al., 2000)), lossy methods are more often used for images. This eliminates storing the data needed to describe details in the image that may only be artifacts of the digitisation process, which most humans cannot even notice. Most image compression methods have both a lossless and a lossy mode. Some of the methods, such as run length encoding and predictive coding, could also be lossless. In this section we we first give a brief discussion on the nature of images, which illuminates their special characteristics that make compression possible. We then look at compression methods, beginning with methods that are mainly designed for low resolution monochrome images, where lossless compression is feasible.

2.2.1 Nature of images

An image of a surface can be viewed as a two-dimensional function of the light intensity falling on the surface. For a given spatial location on the surface, the image is formed from a combination of the illumination on the point, and the surface reflectance at that point. Practically, the range of values of the function is bounded within certain limits, called the *grey-scale*. The image formed in this way is called a *monochrome* (or *grey-scale*) image, as it considers only the intensity values. To convert to digital form, the image is digitized — both in its spatial co-ordinates (image sampling), and in its amplitude values (grey-level quantization). For most practical purposes, the digital image can thus be represented as a two-dimensional array of numbers. Each element in the array is called a picture element, or *pixel*.

Most natural objects come with different colours, not just as simple monochrome objects. Images are thus usually represented by use of certain colour models. The colour models used in image processing generally represent colour information in a three-dimensional co-ordinate system, whereby each colour can

be represented as a single point in the 3-space. Various models have been proposed, and are currently in use. The particular model to use is often dependent on the application. One such model is the RGB model, which represents the image with three independent image planes, one for each of the three primary colours — red (R), green (G), and blue (B). Based on these primary colours, all other colours can be generated. The colour or grey-level value at a given pixel position is also called the pixel value. Images where the pixels can take only two possible values (such as 0 or 1, black or white, etc.) are called bi-level or binary images.

For colour images, the YIQ representation is often used to represent the colour information before compression. Thus, the RGB pixel values have to be converted into the YIQ representation. The Y component (also called *luminance*), represents intensity information, while the I and Q (also called *chrominance*) components represent information about the colour. YIQ provides a better colour model for the human visual system (HVS), and also provides a partial decorrelation of the colour space. For lossy compression, more bits can be used for the Y channel, since humans are more sensitive to changes in intensity than those in colour.

As with text, compression for images also tries to store the image using a smaller space by eliminating some redundancies in the image. For instance, there is always the problem of *coding redundancy* due to the basic scheme used to describe the pixels, e.g. the usual colour, or grey-level representation. Since the image represents a physical surface, neighbouring points on the image are usually similar. Images therefore exhibit some level of spatial redundancy, which is correlation between nearby pixels. The image could also contain very fine details that a human is not likely to observe, due to the limitation of the human visual system. This is sometimes called psycho-visual redundancy (Gonzalez and Woods, 1992). The image itself might also exhibit a limited form of self-similarity. For instance, some part of an image could be similar to some other parts in some way. Image compression schemes aim to expose and then remove one or more of these types of redundancies.

For lossy compression, the reconstructed image is usually not exactly the same as the original image. Measures are then required to indicate the quality of the compression. Both qualitative and quantitative measures can be used. Qualitative measure are based on subjective human evaluation of the reconstructed image. Quantitative fidelity criteria used include the *mean square error* and the *signal to noise ratio*. The quantitative measures usually depend on some distance measure between the original image and the reconstructed image. The distance is usually calculated by varying the parameter of the general Minkowski distance.

Given two images, I_1 and I_2, the general Minkowski distance between them is given by: $d_n(I_1, I_2) = [\sum_{xy} [|I_1(x,y) - I_2(x,y)|]^n]^{\frac{1}{n}}$. With $n = 1$, we have the simple city block distance, $n = 2$ gives the Euclidean distance. The performance of a lossy compression algorithm is then evaluated in terms of its compression ratio and the quality of the compression — where quality could be measured based on both qualitative (subjective) observation and some quantitative fidelity

criteria.

2.2.2 Run length coding for images

Run-length coding (or encoding), RLE, simply involves replacing consecutive occurrences of the same symbol with a code that indicates what the symbol is, and how many times it has been repeated. Although this can be used to compress text, it produces better results when used on images. Image compression is achieved by exploiting the spatial redundancy by using run-length pairs, for instance, along a given path in the image. It is particularly suitable for monochrome images; for example, a scanned page tends to contain large runs of white pixels that can be represented as just one run.

A variant of run-length coding is used in the CCITT fax standards (Hunter and Robinson, 1980), which are currently widely used for fax machines. The runs are assumed to alternate between black and white, and the length of runs are encoded using a static Huffman code that was designed from some sample documents. Such codes can be used directly for compressed-domain searching because it is a lot faster to compare the length of runs than to compare the individual pixels in a run.

There are also two-dimensional extensions of the basic RLE scheme. For instance, for binary images (such as faxes), the relative address coding method (Gonzalez and Woods, 1992) tracks the binary transitions that begin and end each black or white runs. The distances between different types of transition on different rows are calculated and then coded using some variable length code. This often requires the adoption of a convention to determine the run values.

RLE is a lossless compression scheme. The output from RLE could further be compressed by passing it as the input to a variable-length coding scheme, such as Huffman coding. Most practical image compression schemes such as JPEG and MPEG use RLE (and other forms of Huffman coding) during the encoding stage.

Eilam and Vishkin (Eilam-Tzoreff and Vishkin, 1988) described the problem of pattern matching after various transformations on the input string. Initial ideas on pattern matching on RLE sequences were given in (Bunke and Csirik, 1993; Bunke and Csirik, 1995; Apostoloco et al., 1997). In (Adjeroh et al., 1999) it was observed that when video sequences are represented as strings, nearby symbols in the string are often the same or similar, resulting in long runs of repeated (or similar) symbols. To accommodate the symbol repetition in matching the video sequences, a special type of edit operation was defined which works on a representation similar to the RLE.

2.2.3 Predictive image coding

Correlation between pixels implies possible predictability of the pixel values using information from some other pixels. Spatial redundancy can be exploited by representing the image in terms of pixel differences, rather than explicit pixel values. Predictive lossless coding then codes only the new information using

some symbol-encoding method. Practically, the new information (the so-called *prediction error*) is the difference between the actual and the predicted value of the pixel. The major difference between the different methods of predictive coding is in how they determine the prediction error, for instance, the size of the neighbourhoods they consider, or the weights they assign to the neighbours, say based on their distance from the pixel under consideration.

The predictive techniques described in the section on lossless compression can also be adapted to images, although predicting the colour or gray-scale of a pixel is not so simple because we are dealing with a continuous image that has been digitised rather than discrete characters. For example, a gray level of 143 might be predicted when it is actually 142. Practical predictive techniques need to allow for such near misses.

The amount of compression achievable with this approach depends on how far the entropy of the original image can be reduced by using the pixel differences. Because there is usually a significant amount of inter-pixel redundancy in the image, this often leads to significant compression, with no loss. When lossy compression is acceptable, further compression can be achieved by quantizing the prediction error, before it is coded. This reduces the psycho-visual redundancy, but also reduces the accuracy of the representation, and hence introduces some error in the compression process. Generally, with more quantization, we achieve more compression, but also introduce more error. The amount of acceptable error is usually application dependent.

The prediction of bi-level images has been explored by several authors (Langdon and Rissanen, 1982; Moffat, 1991). Predictive compression is used in the JBIG standard, which is intended to replace the older facsimile compression method for bi-level images. JBIG is now an international standard, IS 11544 (ITU–T T.82) (CCITT, 1993). The predictions are coded using an arithmetic coder (Pennebaker et al., 1988a; Pennebaker and Mitchell, 1988), which does not bode well for compressed domain searching. However, as for PPM, it may be possible to exploit prediction information for searching. Also, JBIG includes a resolution-reduction method (Yoshida et al., 1992) that is used to transmit low quality versions of the image that can then be improved as more data is transmitted. This raises the possibility of searching the low resolution images for approximate matches, and then decoding the full image if it looks promising.

2.2.4 The quadtree representation

Under the quadtree representation (Samet, 1984), the image is scanned area by area to identify the areas that contain the same pixel values. It relies on the fact that natural images tend to contain large areas of uniform colour or grey-levels. A quadtree represents an image as a tree, whereby each node corresponds to some square portion of the image. The nodes in turn contain four quadrants of the square sub-image part. The input image (the root) is divided into four quadrants. Each quadrant becomes a child of the root. If the quadrant is homogeneous, i.e. all of its pixels are identical, the quadrant is represented as a single colour, and saved as a leaf node. If the quadrant is not homogeneous, the

quadrant is saved as a child node. Each child node is further decomposed into quadrants, and the process continues recursively until every part of the image has been coded, or a threshold (say in terms of the tree depth is reached).

The result is a tree structure, whereby a node is either a leaf node or contains exactly four children. The size of the quadtree depends on the complexity of the image. For images with large areas of identical pixels, compression is achieved since most of the quadrants will be represented by a single pixel value. However, for degenerate images (for instance, an image where each pixel has a different value), *expansion* rather than compression will result - this can be avoided by using a threshold on the height of the tree.

The quadtree partitioning could be suitable for compressed domain search, especially, if we can determine how to split the query image to correspond to the quadrants used for the compressed images. Its tree structure could also be exploited for fast searching and for progressive image transmission.

The quadtree (and its 3-D extension, the *octree*) is typically used for lossless data compression, for instance in medical imaging. For lossy compression, we can use an appropriate approximation to the quadrants. That is, homogeneity of the quadrants need not be based on equality of the pixel values. If no suitable approximation is found, the quadrant is partitioned further. The quadtree representation is related to the general ideas of vector quantization, run-length encoding, and constant area coding (Gonzalez and Woods, 1992). It has also been used in fractal-based coding (Fisher, 1995).

2.2.5 Block transform coding for images

The most widely used methods for colour and gray-scale image compression are based on block transform codes. Particular examples here are the JPEG standard (Wallace, 1991; Pennebaker and Mitchell, 1993) and the moving picture equivalent, MPEG (LeGall, 1991). Block transform codes divide up an image into blocks, typically 8 by 8 as used in JPEG, and code these by transforming them into a set of frequency domain coefficients, quantising the coefficients, and coding the quantized coefficients using Huffman or arithmetic coding. These steps correspond to the three basic stages of image compression as shown in Figure 1.

The transformation is typically a linear transform, and is usually reversible (ignoring round-off errors). The transformation itself does not produce any compression, but merely exposes the redundancies in the image, which are then exploited by subsequent stages of the compression. For most image blocks, the important information in the image will be packed into the first few coefficients, usually with larger magnitudes. The remaining coefficients (usually the majority) will have small values, which can be ignored, or quantized with little visual distortion in the reconstructed image.

Most lossy compression standards, such as JPEG and MPEG use the Discrete Cosine Transform (DCT), which is relatively fast and effective for images. Some other transforms such as the Fast Fourier Transform (FFT), Walsh-Hadamard Tranmsform (WHT), or Karhunen-Loéve Transform (KLT) are also used.

Various compression methods based on the block-transform have been proposed — see (Netravali and Limb, 1980; Jain, 1981; Gibson et al., 1998). The major differences are in the particular transformation they use, the way they quantize the coefficients (for instance, perceptually-adaptive schemes try to quantize based on the limitations of the human visual system), how the coefficients are chosen and traversed, and the bit allocation policy used to assign bits to the different coefficients.

Although block-transform coding is the predominant approach to lossy image compression, and most images are compressed with lossy schemes, transform coding poses a lot of difficulty for compressed domain searching. The first problem comes with the choice of block positions and block boundaries. The value of the transform domain coefficients depends strongly on both the spatial position of the pixels within the transform block, and the actual value of the pixels. Two images that are similar, but whose blocks are selected in a slightly different manner (for instance, one image is slightly rotated, or translated) could produce different coefficient values. Thus, there is the problem of matching two different blocks in the transform domain, since we cannot guarantee that the images would be perfectly registered before the transform blocks are chosen.

We have assumed in the above that the images are of the same size, or perhaps the same with only a small affine transformation. More difficult problems arise if the images are of different sizes, or if they are different images. This is one of the reasons why, unlike in text pattern-matching, exact matching is usually inappropriate for images, or for multimedia data in general. This is related to the two basic problems in image search and retrieval, which are whether we are to search for a given object or image block *within* another image, or whether we wish to compare the images on whole-picture basis (i.e., a small image block can be compared with a much bigger image block for similarity). The choice often depends on the application.

We can however perform approximate matching using the transform coefficients, by doing some further operations on the coefficients. In general, for image retrieval, searching is performed by use of image features extracted from the pixels, or from their transform coefficients, rather than the exact pixel values (or the exact coefficient values). For compressed domain search, the features are usually computed from the transform coefficients, based on the properties of the transformation used.

As with compressed domain text pattern matching, the reduced nature of the data in the compressed domain has also been exploited for image and video retrieval. For instance, the dc-image (an average image formed using only the dc coefficients of the DCT) has been used to speed up image searching in video sequences, and is the predominant method for fast image and video browsing (Wei et al., 1998; Ngo et al., 1998; Song and Yeo, 1999).

2.2.6 Vector quantization

Vector quantization (VQ) (Netravali and Haskell, 1988; Gersho and Gray, 1992) is based on the concept of compression by pattern substitution. Instead of

quantising individual pixels (scalar quantization), a VQ system creates a limited codebook containing *vectors*, that is, blocks of an image, and replaces each block with the *nearest* vector in the codebook. Nearness here is determined based on certain predefined distance or fidelity criteria. It also implies that an image block that does not appear in the codebook (a kind of vocabulary) will be replaced by its best approximation in the codebook. This also explains why vector quantization is a lossy compression scheme.

VQ decoding is very fast, since the decoder needs only to look up an entry in the codebook, but encoding can be very expensive because a suitable codebook must be determined, and then the best match for each block must be found in the codebook. It may be possible to exploit the searching required during VQ coding to achieve fast compressed-domain matching, since the codebook provides a list of blocks that can be matched with the search pattern, and then only those blocks need be extracted from the compressed domain.

Image indexing and retrieval on VQ-compressed images have been explored in (Idris and Panchanathan, 1997).

2.2.7 Fractal-based coding

Another form of redundancy that is often exploited in data compression is self-similarity. That is, the image could be similar to another image at a different scale. Natural images rarely exhibit complete self-similarity, in that, we may not find another image that is self-similar to some other. However, they often exhibit a limited form of self-similarity (different parts of an image could be quite similar, subject to some similarity threshold). Fractal based compression schemes (Fisher and Woods, 1992; Jacquin, 1992; Jacquin, 1993; Barnsley and Hurd, 1993; Fisher, 1995) generally aim to remove the redundancy which results from this limited self-similarity in natural images.

Fractal transformations (sometimes called iterated function systems) produce a new image by iteratively transforming (and reducing) a copy of an original image. The transformations used are called *contractive* (since they always shrink the image in some way). In the limit, after applying the transformations infinitely many times, the result of such an iterative process will converge to one final image, called an *attractor*. The new image therefore will have details at every scale, and hence is called a *fractal*. The transformations used are usually *affine transformations* (translation, rotation, scale, shearing, flipping and reflection), although any other contractive transform can be used.

Fractal coding is based on some intriguing and counter-intuitive properties of such interative function systems (Fisher, 1995; Salomon, 1998):

- The attractor is independent of the precise order in which the transformations are applied.
- The attractor is independent of the weights attached to each transformation. That is, the number of times each transformation is applied does not matter, so far as each transformation is applied at least once. (This, however, affects the time of convergence to the attractor).

- The final shape (i.e. the attractor) is independent of the shape of the original image to which the transformations are applied!

Therefore, the final image depends only on the particular transformations applied to the original object. A different set of transformations will produce a different result. Hence, for a given attractor, the required contractive transformation is unique.

In practice, it may be difficult to find the correct transformations for real images. Hence, the image is usually partitioned into non-overlapping parts called *ranges*, which can be of any size and shape. Partitioning can be done by simple square/rectangular blocks, using the quadtree, or some other methods, see (Fisher, 1995). For each range, the encoder searches for an image subpart called a *domain* that is the closest match to the range (subject to some distance/similarity threshold). If we know the range and the domain, the next problem is to find the set of transformations that can map the domain into the range. If we do this for all the ranges, we will have a set of transformations that encodes an approximation of the image. Fractal coding is therefore lossy. Also since we can only have an approximation (lossy representation) of the image, the matching of the domain and the range is based on some similarity threshold.

Now, rather than storing the actual image, we store a collection of transformations as the compressed stream. That is, for each range (and the domain that is most similar to it), we store the coefficients of the transformation required to make the domain become very close to the range. Compression is achieved since the number of bits required to store such coefficients will typically be less than those needed to store the image. (For affine transforms, we require only six coefficients for each transformation, although we might need to store some extra information, such as the positions of the range and the domain).

If we apply more transformation steps, we will obtain a closer match between the range and the domain, and thus less error in the compression. This will however lead to less compression, and also to more encoding time. In general, fractal coding is quite time consuming, since (in theory) we might need to perform an infinitely large number of transformations. In practice, we can set a threshold on the maximum number of steps — which directly affects the fidelity of the compression. Other ways to speed up the encoding is by classifying the domains and ranges based on certain criteria such as the density of edges. Domains in the same class as the range are expected to produce the best match for the range. Encoding time is therefore reduced by using only domains that are in the same class as the range when searching for matches.

Conceptually, compressed image matching should be possible with fractal-based image coding. Since the domain can be any arbitrary shape, given an input query image, we can apply a set of transformations on an arbitrary domain until we obtain an attractor that is close to the query pattern. Let us call the set that produced the attractor W. Since the tranformations will be unique for any given attractor, we can search for the query image without decompression by just checking if the set of transformations in W also appeared in the compressed stream.

With the use of domains and ranges, fractal image coding also has the potential of being used for object-level search for images in the compressed stream. Although initial ideas on compressed domain searching for fractal-based compression have been investigated (Zhang et al., 1995a; Zhang et al., 1995b), this is an area that still requires further study.

2.2.8 Subband and Wavelet-based coding

Transform coding is just one type of the general frequency-domain coding techniques. The general approach is to decompose the original image into different (spatial) frequency components, with the aim of reducing the correlation in the original data, and to pack the most important information into fewer coefficients to facilitate compression.

Subband decomposition and the wavelet transform are two other methods that decompose the image into different components. In subband coding (Woods, 1985; Gibson et al., 1998) the image is decomposed into different subbands using different digital filters or band-pass filters. The components in each band is then quantized and encoded differently. Usually, more bits are allocated to those bands that contain information that are more sensitive to human sensory perception.

Wavelet-based compression is a type of subband coding, in which the image is decomposed into different bands at different resolutions (DeVore et al., 1992; Topiwala, 1998). The idea is based on the pyramidal image representation (Burt and Adelson, 1983). At each resolution, the image is represented as some averages and differences, called *detail coefficients*. The differences provide information about important details needed for exact reconstruction of the original image. Typically, the wavelet transform will produce one average image at the top left corner, and smaller numbers, differences, or average of differences elsewhere.

Compression is achieved because the differences are generally smaller than the original pixel values. These can be compressed using any of the lossless compression schemes, such as RLE or Huffman coding. Also, there is usually pronounced correlation across subbands (Topiwala, 1998), and this can be exploited for further compression. Lossy compression (and hence higher compression) can be realized by quantizing some of such small differences to zero.

Wavelets, subband coding and fractals are emerging methods for data compression, but they generally require more computations than other methods. Also, their performance (for instance in terms of quality of reconstructed image) is not much better. However, they provide avenues for significantly higher compression ratios, which make them good candidates for archival applications, where searching and retrieval are important activities. As with transform coded streams, the average image can be used for a rough query and fast browsing of the database. More precise image searching on the compressed stream is still difficult.

2.2.9 Pattern matching image compression

Dictionary-based methods such as the LZ family exploit some repetition in the text, and have traditionally been used for lossless compression. Pattern matching image compression (PMIC) is a natural extension of the LZ approach to lossy compression. The basis is the idea of *approximate repetitiveness* (Luczak and Szpankowski, 1994; Alzina et al., 1999) which is detected by approximate pattern matching.

If a part of the already coded data re-occurs in an approximate sense, subsequent occurrences can be coded as a direct or indirect reference to the first occurrence. The approximate re-occurrence may or may not be continuous, and the nature of the re-occurrence may be different for different types of data (say images, and text). Hence different distortion measures may be required for different data types. The squared-error distortion measure (MSE) is often used for images.

(Constantinescu and Storer, 1994) proposed a lossy extension of the LZ78 algorithm for use with vector quantization. The concept of *waiting time* was used by (Steinberg and Gutman, 1993) to extend the Lempel Ziv algorithm to lossy compression. Here, the waiting time is modelled as the number of symbols before an approximate match to a string of a given length re-occurs for the first time in the text (or equivalently, the length of the shortest string that contains an approximate match to a string of a given length). Szpankowski and his colleagues focused on the approximate prefix analysis for the LZ family (Luczak and Szpankowski, 1994; Luczak and Szpankowski, 1997), and their use in image compression (Atallah et al., 1999; Alzina et al., 1999; Alzina et al., 2000).

The general approach to pattern matching image compression (PMIC) is as follows (Atallah et al., 1999):

1. Choose an appropriate distortion criteria (such as the squared error or the absolute error) to determine the match distance between strings;

2. Select some known part of the image as the database (for instance, the last few rows or subimages);

3. Search for the longest prefix in the uncompressed image that approximately matches a substring in the database (that is, the prefix should be within a pre-defined distance threshold from the matching substring in the database);

4. Instead of storing the entire prefix, store a pointer to the occurrence of a match, and the match difference.

The database and the uncompressed image can be considered either as a 1D sequence, (example, traversing the image in row-order), or as a sequence of 2D subimages. Thus, we could have one dimensional-PMIC or two-dimensional PMIC (based on 2D pattern matching). Since the LZ algorithms have been found suitable for compressed pattern matching, it is expected that image

searching could be performed directly on the lossy extensions, if we take cognizance of the distortion measures used during the compression stage.

It has been reported that, in terms of compression ratio and quality, the PMIC methods are generally comparable with transform-based methods (such JPEG), and methods based on wavelets. However, they require longer compression time, but much shorter decompression time (Atallah et al., 1999).

2.2.10 Progressive image transmission

Progressive image transmission (Anastassiou et al., 1983; Witten and Cleary, 1983; Sharman, 1992; Sharman et al., 1992) involves sending a low quality image followed by successively more detail, so the picture gradually becomes clearer in front of the user. This method is often used on the World Wide Web so that the user can cancel a page early in the transmission if it is not of interest.

Some image compression standards include options for progressive image transmission. For instance, with transform-based schemes (block-transform, subband, and wavelet decomposition schemes), progressive transmission can be achieved by first sending the first few coefficients (or frequency bands), and then transmitting information from more bands progressively. Another approach is to send the most significant bits from all the frequency coefficients first, and progressively send the remaining bits.

Progressive image transmission raises the possibility of searching a low resolution version of an image, and then focusing in on "interesting" parts of the image. For instance, if the coding scheme is based on the DCT, the dc components can be sent initially, which will be used to form the dc image, which can in turn be used to perform a fast appraisal or search on the image, before all the data components arrive.

We have described the currently popular methods for image and text compression. The major difference between text compression and image compression is that the former is lossless while the later is lossy. From a modelling point of view, this difference is primarily due to the incorporation of a quantization stage in lossy compression. The quantization stage directly affects both the amount of error introduced, and the amount of compression achieved. Although most lossy (image) compression methods incorporate some form of lossless (text) compression as a backend, some image compression methods are explicitly lossless (Memon et al., 2000). Other general ideas on image compression can be found in various published materials (Netravali and Limb, 1980; Jain, 1981; Gibson et al., 1998; Salomon, 1998; Cohn, 1994).

3 Search strategies

Searching for patterns is an important function in many applications, for both humans and machine. The *pattern searching problem* can be stated as follows: given a query string (the *pattern*), and a database string (the *text*), find one or all of the occurrences of the query in the database. The problem then is to

search the entire text for the requested pattern, producing a list of the positions in the text where a match starts (or ends). In this section, we describe the pattern matching problem, and the general methods that have been used to reduce the time required. We also briefly describe the image pattern-matching problem, which is more usually considered as a problem of image retrieval.

3.1 The pattern matching problem and its variants

Solving the pattern searching problem depends on a variant of the *string pattern matching problem*: given two strings, determine whether they are matches or not. Matches between strings are determined based on the distance between them.

The distance is traditionally calculated using the *string edit distance* (also called the Levenstein distance). Given two strings $A : a_1...a_u$, and $B : b_1...b_m$, over an alphabet Σ, and a set of allowed edit operations, the edit distance indicates the minimum number of edit operations required to transform one string into the other. Three basic types of edit operations are used: insertion of a symbol, $(\epsilon \rightarrow a)$; deletion of a symbol, $(a \rightarrow \epsilon)$; and substitution of one symbol with another $(a \rightarrow b)$; (ϵ represents the zero-length empty symbol, and $x \rightarrow y$ indicates that x is transformed into y). The edit operations could be assigned different costs, using suitable weighting functions. The edit distance is a generalization of the Hamming distance, which considers only strings of the same length, and allows only substitution operations. Computing the edit distance usually involves dynamic programming, and requires an $O(mu)$ computational time.

Given a text string A, and a pattern string B, the *exact string matching problem*, is to check for the existence of a substring of the text that is an exact replica of the pattern string. That is, the edit distance between the substring of A and the pattern should be zero. Exact pattern matching is an old problem, and various algorithms have been proposed (Wagner and Fischer, 1974; Boyer and Moore, 1977; Knuth et al., 1977; Sellers, 1980).

A variant of the pattern matching problem is the *k-difference problem* also called *approximate string matching.* The problem is to check if there exists a substring A_s of A, such that the edit distance between A_s and B is less than k. Another form of approximate matching, the *k-mismatch problem*, checks for a substring of A having only a maximum of k mismatches with B. That is, only the substitution operation is allowed. The parameter k thus acts as a form of threshold to determine the correctness of a match. As with the exact matching problem, different algorithms have also been proposed for the case of approximate matching (Ukkonen, 1985; Galil and Park, 1990; Chang and Lampe, 1992; Myers, 1994).

Other variants of the pattern matching problem have also been identified, usually for specific applications. Examples include

- pattern matching with swaps (Lee et al., 1997): a transposition of two symbols (or symbol blocks) in one of the strings is treated specially by

using different weights;

- pattern matching with fusion (Tsai and Yu, 1985; Adjeroh et al., 1999): consecutive symbols of the same character can be merged into one symbol, and one symbol can be split into different symbols of the same character;
- pattern matching with don't cares (Akutsu, 1994): a more general form of pattern matching in which wild-card characters can be allowed in both the text and the pattern. This is different from the usual approximate pattern matching in that now the don't care characters can occupy fixed positions in the pattern (or text) as opposed to the usual case where we have the flexibility of alignment to obtain a maximal match;
- matching with scaling (Amir et al., 1992): matching when a potential match is a scaled version of the pattern, similar to matching with fusion;
- multiple pattern matching (Idury, 1994; Amir and Calinescu, 1996): a generalization of the pattern matching problem, in which various patterns can be searched for in parallel, also called dictionary matching.
- super-pattern matching: finding a pattern of patterns (Knight and Myers, 1999).
- multidimensional pattern matching (Amir, 1992; Landau and Vishkin, 1994; Giancarlo and Gross, 1997): matching when the text and pattern are multidimensional — typically used for images (2D pattern matching) or video (2D/ 3D pattern matching).

3.1.1 Compressed pattern matching

In general, compressed pattern matching involves one or more of the above variants, with the constraint that, either the text, the pattern, or both are in compressed form (Amir and Benson, 1992; Karpinski et al., 1995). *Fully compressed pattern matching* is when the text and the pattern are both compressed, and matching involves no form of decompression.

There is a general problem that the format used to represent the compressed data is usually different from that of uncompressed data. More seriously, applying the same compression algorithm to two identical patterns that have different contexts could lead to completely different representations. That is, the same pattern located in two different text regions could result in different representations. Matching in such an environment will then have to consider the specific compression scheme used, and how the context could affect the compression. For lossy compression, the effect of the introduced error would also have to be considered, and once again, the error introduced could depend on the context.

3.1.2 Applications of pattern matching

Although pattern matching is sometimes pursued for its theoretical algorithmic significance, it also has applications in various real life problems. Traditional

areas where pattern matching has been used include simple spell checkers, comparing files and text segments, protein and DNA sequence alignments (Waterman, 1989; Rigoutsos and Califano, 1994; Chang and Lawler, 1994), automatic speech recognition (Sakoe and Chiba, 1978; Ney and Ortmanns, 2000), character recognition (Bunke and Sanfeliu, 1990), handprint recognition (Garris et al., 1997), shape analysis (Tsai and Yu, 1985), and general computer vision (Tsai and Yu, 1985; Bunke and Sanfeliu, 1990).

Recently, new applications of string pattern matching have been reported. Examples are in image and video compression (Alzina et al., 2000), audio compression (Alzina et al., 2000), video sequence analysis (Adjeroh et al., 1999), music sequence comparison (Mongeau and Sankoff, 1990), and music retrieval.

3.2 Search strategies for text

The näive pattern-matching algorithm runs in $O(mu)$. It generally ignores context information that could be obtained from the pattern, or from the text segment already matched. Most algorithms that provide significant improvement in the matching make use of such information, by finding some relationship between the symbols in the pattern and/or text.

Fast methods for string pattern matching is an area that has long been investigated, especially for exact pattern matching (Boyer and Moore, 1977; Knuth et al., 1977; Crochemore and et al, 1994). The methods can be broadly grouped as either *pre-indexing*, *pre-filtering*, or their combination. Pre-indexing (or preprocessing) usually involves the description of the database strings using a pre-defined index. The indices are typically generated by use of some hashing function or a scoring scheme (Myers, 1994; Rigoutsos and Califano, 1994; Chang and Lawler, 1994). Pre-filtering methods generally divide the matching problem into two stages: the filtering stage and the verification stage (Owolabi and McGregor, 1988; Wu and Manber, 1992b; Pevzner and Waterman, 1993; Chang and Lawler, 1994; Sutinen and Tarhio, 1996). In the first stage, an initial filtering is performed to select candidate regions of the database sequence that are likely to be matches to the query sequence. In the second stage, a detailed analysis is made on only the selected regions, to verify if they are actually matches.

The performance (in both efficiency and reliability of results) depends critically on the pre-filtering stage: if the filter is not effective in selecting only the text regions that are potentially similar to the pattern, the verification stage will end up comparing all parts of the text. Conversely, any region missed during the filtering stage can no longer be considered in the verification, and hence any false misses incurred at the first stage will be carried over to the final results. Pattern matching algorithms with sub-linear complexity have recently been reported (Myers, 1994; Chang and Lawler, 1994). They generally combine both pre-indexing and pre-filtering methods. For (Chang and Lawler, 1994), sub-linearity was defined in the sense of Boyer-Moore (Boyer and Moore, 1977): on average, fewer than u symbols are compared for a text of length u. That is, the matching time is in $O(u^p)$ for some $0 < p < 1$.

Equivalently, fast algorithms have also been proposed for approximate string matching. Ukkonen (Ukkonen, 1985) suggested the use of a cut-off, which avoids calculating portions of a column if the entries in the edit distance table can be inferred to be more than the required k-distance. Galil and Park (Galil and Park, 1990) proposed some methods based on the observation that the diagonal of the edit distance matrix is non decreasing, and that adjacent entries along the row or columns differ by at most one (when equal weights of unity are used for each edit operation). Chang and Lawler (Chang and Lawler, 1994) proposed the column partitioning of the matrix based on the matching statistics — that is, the longest local exact match. A general comparison of approximate pattern matching algorithms is presented in (Chang and Lampe, 1992).

In this section we discuss methods for pattern matching in uncompressed text. The methods used for approximate pattern matching generally make use of techniques for exact pattern matching. Furthermore, the various proposed fast algorithms for exact pattern matching can be traced to one or more of the three basic fast algorithms — KR, KMP, and BM algorithms. All three algorithms use some form of pre-processing. BM and KR also used pre-indexing and verification. We discuss the three algorithms in some more detail in this section.

3.2.1 Linear search

Although a simple linear search is often regarded as the least efficient method for searching, it has some interesting variants that can perform surprisingly well. In particular, if the access pattern to the text is known, then more frequently accessed records can be placed nearer the front, and if the probability distribution of access is skewed this can result in very efficient searching. "Self-adjusting" lists (Sleator and Tarjan, 1985) exploit this by using various heuristics to move items towards the front when they are used.

This idea can be extended to compressed-domain searching by observing that the order of the data in the compressed file might be permuted to put frequently accessed items towards the front. For example, in an image, areas that have a lot of detail might be more likely to be chosen. Savings can also be made by putting smaller items towards the front, if they are likely to cost less to make a comparison.

3.2.2 The Karp-Rabin Algorithm

The Karp-Rabin (KR) algorithm (Karp and Rabin, 1981) is based on the concept of hashing, by considering the equivalence of two numbers modulo another number. Given a pattern P, the m consecutive symbols of P are viewed as a length-m d-ary number, say P_d. Typically, d is the size of the alphabet, $d = |\Sigma|$. Similarly, m-length segments of the text T are also converted into the same d-ary number representation. Suppose the numeric representation of the $i-$th such segment is $T_d(i)$. Then we can conclude that the pattern occurs in the text if $P_d = T_d(i)$, for some i — that is, if the numeric representation for the pattern

is the same as that of some segment of the text.

The KR algorithm provides fast matching by pre-computing the representations for the pattern and the text segment. For the m-length pattern, this is done in $O(m)$ time. Interestingly, the representation for each of the $(u - m)$ possible m-length segments of the u-length text can also be computed in $O(u)$ total time, by using a recursive relationship between the representations for consecutive segments of the text. Hence, the algorithm takes $O(u + m)$ time to compute the representations, and another $O(u)$ time to find all occurrences of the pattern in the text.

A problem arises when the pattern is very long, whereby the corresponding representations could be very large numbers. The solution is to represent the numbers to a suitable modulus, usually chosen as a prime number. This may however lead to the possibility of two different numbers producing the same representation, leading to spurious matches. Hence, a verification stage is usually required for the KR algorithm. The chance of a spurious match can be made arbitrary small by choosing large values for the modulus. The time required for verification will usually be very small when compared to that of matching, and hence can be ignored. On average, the running time is $O(u + m)$, while the worst case is $O((u - m + 1)m)$.

The basic KR algorithm has been extended and has been used for 2-D pattern matching (Bird, 1977; Zhu and Takaoka, 1989).

3.2.3 The Knuth-Morris-Pratt Algorithm

The KMP algorithm (Knuth et al., 1977) simulates a pattern-matching automaton. It uses certain information gained by considering how the pattern matches against shifts of itself to determine which subsequent positions in the text can be skipped without missing out possible matches.

The information is pre-computed by use of a *prefix function*. In general, when the pattern is matched against a text segment, it is possible that a prefix of the pattern will match a corresponding prefix of the text. Suppose we denote such prefix of the pattern as P_p. The prefix function determines which prefix of the pattern P is a suffix of the matching prefix P_p. The prefix function is pre-computed from the m-length pattern in $O(m)$ time using an iterative enumeration of all the prefixes of $p_1 p_2 \ldots p_m$ that are also suffixes of $p_1 p_2 \ldots p_q$, for any $q, q = 1, 2, \ldots, m$.

By observing that a certain prefix of the pattern has already matched a segment of the text, the algorithm uses the prefix function to determine which further symbol comparisons will not result in a potential exact match for the pattern, and hence skips them. The average matching time is in $O(m + u)$.

The KMP algorithm is one of the more frequently cited pattern-matching algorithms. It has also been used for multidimensional pattern match (Baker, 1978) and for compressed domain matching (see Table 2).

3.2.4 The Boyer-Moore Algorithm

Like KMP, the BM algorithm matches the pattern and the text by skipping characters that are not likely to result in exact matching with the pattern. Like the KR algorithm, it also performs a pre-filtering of the text, and thus requires an $O(m)$ verification stage. Unlike the other methods, it compares the strings from right to left of the pattern.

At the heart of the algorithm are two matching heuristics – the *good-suffix heuristic* and the *bad-character heuristic*, based on which it can skip a large portion of the text. When a mismatch occurs, each heuristic proposes a number of characters that should be skipped at the next matching step, such that a possibly matching segment of the text will not be missed.

The match is performed by sliding the pattern over the text, and by comparing the characters right to left, starting with the last character in the pattern. When a mismatch is found, the mismatching character in the text is called the "bad character". The part of the text that has so far matched some suffix of the pattern is called the "good suffix". The bad-character heuristic proposes to move the pattern to the right, by an amount that guarantees that the bad character in the text will match the rightmost occurrence of the bad character in the pattern. Therefore, if the bad character does not occur in the pattern, the pattern may be moved completely past the bad character in the text. The good-suffix heuristic proposes to move the pattern to the right, by the minimum amount that guarantees that some pattern characters will match the good suffix characters previously found in the text. The BM algorithm then takes the larger of the two proposals.

It is possible that the bad-character heuristic might propose a negative shift (i.e. moving back to the already matched text area). However, the good-suffix heuristic always proposes a positive number, guaranteeing progress in the matching.

The bad-character heuristic requires $O(m + |\Sigma|)$ time units while the good-suffix heuristic requires $O(m)$. The BM algorithm has a worst case running time of $O((u - m + 1)m + |\Sigma|)$. The average running time is typically $\leq O(u + m)$. Overall, the BM algorithm generally produces better performance than the KMP and the KR algorithms for long patterns (large m), and relatively large alphabet sizes. See (Hume and Sunday, 1991; Crochemore and et al, 1994) for new improvements on the BM algorithm.

3.2.5 Bit-parallel Algorithms

The SHIFT-OR and SHIFT-AND algorithms (Baeza-Yates and Gonnet, 1992) are another family of algorithms that have been proposed to improve the efficiency of string pattern matching. These produce speed-ups by exploiting the parallelism in the bit level representation of the characters in the symbol alphabet. The bit-parallel algorithms have also been used in compressed pattern matching (Navarro and Raffinot, 1999; Kida et al., 1999; Moura et al., 2000). There are also methods based on automata theory (Cormen et al., 1990). Navarro and

Raffinot (Navarro and Raffinot, 2000) proposed methods that combine suffix automata and bit-parallel algorithms.

Various other algorithms have also been proposed for both exact and approximate pattern matching, most of them being some modification or combination of the above methods (Landau and Vishkin, 1988; Wu and Manber, 1992a; Wu and Manber, 1992b; Amir et al., 1992; Takaoka, 1994; Takaoka, 1996). A more recent survey by Hume and Sunday (Hume and Sunday, 1991) describes a more efficient variant of the Boyer-Moore method. (Crochemore and LeCroq, 1996) gives a brief overview of pattern-matching methods, including the BM and KMP algorithms. The paper also discusses text compression, but not the relationship between the compression and pattern matching.

The basic pattern matching algorithms have been extended to two dimensional pattern matching (Bird, 1977), which was improved by (Zhu and Takaoka, 1989). Baker (Baker, 1978) applied string matching algorithms to character arrays. The algorithms also represent the primary building blocks for compressed domain pattern matching, see Table 2.

3.3 Search strategies for images

Recognizing patterns in static or moving images is important for many applications, including tracking moving objects, character recognition, and face recognition. When exact matching is needed, it is possible to define simple functions or statistics on the image, based on which the matching can be performed. This could be useful when the results of image matching is to be used by a machine — for instance for further processing (such as in robot navigation in computer vision).

In practice, however, the result of image matching will be used by humans, which means that human subjectivity will have to be considered. Image matching is therefore typically based on *similarity* rather than exact matches. Issues such as distance or similarity measures between images thus become important. The metrics used are generally similar to those used to determine the quantitative fidelity for compressed images (see section 2.2.1). Other problems include the large amount of data often involved, and the huge computation that is generally needed.

For most applications that require image matching, approximate rather than exact matching is all that is needed. Image matching is therefore often performed in terms of similarity matching. The image similarity-matching problem is similar to the pattern-matching problem. Given an image database D_I and a target query image Q, the similarity matching problem is to report all images in D_I that are similar to Q (i.e. all images with a similarity distance not exceeding a given threshold). The matching is on *whole-image* basis and the retrieved images are then ranked according to their similarity to the target image. When the threshold is zero, we have exact matching. There is also a variant of this problem, *object-level image search*: given a target image Q, search *within* each image in D_I and report all subimages that match Q. In this case, the searching requires a more careful description of object shapes or contours,

and also matching within each image in the database. This distinction is often important, as it affects the time required for the match, the method to be used in the search, and also the quality of the results. Below, we briefly describe the general methods that are used in image matching.

3.3.1 Template matching

Template matching (Pratt, 1991; Gonzalez and Woods, 1992), is the simplest approach to image matching. It does a point by point comparison of the pixel positions in the image. The target image is used as a template. After choosing a suitable starting point, corresponding positions on the template and the database image are compared point by point. The overall difference is then added, to determine the match distance between the images.

Choosing a starting point requires the difficult problem of establishing a correspondence between a point in the target image and the same (or similar) point in the database image. Because of this difficulty, template matching often involves an exhaustive consideration of each pixel position in the target image as a candidate starting point. The match distance is then chosen as the minimum obtained from all the starting points.

To reduce the time required for template matching, some methods partition the images into subimages, and the comparison is performed block by block, rather than point by point. Also, since approximate matching is often adequate, most algorithms for image matching only perform approximate matching. These use image statistics, such as average colour or the variance of the pixel values to estimate the similarity between images. Others use a feature-based approach, in which important features in the image are extracted, based on which different images (or subimages) are matched. The features are usually based on shape, colour, texture, or spatial information in the image. The feature-based approach is the primary method used in image retrieval (Pentland et al., 1996; Smeulders et al., 2000).

To further reduce the time required, some methods try to detect some important areas or points in the image (interest points), and use these to speed up the matching. For instance, this could be done by matching only at the interest points or regions, or by using them to prune the search for candidate matches.

3.3.2 Other search strategies for images

Conceptually, methods for 2D pattern matching could be applied to the problem of image matching. For instance, methods for 2D approximate pattern matching can be well suited for comparing images. One problem is often the huge alphabet size that may be involved. For images, the alphabet size is typically the number of colour levels in the image (usually 256 or even millions of colour levels).

Other techniques for compressed domain image matching have also been explored. The basic idea is to define some functions using the transform coefficients, based on which two images (or image subparts) can be compared for approximate matching. The matching is often in terms of the shape, texture,

or general intensity characteristics in the images. Generally, approximate image matching is performed as a basic process in image retrieval. Methods for feature-based image retrieval have been reported for DCT-based compression (Wei et al., 1998; Ngo et al., 1998), VQ-based methods (Idris and Panchanathan, 1997), fractal-coding (Zhang et al., 1995b), and wavelets-based compression (Mandal et al., 1999b). A recent comprehensive survey on image retrieval can be found in (Smeulders et al., 2000).

4 Relationship between searching and compression

As was seen in the previous sections, there is a strong relationship between searching and compression. Some authors have considered compression as basically a pattern matching problem (Luczak and Szpankowski, 1997; Atallah et al., 1999; Alzina et al., 2000). More generally, most compression methods require some sort of searching:

- The Ziv-Lempel methods search the previously coded text for matches.
- PPM methods search for previous occurrences of a context using a trie data structure to predict what will happen in the current one.
- DMC uses a finite state machine to establish a context that turns out to have a similar meaning to the PPM context (Bell and Moffat, 1989). This is akin to algorithms such as Boyer-Moore constructing a machine to accelerate a search.
- VQ must search the codebook for the nearest match to the pattern being coded.
- PMIC has to search for approximate repetition on a prefix of the uncompressed image in the compressed part of the image. Here, matches are defined only in an approximate sense based on a specific distortion criterion.
- MPEG requires searching as part of its motion estimation and motion compensation — the key aspects of the MPEG standard, as they affect both the compression ratio and compression time. Motion estimation requires a fast method to determine the motion vectors, and always involves searching for the matching blocks withing a spatio-temporal neighbourhood. While the quality of the compression improves with more search area, the compression time increases.

In (Khan and Fatmi, 1993), data compression was viewed as a pattern recognition problem. Explicit considerations on the data structures used in searching as a way of improving the compression performance have been considered in (Bell and Kulp, 1993; Szpankowski, 1993; Constantinescu and Storer, 1994)

The relationship between pattern matching and compression for images have been studied in (Atallah et al., 1999; Alzina et al., 2000). More theoretical studies on optimal and suboptimal data compression with respect to pattern matching can be found in (Steinberg and Gutman, 1993; Szpankowski, 1993; Yang and Keiffer, 1996; Luczak and Szpankowski, 1997). A comparative study of pattern-matching image compression algorithms is presented in (Yang and Kieffer, 1995)

In general, for both lossy and lossless coding, more extensive searching often results in more compression, but with correspondingly more compression time. For lossy compression, more search usually leads to less error in the compression (i.e. better quality in the reconstructed image). There is thus a trade-off between the extent of the search and the compression time.

More importantly, the different searching activities may be exploited later for compressed-domain pattern matching. In principle we need only code the pattern to be located, and then search for the compressed pattern in the compressed data. However, because coding can depend on the context of the item being coded, this näive approach will not work. Furthermore, we may be looking for an approximate match, and two patterns that are similar may not appear to be similar in the compressed domain. A solution could be to constrain the compression so that overlaps between contexts are suitable for matching. An example here is the tagged Huffman coding used in (Moura et al., 2000).

Conversely, it is also possible to use compression for pattern matching. In one study (Johansen, 1994), the matches used by an LZ coder are used to infer which class an object in the image belongs to. Most image matching methods use (pre)classification to reduce the size of the required search space.

In (Shibata et al., 1999a), byte-pair encoding (BPE) was proposed as a compression mechanism that is specially tuned to speed up pattern matching, while in (Manber, 1997; Shibata et al., 2000) compression was used primarily for the purpose of improving search time, without necessarily considering the compressio ratio or the compression/decompression time. In (Inglis and Witten, 1994) a compression method is even used as the basis for searching, and the results of the search are in turn used as matches for a textual image compression system. The amount of compression achieved is used to determine whether the target data fits the model (which was trained on a template or another object of interest). This is akin to the work in (Maa, 1993), where bar codes in an image are recognised because of the way they compress. However, in the case of the bar codes, the pattern being recognised just happens to induce observable behaviour in a general purpose compressor.

In general, for a compression scheme to be suitable for compressed pattern matching, the scheme may need to provide random access to different points in the compressed data (this may require splitting the data into blocks and coding blocks of data at a time), a dictionary or vocabulary of the codewords, and a fixed code assignment for the encoded data stream. In Figure 1, we have used markers to show the several points where a compressed domain search can take place in the compression-decompression process.

5 Searching compressed data: performance measurement

To discuss performance issues in compressed pattern matching, it is appropriate to consider performance from the viewpoint of the two main aspects of the problem, namely, compression and pattern matching. Compression algorithms are usually evaluated in terms of the compression ratio they can achieve, coding complexity, decoding complexity, and the extra space required during the compression.

5.1 Performance measures for compression algorithms

The compression ratio measures how good the compression algorithm is in reducing the size of the given input data. It is defined as the ratio of the compressed data size to the original data size; more specifically, the number of bits needed to store the compressed data to the number of bits needed to store the original uncompressed data. Since the traditional objective of most compression algorithms is to reduce the space required to store the original data, the compression ratio is a very important measure of compression performance.

Depending on the type of modeling or transformation used by the algorithm, the compression ratio may not necessarily vary linearly with the original data size. For instance, with Huffman coding, the size of the vocabulary grows slower than the file size, since after some point, most of the codewords that are encountered would have appeared earlier in the text. Hence the amount of compression increases with the file size. This is exemplified in Figure 3, taken from (Moura et al., 2000). On the other hand, with LZ compressed text, the compression ratio is practically independent of the original data size, since these typically search only within a small window for repeated words. Without the restriction on the window size, the compression will improve with data size, but at the expense of higher coding complexity.

The coding and decoding complexity describe the amount of computation required at the coding or decoding stage, as the case may be. Practically, these relate to the speed of compression and decompression respectively. Typically, more coding complexity will be due to more extensive search during the compression (for instance, larger search windows for LZ compression), or other kinds of computation-intensive operations (such as lexicographic sort in BWT). More computation and hence higher coding complexity usually leads to better compression ratios.

Some algorithms provide symmetric complexity, in that, both the coding and decoding complexity are the same. This is often important in *on-line* algorithms, such as those used in modems, where the current data stream must be processed (compressed or decompressed) before the next data stream arrives. For some other applications, for example images in a web page, where data could be compressed just once, but might be decompressed many times, the symmetry may not be required and *off-line* algorithms may be used. Here lower

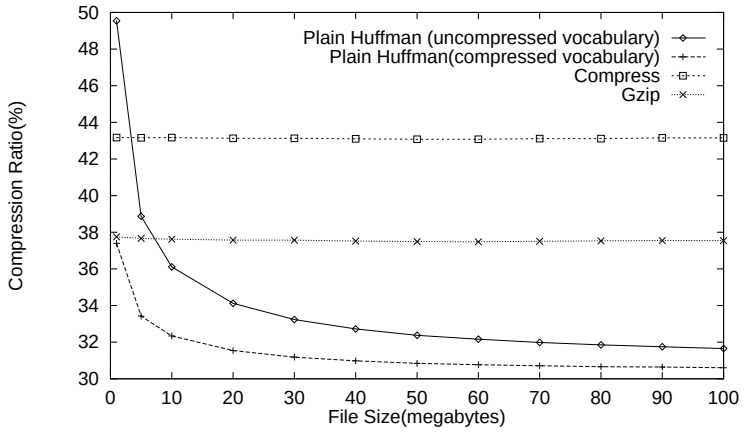

Figure 3: Variation of compression ratio with input file size for diferent compression algorithms. Figure taken from (Moura et al., 2000).

decompression times might be more beneficial, even with a higher compression time.

The extra space criterion (Amir et al., 1996) indicates the space-complexity of the algorithm. It shows how much memory space the algorithm will require to store some temporary data during compression. For instance, this may be used to store the Huffman tables, and will typically increase with increasing data size. The block size for BWT and the size of the priming text in PPMD can be related to the space complexity, but not necessarily in the sense of extra-space as used in (Amir et al., 1996).

For lossy compression, an extra performance measure is the quality of the reconstructed data. The quality is measured both qualitatively and quantitatively. Qualitatively, subjective perceptual criteria are used, for instance, by empirically comparing the reconstructed data with the original data using a human observer. Quantitative performance measures are based on different fidelity criteria, such as the mean square error or the signal-to-noise ratio. In general, higher compression ratios often result in lower compression quality.

More recently, with the increasing need to access data in its compressed form without decompression, the capability for random access to the compressed data and the ability to support search have emerged as new criteria for evaluating compression techniques. Table 1 (expanded from (Baeza-Yates and Ribeiro-Neto, 1999)) shows a comparison of general lossless compression methods.

5.2 Performance measurement for pattern matching algorithms

For traditional (text) pattern matching, the major performance measure is the complexity of the matching algorithm — in terms of both time and space. In

Performance Measure	Arithmetic coding	Character Huffman	Word Huffman	Ziv-Lempel	PPM	BWT	DMC
compression ratio	good	poor	good	good	very good	good	good
compression speed	slow	fast	fast	very fast	very slow	very fast	very slow
decompression speed	slow	fast	very fast	very fast	very slow	very fast	very slow
memory space	low	low	high	moderate	high	high	high
compressed pat. matching	no	yes	yes	yes	yes	yes	yes
random access	no	yes	yes	no	no	no	no

Table 1: Comparison of compression methods. Expanded from (Baeza-Yates and Ribeiro-Neto, 1999), pp. 187. Note: for compression ratio, c.r., very good: $c.r. < 0.30$; good: $0.30 < c.r. < 0.45$; poor: $c.r. > 0.45$.

general, with u as the text size and m as the pattern size, the worst case complexity is in $O(um)$, while some linear algorithms that run in $O(u + m)$ are available. Algorithms with sub-linear complexity have also been reported (Myers, 1994; Chang and Lawler, 1994). In practice, the complexity directly affects the system response time to user queries.

With image retrieval, the pattern matching is necessarily approximate. However, the approximation required here cannot easily be described in terms of the usual k-distance or k-mismatches. Other measures of effectiveness, such as precision, recall and ranking are therefore needed. The precision shows how good the algorithm is in retrieving *only* the correct (or similar) matches, while recall indicates how well the algorithm can retrieve *all* correct matches. Once again, for certain data types, such as images, what is correct or similar could be quite subjective, and will depend on the application (Smeulders et al., 2000). For conventional exact pattern matching algorithms in text, perfect precision and recall are usual. However, under compressed pattern matching, however, the context of the pattern in the text could lead to possible mis-detection of the pattern in the text. Precision and recall could therefore be relevant in compressed pattern matching for both lossy and lossless compression.

5.3 Performance measures for compressed pattern matching

Based on the foregoing, we can enumerate some important points that can be considered in measuring the performance of compressed domain pattern matching algorithms:

- *Complexity and speed*: The time complexity is related to the speed of the algorithm and shows how fast the algorithm could be. The complexity here is only the theoretical complexity, but should reflect the speed of the algorithm in practice.

- *Extra space*: Like the time complexity, the criterion of extra space shows the theoretical space complexity of the algorithm. It could also have a

bearing on the amount of resources the algorithm could require, especially for huge-volume applications, when the text or even the pattern could be quite large. See Table 2 (Section 6) for the theoretical performance (in terms of time and space complexity) of various proposed text-based compressed-pattern matching algorithms.

- *Optimality*: In (Amir et al., 1996; Amir et al., 1997), the concept of optimality was introduced to the problem of compressed domain pattern matching. Generally, compressed-domain searching is regarded as optimal if the factor by which the search is sped up is no less than the factor by which the text has been compressed (i.e. the inverse of the compression ratio). (Schmalz, 1992) provides a discussion on how the speedup can be measured. One question that arises is if the speed up can be better than the compression ratio.

- *Comparison with decompress-then-search*: Another way the performance can be measured is by considering how fast the compressed pattern matching is, as compared with performing the same searching operation on the uncompressed data using existing text matching algorithms — the so-called decompress-then-search technique. In (Navarro and Raffinot, 1999), it was argued that, in practice, compressed pattern matching cannot be faster than uncompressed pattern matching for some compression methods, such as LZ77. A stricter criteria would be to remove the decompression time, and compare how fast the compressed-pattern matching algorithm could be when compared with the fastest traditional (i.e. uncompressed) pattern matching algorithm. If we can achieve fast searching by using compressed domain methods for any compression scheme, one can then take advantage of this to compress the data with the primary objective of improving search time. (Shibata et al., 2000) suggest using compression for this type of purpose. These issues are related to the issue of optimality. In general, since a compressed pattern matching algorithm will typically consider a smaller amount of data, using an optimal compressed pattern-matching algorithm should be faster than doing the same search on uncompressed data. But this may not always be the case in practice.

- *Precision and recall*: This will be important for searching on data that is compressed using lossy-compression methods. Since the retrieved results may no longer be exact matches to the original pattern, we will require a way of knowing the effectiveness of the matching. Taken together, precision and recall can provide us with an objective way to evaluate a searching algorithm. We note that this is not a problem in traditional k-approximate pattern matching used in text retrieval, because the effectiveness of the match is implicitly defined by the distance parameter k. With compressed pattern matching, however, depending on the compression method and the context of a given pattern in the text, it is possible that a search algorithm could miss out an occurrence of a query pattern in the text — even for

exact matching. Therefore, precision and recall will also be relevant in considering compressed pattern matching for text.

- *Ranking*: Once again for lossy-compression, this is one other way of evaluating the effectiveness of compressed domain pattern matching. The precision and recall usually do not provide adequate information on the effectiveness of the retrieval, especially for perceptual data, such as multimedia data (images, video and audio). The ranking measure will show how well the ranking of the results produced by the algorithm compares with those produced by a non-compressed domain matching algorithm and/or with the ranking produced by a human observer.

- *Others*: There could also be other ways to check the performace. For instance, how the method performs in other related activities, such as data mining, which involves a series of pattern matching operations.

6 Searching Compressed Data: Lossless Compression

The compressed domain pattern matching problem is to find the occurrence(s) of a given pattern in the compressed text without decompression. Lossless compression algorithms can typically compress files to within the range of 50% to 20% of the original uncompressed file size, depending on which algorithm is being used and the type of text. Theoretically, for reasonably large files, the relative size of the compressed file should usually decrease with increasing input file size (see Figure 3). In practice, for real text files, the size of the compressed file is almost linearly proportional to the original size. Thus, a linear pattern matching algorithm on the compressed file with a large proportionality constant could actually perform worse than searching on the original file (Kida et al., 1999; Navarro and Raffinot, 1999). We have noted earlier that pattern matching can be regarded as the basis of compression. The process can be reversed, in the sense that the compression algorithms can be designed to aid pattern search. Manber (Manber, 1997) proposed such an algorithm for a class of files that are searched often such as catalogs, bibliographic files and address books. The basic idea was to substitute common bigrams of characters with special symbols that can still be encoded in one byte. The scheme allows the basic pattern matching algorithms to be used for fast pattern search, although the reduction of the file size was only to about 30% of the original size. Shibata et al (Shibata et al., 1999b) used a similar approach and used a faster Boyer-Moore algorithm for pattern search. For the dictionary compression method LZ77, the algorithm can be designed to recognize certain patterns that will be searched in the compressed data later. If one is interested in performing word-based search in the compressed domain, the LZ77 algorithm could be made to

put words or phrases in the dictionary that start or finish in word boundaries. A similar approach will be applicable to LZ78 or LZW algorithms which deal with patterns indirectly via pointers.

Compressed domain pattern matching algorithms without any constraint on the patterns have also been proposed based on LZ families. (Amir et al., 1996) described a method for searching LZW coded files which works in time and space $O(m^2 + n)$, where m is the size of the uncompressed pattern, and n is the size of the compressed text. The algorithm uses a special data structure called a dictionary trie which implicitly decodes the compressed text without producing the output symbols. Farach and Thorup (Farach and Thorup, 1995) described a randomized algorithm to determine whether a pattern is present or not in LZ77 compressed text in time $O(m + nlog^2(u/n))$, where u is the length of the uncompressed text. See also (Kosaraju, 1995). (Barcaccia et al., 1998) extended the work of (Amir et al., 1996) to an LZ compression method that uses the so-called ID heuristic (Miller and Wegman, 1985). The ID heuristic is also known as LZMW. This heuristic grows the phrases in the dictionary by concatenating pairs of adjacently parsed phrases, rather than just adding one character to an existing phrase. The search method is able to exploit these large components to keep track of whether or not they contain the target pattern. Their algorithm requires $O(m + t)$ space, where m is the pattern size and t is the maximum target length. This is essentially optimal. However, the search time is $O(n(m + t))$, where n is the size of the compressed file. This is not as good as the "optimal" time established by Amir and Benson (Amir et al., 1997), which is $O(n + m)$. Navarro and Raffinot (Navarro and Raffinot, 1999) proposed a hybrid compression scheme between LZ77 and LZ78 which can be searched in $O(min(u, nlogm) + r)$ average time, where r is the total number of matches. (Karpinski et al., 1995) considered fully compressed pattern matching for LZ coded data, where the search pattern is also compressed, and neither are decompressed during searching. Recently, a dictionary-related compression method based on anti-dictionary (the words that do not appear in the text) has been proposed (Crochemore et al., 2000). (Shibata et al., 1999b) used this idea to develop an algorithm that preprocesses a pattern of length m and an anti-dictionary of size a in $O(m^2 + a)$ time and then determines all occurrences of the pattern by a linear scan of the compressed text of length n.

The LZ family of algorithms have also been used in the context of PMIC for approximate pattern matching for lossy compression (See section 2.2.9). Manber (Manber, 1997) has described a compression system that allows for approximate pattern matching in the presence of errors, although the compression is lossless.

(Mukherjee and Acharya, 1994) described techniques for searching Huffman compressed files. A simple search of the Huffman coded file to find a compressed pattern using a fast search algorithm such as KMP will not produce correct results. Consider the following Huffman codes: $a = 0, b = 10, c = 110$ and $d = 111$. If the text is $T = abbacdabca$, the compressed text will be $c(T) = 0101001101110101100$. If the pattern is $P = ab$, the compressed pattern is $c(P) = 010$. If we now do a pattern matching of the compressed pattern against the compressed text, we will get three matches of which the second match

beginning at the third bit position from left is a false match. Furthermore, if the pattern is of the form $P = aXbYc.$ where X is a wild-card character and Y is a variable length wild-card character (that is, any sequence of characters of finite length), the representation of P becomes ambiguous in compressed form and the pattern matching algorithms (Fischer and Paterson, 1974) fail to work.

The basic idea of Mukherjee's algorithm is to use a data structure that will determine the byte boundaries in the variable length coded compressed text to initiate pattern search with respect to the compressed pattern. Related VLSI algorithms have also been published (Mukherjee and Acharya, 1995). Their method raises the possibility of searching for part of a variable length compressed string, even if the compressed file is only searched on byte boundaries. This is achieved by searching for all variations of the search strings generated by starting at different points in the string. Only eight starting points need to be considered to cover every possible way the coded string could cross a byte boundary. It seems that this idea could also be applicable to the Burrows-Wheeler transform (BWT) (Burrows and Wheeler, 1994), as part of BWT involves sorting the data according to context. Mukherjee and Acharya's method (Mukherjee and Acharya, 1994) can also be extended to handle patterns with fixed or variable length wild card characters and to search data that has been compressed with an adaptive Huffman code.

(Moura et al., 2000; Ziviani et al., 2000) proposed a semi-static word-based modeling scheme for Huffman coded compressed text files which can be searched directly at word boundaries using any fast sequential pattern search algorithm. The coding alphabet is byte oriented and not bit oriented. The first bit of the byte is used to mark the beginning of a word. The authors report a factor of 2 improvement in the search time of the compressed file in comparison to searching using the uncompressed files, and a 33% compression ratio for the TREC 3 collection (TREC, 2000).

Compressed domain pattern matching with arithmetic coding (in its standard formulation) is not possible. This can be proved by the following counter example: Let $\Sigma = \{a, b, c, d\}$ and assume that the probabilities of these four characters are 0.5, 0.25, 0.125 and 0.125 respectively. Also assume that the precision of the arithmetic code is up to 4 decimal digits. Given the text $T = ababacabdbaabbacabacac$ and pattern $P = baca$, the compressed text could be represented by $c(T) = [0.2874, 0.9425, 0.7838]$ where 0.2874, 0.9425 and 0.7838 are the decimal form of the arithmetic codes of the substrings *ababacab*, *dbaabba* and *cabacac* respectively (concatenation of these substrings is the text string T), using the above set of probabilities. The compressed pattern is $c(P) = [0.5950]$, using the same statistics for the characters. Given the above $c(T)$ and $c(P)$, we cannot determine the occurrence of the above pattern P in the text T, although the pattern occured at three positions of the text beginning at the positions 4, 14, 18 respectively.

The PPM compression methods use a trie-like data structure to search for the longest context in the portion of the text already encoded. This structure is also built and maintained by the decoder, which raises the possibility of using the structure for compressed domain searching. If arithmetic coding is used, the

decoding has to be completed in full but the decoded output can be ignored. On the other hand if Huffman coding is used or the contexts are stored in the trie corresponding to word boundaries, there is the possibility of doing a direct search on the compressed data. Also, the trie could be stored in memory after decoding to use as an index to perform multiple pattern matching operations on the text. If the file is compressed using the DMC algorithm, as with PPM, there may be some possibility of using the finite state machine data structure to search for patterns after the whole text has been decoded.

The BWT compression method is based on lexicographic sorting of the forward context of each of its characters in the block. As such, a binary search algorithm can be employed to locate patterns at the encoding side. The decoder has only limited information about the sorted context, but it may be possible to exploit this to perform an initial match on two symbols (a character and its immediately preceding character), and then decode only that part of the text to see if the pattern match continues. It is possible to reconstruct the block sorted matrix at the decoder side and then perform arbitrary length pattern search. The process has to be repeated for each block to complete the pattern search. An inverted index file giving the blocks where the pattern may possibly occur, might expedite the search process.

Amir and Benson (Amir and Benson, 1992) described a method for searching two-dimensional data that has been compressed by run-length coding. An "optimal" version is described in (Amir et al., 1997). The general case of pattern matching for a class of "highly compressed" two-dimensional texts is explored by (Berman et al., 1996a; Berman et al., 1997; Berman et al., 1996b). They distinguished between compressed pattern matching, where the text is compressed, and fully-compressed pattern matching, where both the search pattern and the text are compressed.

Maa (Maa, 1993) considers a special case where the pattern to be located is a bar-code. Maa observes that for the CCITT fax standard, which uses both vertical and horizontal run-length coding, bar codes create distinctive coding patterns, and can be detected reliably. It may be possible to extend this idea to other types of images; for example, half tone images will compress very poorly using run-length coding; text will have many short runs; and line drawings will have many long runs of the same color.

Table 2 below shows the theoretical performance for various proposed algorithms for compressed pattern matching, using lossless compression schemes.

7 Searching compressed data: lossy compression

If the data being searched has been compressed with a lossy method then the pattern matching needs to be approximate to accommodate the possibility that the target of the search may have been changed slightly by the compression

s.n		Compression method	Search strategy	Exact match	Approx. match	Time complexity	Space complexity
0		naive		$\surd$	$\surd$	$O(u)$	$O(n+m)$
1		RLE	d.p.	$\surd$	$\surd$	$O(u\,m_c$ or $O(nu+mm_c$	
1b		RLE	LCS	$\surd$	$\surd$	$O(nm_c log(nm_c)$	
2		LZ77		$\surd$		$O(n\log^2(\frac{u}{n})+m)$	
3		LZ78, LZW	KMP	$\surd$		$O(n+m^2)$ or $O(n\log m+m)$	$O(n+m^2)$ or $O(n+m)$
4		word Huffman	BM, SHIFR-OR	$\surd$	$\surd$	$O(n+m)$ or $O(n+m\sqrt{u})$	$O(\sqrt{u})$
5		LZ78, LZW	d.p.	$\surd$	$\surd$	$O(mkn+r)$ $O(k^2n+\min\{mkn, m^2(m\Sigma)^k\}+r)$	$O(mkn+n\log n)$ or
6		LZW	suffix trees	$\surd$		$O(n+m\sqrt{m}\log m)$ or $O(nk+m^{1+\frac{1}{\alpha}}\log m), \alpha \geq 1$	
7		LZ77		$\surd$		$O(n+m)^6$	$O(n\log^2 u+n^2\log u +m_c\log\log m_c)$
8		LZ77, LZ78	SHIFT-OR	$\surd$		$O(\min\{u, n\log m\}+r)$ or $O(\min\{u, mn\}+r)$ w.c.	$O(n+r)$
9		LZW	SHIFT-AND	$\surd$	$\surd$	$O(n+r)$	$O(n+m)$
10		LZ78, LZW	BM	$\surd$		$\Omega(n)$, $O(mu)$ w.c.	$O(n+r)$
11		antidictionaries	KMP	$\surd$		$O(m^2+a+n+r)$	$O(m^2+a)$
12		gen. dictionary	BM	$\surd$		$O(f(d).(d+n)+n.m+m^2+r)$	$O(d+m^2)$

Table 2: Methods for compressed pattern matching for text. See Table 3 for the corresponding references. Key: P original pattern, P_c compressed pattern, T original text, T_c compressed text, *a:* size of antidictionary, d: size of dictionary, $u = |T|$= length of uncompressed text, $n = |T_c|$= length of compressed text, $m = |P|$ = length of uncompressed pattern, $m_c = |P_c|$ = length of compressed pattern, k number of differences allowed, Σ alphabet size, r number of occurrences of pattern in text, $f(d)$ function of the dictionary, depends on the tokens in d, d.p. = dynamic programming, LCS = longest common subsequence.

s.n	Reference
0	
1	(Bunke and Csirik, 1993; Bunke and Csirik, 1995)
1b	(Apostoloco et al., 1997)
2	(Farach and Thorup, 1995)
3	(Amir et al., 1996)
4	(Ziviani et al., 2000; Moura et al., 2000)
5	(Kärkkäinen et al., 2000)
6	(Kosaraju, 1995)
7	(Karpinski et al., 1995; Gąsieniec et al., 1996)
8	(Navarro and Raffinot, 1999)
9	(Kida et al., 1999)
10	(Navarro and Tarhio, 2000)
11	(Shibata et al., 1999b)
12	(Shibata et al., 2000)

Table 3: References for Table 2

process. Although lossy compression can be used for different types of multimedia data (such as video, images and audio), here our emphasis is on images. In general, approximate pattern matching for images has been studied in the context of content-based image retrieval. Content-based retrieval is concerned with retrieving images based on their true (visual) content, rather than by some textual description of such content.

Like the usual pattern matching problem, the image retrieval problem can be stated as follows: given a database of images, and a query image, find all the images in the database that are similar to the query image. Object-level image retrieval is a variant of this, where the objective includes finding all the database images that contain image subparts that are similar to the query image. In general, the image retrieval problem is mainly concerned with dermining the following:

- *The features to be used*: Because of the subjective (visual) nature of image contents, different features of the image (such as color, shape, texture, spatial information, etc.) could be used in matching images. The appropriate features to be used often depend on the particular application.

- *The representations to be used*: When we know the features to be used, we will need to find appropriate representations for the features that will be suitable for retrieval. (Chang et al., 1997) provides an overview of techniques for finding images in large archives. They point out the importance of representing the information in a form suitable for searching. Different features require different representations, and at times, the same feature can be represented in different ways.

- *The similarity criteria*: The suitable similarity criteria usually depend on the particular feature that is being used. Sometimes, a combination of the features (and hence various similarity metrics) will be required for effective matching of the images.

For each feature, appropriate methods are required for their extraction, efficient representation, and proper similarity matching. Recent surveys on image retrieval can be found in (Aigrain et al., 1996; Rui et al., 1999; Smeulders et al., 2000). Content-based retrieval systems are surveyed in (Veltkamp and Tanase, 2000).

Because of the promise of efficiency in compressed domain operations, and since images are now typically stored in the compressed form, compressed domain image retrieval has attracted some serious attention and various methods have been proposed (Ahanger and Little, 1996; Mandal et al., 1999a). The general problem of image matching involves some form of image processing and analysis. Schmalz (Schmalz, 1992) presented a general introduction to processing compressed data. He also provided an overview of the problem of recognising patterns in compressed data (Schmalz, 1995a) and gave optical processing methods for the problem (Schmalz, 1995b).

Below, we use the basic compression methodologies (block-transform, wavelets, VQ, fractals, etc.) as a guide and provide a brief survey of the reported work on compressed pattern matching for images.

In general, the techniques use the statistics of the compressed domain coefficients to form a feature vector for the database images, which are then used for later matching with those from a query image. For instance, Chang and his colleagues have studied various issues in compressed domain image and video manipulations, including methods for searching, especially for DCT and wavelet-based coding (Meng and Chang, 1996; Smith and Chang, 1994a; Wang and Chang, 1995; Chang, 1995). In (Smith and Chang, 1994b), they computed the mean and variance of the coefficients from the DCT blocks, and used these to form the feature vector for the images. (Reeves et al., 1997) used a similar method, but observed that not all the DCT coefficients are very useful in discriminating between images, and hence used the first few significant DCT coefficients.

Lew and Huang (Lew and Huang, 1994) investigated matching in transform block encoded images, specifically using the Karhunen-Loéve transform and the discrete cosine transform (DCT). Li, Turek and Feig (Li et al., 1995) also investigated searching DCT coded data, but in a progressive transmission situation where the low-frequency DCT information is searched first. Other methods for searching on DCT-based compressed images have been reported in (Wei et al., 1998; Ngo et al., 1998).

Zhang and colleagues (Zhang et al., 1995a; Zhang et al., 1995b) performed texture-based image retrieval on fractal-compressed images by matching the fractal codes. Given two images, fractal code matching is performed by comparing their range transformations. They looked for overlaps in the images, since each range in an image is best approximated by a domain that lies within the image. With fractal coding, given two identical range blocks each in a different image, the domain blocks that will provide the best approximation for the two range blocks must be identical. Thus the fractal codes for the two range blocks must be the same. If two images are identical, corresponding range images (and hence their fractal codes) will also be identical, and hence retrieval can be performed based on the fractal codes.

For the more usual case, when the images are not identical, the images are decomposed into smaller blocks, which are then coded as fractals. Since only some of the ranges in a query image will match some other ranges in a database image, they defined a *matching rate*, which is the number of matched transformations between two images. Similarity between images is then measured based on the matching rate, where a higher matching rate suggests more similarity.

A slightly different approach was taken in (Sloan, 1994). Here, the query image is combined with each image in the database to generate a synthetic image via fractal coding. Using a scoring function, the generated code is analyzed to determine to what extent the query image is described in terms of itself or in terms of the database image. Using the score, the similar images to the query image can be determined. The problem with this method is its huge computational complexity. For a moderately sized database, and with reasonably sized

domains and ranges, it might take too long to generate and code the synthetic images, and then to search for the best matching approximations.

Indexing and retrieval for VQ-coded images have been explored in (Idris and Panchanathan, 1997). Here the indices of the codewords were defined as labels for the images, and a histogram of the labels was then used for image retrieval. In (Vellaikal et al., 1995), the VQ codewords were used as image content descriptors suitable for retrieval. Oehler and Gray (Oehler and Gray, 1993) use the information available in VQ-based coding to identify tumours in computerised tomography (CT) images, and to identify roads and buildings in aerial images.

Because of its multiscale image decomposition capability, the subband/wavelet-based compression schemes have been quite popular for texture-based image retrieval. In (Chang and Kuo, 1993), images were decomposed into different subbands, and feature vectors derived from the most significant coefficients in the middle subbands were used for texture matching. A similar approach was taken in (Smith and Chang, 1994b), where they used the energy of the subbands to determine the texture features to be used for image matching. In (Mandal et al., 1999b), a histogram of the wavelet coefficients from different subbands was used for image retrieval. They observed that while different images could have similar overall histograms, the statistics of the different bands are likely to be different, and hence can be used to discriminate between images. (Manjunath and Ma, 1996) considered Gabor wavelets, and used the mean and standard deviation of the coefficients in each subband to differentiate between different images. In (Wickerhauser, 1994), information from different wavelet basis was used to investigate ways to recognise and classify images.

A comparison of fractal coding and wavelet transforms in image retrieval is provided in (Zhang. et al., 1996); (Idris and Panchanathan, 1995) compare wavelets and vector quantization, while (Smith and Chang, 1994b) provide a similar comparison for wavelet-based compression and DCT for texture retrieval. It was concluded that retrieval is generally faster on wavelet-based compressed images than on fractal-coded images; that wavelets are better for images with strong edges, while fractals are better for more general images; that fractals are better suited for comparing sub-images within database images; and that wavelet-based image coding is more suitable for texture-based retrieval than the DCT. Fractals could thus provide a method for approaching the difficult problem of object-level image retrieval.

We observe that the exact performance of the compressed domain image retrieval schemes often depends on the specific features used, and on the application. There is still no one compression technique that has proved to be better than all the others for the different features we might want to use to retrieve images. Some techniques have thus attempted to retrieve images when the underlying compression is based on a hybrid of two or more compression schemes. In (Idris and Panchanathan, 1995), wavelet-based vector-quantized images were considered. The original image is decomposed using the wavelet transform and the wavelet coefficients are then coded using vector quantization. The codebook used in the VQ was then used as the index for later image retrieval. (Swan-

son et al., 1996) also considered retrieving images compressed with a hybrid of wavelets and VQ, but with some image regions coded with the DCT (the JPEG algorithm).

Some methods for compressed-pattern matching have also been proposed for images compressed with non-traditional coding techniques. Vasconcelos and Lippman (Vasconcelos and Lippman, 1997) argue that a library of relevant objects can be developed during the compression of a file, so the search component of the coding does double duty as searching later on. This principle is similar to some compressed full-text retrieval systems where a lexicon of words is constructed which is used as a library for compression and also an index for full-text searching (Witten et al., 1999). Chen and Bovik (Chen and Bovik, 1992) proposed a system that attempts to code sub-images that are "meaningful to a normal human observer." Their system is called *visual pattern image coding* (VPIC). A similar approach was taken by Gerek et al (Gerek et al., 1996), who developed a textual image compression system that is able to search the compressed data.

A more unusual form of lossy compression is the *signature file* (Harrison, 1971), which has been popular in text information retrieval systems. A "signature" is created for each document or record using hashing, and a simple probabilistic test can be used to find possible matches of a search key with a signature. This is a specialist system, and allows complex queries including boolean combinations of terms. The signatures are not intended to be decompressed (in fact, the system is not really touted as a compression scheme).

8 Directions for further research

So far, we have surveyed research on different aspects of the problem of searching on compressed data. In this section, we speculate on what is likely to preoccupy researchers in this field in the short term and in the long run. The short term concern will be mainly on problems that deserve immediate attention, or those that can build on existing ideas, while the long term research direction might question some of the current approaches used in pattern matching, compression, or compressed pattern matching. Below, we describe these research activities under six subheadings: new compressed pattern matching algorithms, new compression algorithms, new applications, integration and adaptation, and hardware implementation.

8.1 New compressed pattern matching algorithms

Compressed pattern matching is a relatively new field. The next obvious step is to make incremental improvements on some of the currently proposed methods, for example, making them faster or to use less extra space. However, although the focus has been mainly on a few compression methods (such as Huffman coding and LZ compression for text, or the RLE and DCT for images), algorithms are still required for pattern matching on other/new compression algorithms.

For instance, a few methods have been proposed (Shibata et al., 1999b) for searching text compressed using schemes such as antidictionaries (Crochemore et al., 2000). But we are yet to see algorithms that search on data compressed with block-sorted contexts such as BWT. Because the performance measures could be different when searching is considered, algorithms that may not be very good on compaction and coding complexity could provide better performance for searching.

In the long run, researchers will still be looking for possible solutions to some currently hard questions. For instance, can there be some variations on arithmetic coding that will make it possible for compressed pattern matching? How can we perform precise image searching directly on compressed data? How can we make new compression algorithms (such as wavelets and fractals) faster, which will make them more generally attractive for multimedia data compression, and hence more important for compressed pattern matching in images or video?

Further, current algorithms have focused mainly on the basic pattern-matching problem (exact or approximate pattern matching). However, in the short term, there is a need to start looking into compressed solutions to the other variants of the compressed pattern-matching problem, especially those with immediate applications. For instance, compressed multidimensional pattern matching will find use in new applications, such as multimedia data analysis, while compressed solutions to the dictionary matching and super pattern-matching problems will be of interest to researchers in data mining. In the long term, we expect some compressed solutions to the more theoretical variants of the pattern-matching problem, such as pattern matching with don't cares.

Although searching on compressed data is inherently more efficient than on uncompressed data, the use of parallel algorithms can provide further efficiency improvements. Parallel algorithms have been proposed for the traditional pattern matching problem (Galil and Giancarlo, 1997; Giancarlo and Gross, 1997). We envisage similar efforts in developing parallel compressed pattern matching algorithms, for both text and images. There are two obvious ways to do this: either develop parallel versions of existing compressed pattern matching algorithms, or to develop analogous compressed pattern matching versions for currently available parallel pattern matching algorithms.

For images, the main attention has been on DCT-based compression schemes, perhaps due to the availability of DCT-based standards, such as JPEG and MPEG. Very little has been done on searching on images compressed with other techniques, such as the subband decomposition techniques or fractals. Yet some of these emerging image compression methods (such as fractals and wavelets) have shown promise for huge compression ratios, which make them candidates for archival applications, or typical multimedia environments. Further, other schemes (such as quadtrees and fractals) could provide ideas for addressing the difficult problem of object-level searching in compressed images. Already some of these have been found to be particularly suitable for searching based on particular image features. With the success of whole-image approximate searching using DCT based features, we anticipate more effort in developing equivalent re-

trieval algorithms for images that are compressed with these emerging schemes.

The harder question is the problem of precise search on images — whether on compressed or non-compressed images. For compressed images, matching has generally been approximate, since the compression is usually lossy. Thus, initial attempts to this problem will have to look at developing pattern-matching algorithms for lossless image compression schemes, such as JPEG-LS (the lossless version of JPEG (Weinberger et al., 2000)). Further, with the increasing importance of applications for lossless image compression (such as medical imaging), researchers will start to look at the problem of developing equivalent algorithms that can search directly on lossless compressed images.

8.2 New search-aware compression algorithms

Already there have been proposals for *search-aware compression algorithms* — that is, compression schemes designed with search in mind (Manber, 1997; Shibata et al., 1999b). We envisage that this trend will continue, especially with the increased demand for "compress-once, search-many" applications, such as digital libraries, and multimedia databases. In this situation, the ability to support later searching becomes a performance criterion used to measure the algorithms.

Image compression schemes have, however, not generally considered the problem of search at the compression stage. This is bound to change. Already, as part of the compression process, some image compression methods provide useful by-products, (such as code vocabulary (e.g. VQ) or data structures (e.g. PMIC)), which can be exploited later to search the compressed data. Also, for images, object-level search (i.e. within-image search for objects) has posed a serious problem and has been largely ignored. The major issue has been how to describe the shape boundaries in the transform domain. New coding methods, such as the shape-adaptive DCT-based schemes (Sikora et al., 1995) open up the possibility of now describing object boundaries as part of the compression. New standards such as MPEG-4 and MPEG-7 are also considering issues related to object-level access in the compressed data (Zhang et al., 1997; Sikora, 1997; MPEG-4, 2000).

In general, new multimedia compression algorithms are expected to take the issue of searching directly on the compressed data more seriously. In the least, new search-aware compression algorithms that are tuned for special applications where searching is a key issue (such as web-based applications, digital libraries, multimedia databases) are expected in the near term for lossy compression. In the long term, we expect a similar development for lossless image compression.

8.3 New applications

Traditionally, pattern matching (especially exact-pattern matching) has been used mainly in text-based environments. New applications, such as image retrieval has motivated the need for approximate pattern matching for images. But the simple problem of exact matching is yet to be solved for images, due

to the difficult question of image registration. Notwithstanding this, we still envisage that emerging applications, such as web-based information retrieval (Kobayashi and Takeda, 2000), digital libraries, and multimedia information systems will drive the need for compressed pattern matching in new application environments involving different data types.

This will particularly be the case for applications that require access to multimedia content, where the large data sizes typically involved still make efficiency considerations a major issue. Another instance here is the potential impact of solutions to the problems of super-pattern matching or dictionary matching on new applications, such as data mining. More generally, in the long term, the development of compressed pattern matching algorithms will encourage the drive for some new application areas, which have so far been thought to be too time consuming.

8.4 Performance measures

The performance measure for new search-aware compression algorithms will include the capacity for explicit search support, along with the traditional measures of data compaction, complexity, and quality (in the case of lossy compression). Short-term problems here include the development of benchmark databases for testing the various algorithms. There are already a number of standard databases for testing text compression algorithms (CanterburyCorpus, 2000; CalgaryCorpus, 2000), or for text-based retrieval systems (TREC, 2000). A similar benchmark for evaluating algorithms that search directly on compressed images requires immediate attention. Further, we have enumerated a number of parameters based on which compressed pattern-matching algorithms can be evaluated. In the long run, researchers might also need to re-examine these and the current measures, possibly, with a view to developing application-specific measures of performance.

8.5 Integration and Adaptation

Most environments will involve data corpora that are compressed with different compression schemes. This is currently typical of web-based applications, and the number of such applications is bound to increase in the future. One problem will be how to perform search transparently on the different data formats or data types.

In the long run, we envisage a proliferation of compression schemes (and hence different methods to search on them directly), and the availability of different search techniques on the same compression algorithm. Moreover, for image matching algorithms, there is the possibility of missing the occurrence of the query image in the database, or returning wrong results altogether. When we consider all these, along with the different performance criteria (some of them opposing to each other), the obvious question will be how to make compressed pattern matching to be adaptive.

Adaptation could be in different forms. For example, in the choice of the compression scheme — for instance, based on the perceived future search activities on the data and/or the matching algorithm that will be adopted; choice of the particular compressed pattern matching algorithm to use — for instance, based on the weights attached to the different performance measures; the use of a hierarchy of search algorithms (especially for image matching algorithms) — akin to the idea of *metasearch* used for web search engines (Silberg and Etzion, 1997; Lawrence and Giles, 1998).

8.6 Hardware implementation

In parallel with the development of compression algorithms, very large scale integrated (VLSI) technology has made tremendous strides and opened up the possibility of implementing a system-on-chip (SoC) to perform real-time data compression functions over mobile and network based communication channel. Several papers have appeared in the literature (we mention only a small subset of these here) to perform hardware data compression. The basic operation of the LZ algorithm is a string matching operation. Hardware architectures for this are based on content-addressable memory (CAM) (Jones, 1992; Lee and Yang, 1995; Craft, 1998; Lin and Wu, 2000), linear systolic arrays (Ranganathan and Henriques, 1993; Jung and Burleson, 1998; Chen and Wei, 1999) and reconfigurable field programmable gate arrays (FPGAs) (Nunez et al., 1999; Huang et al., 2000). Hardware algorithms for Huffman and arithmetic compression have also been proposed by several authors (Pennebaker et al., 1988b; Arps et al., 1988; Mukherjee et al., 1991; Mukherjee et al., 1993) (Mukherjee and Acharya, 1995; Jiang, 1995; Liu et al., 1995; Parhi, 1992; Park et al., 1995; Freking and Parhi, 1999).

Several hardware architectures have been reported for DCT, JPEG, MPEG and wavelet based image compression, run-length encoding and move-to-front encoding. The references are too numerous to mention here.

With regard to hardware for compressed domain pattern search, only a few papers appear in the literature (Mukherjee and Acharya, 1995; Ercal et al., 2000; Wilson et al., 2000). Future work on hardware algorithms for data compression could be directed towards implementing some of the better performing lossless algorithms like BWT, PPM and DMC, and development of higher speed video and image compression hardware. Development of compressed domain pattern matching both for lossless and lossy compression algorithms will be an exciting challenge for future work.

9 Conclusion

In this paper, we have surveyed recent and past efforts in searching reduced (compressed) text and images. We outlined the basic concepts and assumptions used in data compression and in pattern matching. The paper also identified the special relationship between data compression and pattern matching (search-

ing). It was observed that searching is an important aspect of compression, while on the other hand, compression can be used to improve later searching. Algorithms that search directly on lossless compressed data have focused mainly on Huffman codes and the LZ family of coding algorithms. Searching on lossy image compression has generally been in the context of image retrieval, and generally on an approximate basis, with emphasis on transform-coded images.

The paper proposes six measures of performance for compressed pattern-matching: complexity and speed, extra space, optimality, precision and recall, ranking, and comparison with decompress-and-search algorithms. We speculate that in the future, short term and long term trends in compressed pattern matching will focus on important aspects of the problem: new compressed pattern matching algorithms, new search-aware compression algorithms, development of new applications, performance measures and benchmarks, adaptation in compressed pattern matching, parallel algorithms and hardware implementation.

References

Adjeroh, D. A. and Lee, M. C. (1997). Robust and efficient transform domain video sequence analysis: An approach from the generalized color ratio model. *Journal of Visual Communication and Image Representation*, 8(2):182–207.

Adjeroh, D. A., Lee, M. C., and King, I. (1999). A distance measure for video sequence similarity matching. *Computer Vision and Image Understanding*, 75(1):25–45.

Ahanger, G. and Little, T. D. C. (1996). A survey of technologies for parsing and indexing digital video. *Journal of Visual Communication and Image Representation*, 7(1):28–43.

Aigrain, P., Zhang, H. J., and Petkovic, D. (1996). Content-based representation and retrieval of visual media: A state-of-the-art review. *Multimedia Tools and Applications*, 3:179–202.

Akutsu, T. (1994). Approximate string matching with don't care charaters. *Proceedings, Combinatorial Pattern Matching, LNCS 807*, pages 240–249.

Alzina, M., Szpankowski, W., and Grama, A. (1999). 2D-Pattern matching image and video compression: preliminary results. *Proceedings, Data Compression Conference*.

Alzina, M., Szpankowski, W., and Grama, A. (2000). 2D-Pattern matching image and video compression: theory,algorithms, and experiments. *IEEE Transactions on Image Processing*, 9(8).

Amir, A. (1992). Multiple pattern matching. Technical Report GIC-CC-92/29, College of Computing, Georgia Institute of Technology.

Amir, A. and Benson, G. (1992). Efficient two-dimensional compressed matching. In (Storer and Cohn, 1992), pages 279–288.

Amir, A., Benson, G., and Farach, M. (1996). Let sleeping files lie: Pattern matching in Z-compressed files. *Journal of Computer and System Sciences*, 52:299–307.

Amir, A., Benson, G., and Farach, M. (1997). Optimal two-dimensional compressed matching. *Journal of Algorithms*, 24:354–379.

Amir, A. and Calinescu, G. (1996). Alphabet independent and dictionary scaled matching. *Combinatorial Pattern Matching, LNCS 1075*, pages 320–334.

Amir, A., Landau, G., and Vishkin, U. (1992). Efficient pattern matching with scaling. *Journal of Algorithms*, 13:2–32.

Anastassiou, D., Brown, M., Jones, H., Mitchell, J., Pennebaker, W., and Pennington, K. (1983). Series/1-based videoconferencing system. *IBM Systems Journal*, 22(1/2):97–110.

Apostoloco, A., Landau, G. M., and Skiena, S. (1997). Matching for run-length encoded strings. *Proceedings, Complexity and Compression of Sequences.*

Arps, R. B., Truong, T. K., Lu, D. J., Pasco, R. C., and Friedman, T. D. (1988). A multi-purpose VLSI CAD chip for adaptive data compression of bilevel images. *IBM Journal of Research and Development*, 32(6):775–795.

Atallah, M., Génin, Y., and Szpankowski, W. (1999). Pattern matching image compression: algorithmic and experimental results. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21:618–627.

Baeza-Yates, R. and Gonnet, G. H. (1992). A new approach to text searching. *Communications of the ACM*, 35(10):74–82.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval.* Addison Wesley, Harlow, England.

Baker, T. (1978). A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM Journal on Computing*, 7(4):533–541.

Barcaccia, P., Cresti, A., and Agostino, S. D. (1998). Pattern matching in text compressed with the ID heuristic. In (Storer and Cohn, 1998), pages 113–118.

Barnsley, M. F. and Hurd, L. P. (1993). *Fractal Image Compression.* AK Press, Ltd, Wellesly, Mass.

Bell, T. and Kulp, D. (1993). Longest-match string searching for Ziv-Lempel compression. *Software—Practice and Experience*, 23(7):757–772.

Bell, T. and Moffat, A. (1989). A note on the DMC data compression scheme. *Computer Journal*, 32(1):16–20.

Bell, T. C. (2000). Data compression. In Ralston, A., Reilly, E., and Hemmendinger, D., editors, *Encyclopedia of Computer Science*, pages 492–496. Nature Publishing Group, fourth edition.

Bell, T. C., Cleary, J. G., and Witten, I. H. (1990). *Text Compression*. Prentice Hall, Englewood Cliffs, NJ.

Berman, P., Karpinski, M., Larmore, L., Plandowski, W., and Rytter, W. (1996a). The complexity of two-dimensional compressed pattern matching. Technical Report TR-96-051, International Computer Science Institute, Berkeley, CA.

Berman, P., Karpinski, M., Larmore, L., Plandowski, W., and Rytter, W. (1996b). The complexity of two-dimensional compressed pattern-matching. Technical Report 85156-CS, University of Bonn, Department of Computer Science.

Berman, P., Karpinski, M., Larmore, L., Plandowski, W., and Rytter, W. (1997). On the complexity of pattern-matching for highly compressed two-dimensional texts. In *Combinatorial Pattern-Matching, 8th Annual Symposium*. LNCS 1264.

Bird, R. S. (1977). Two dimensional pattern matching. *Information Processing Letters*, 6(5):168–170.

Boyer, R. and Moore, J. (1977). A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772.

Bunke, H. and Csirik, J. (1993). An algorithm for matching run-lenght coded strings. *Computing*, 50:297–314.

Bunke, H. and Csirik, J. (1995). An improved algorithm for computing the edit distance of run-lenght coded strings. *Information Processing Letters*, 54:93–96.

Bunke, H. and Sanfeliu, A., editors (1990). *Syntatic and Structural Pattern Recognition: Theory and Applications*. World Scientific, Singapore.

Burrows, M. and Wheeler, D. (1994). A block-sorting lossless data compression algorithm. Technical report, Digital Equipment Corporation, Palo Alto, California.

Burt, P. J. and Adelson, E. H. (1983). The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540.

Buyanovsky, G. (1994). Associative coding. *Monitor*, 8:10–19. In Russian.

CalgaryCorpus (2000). The Calgary Corpus, ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus.

CanterburyCorpus (2000). The canterbury corpus, http://corpus.canterbury.ac.nz.

CCITT (1993). Draft recommendation T.82 & ISO DIS 11544: Coded representation of picture and audio information—progressive bi-level image compression.

Chang, S.-F. (1995). Compressed-domain techniques for image/video indexing and manipulation. In *Proceedings of the International Conference on Image Processing*, volume 1, pages 314–317.

Chang, S.-F., Smith, J., Meng, H. J., Wang, H., and Zhong, D. (1997). Finding images/video in large archives: Columbia's content-based visual query project. *D-Lib magazine*.

Chang, T. and Kuo, C. J. (1993). Texture analysis and classification with a tree-structured wavelet transform. *IEEE Transactions on Image Processing*, 2(4):429–441.

Chang, W. I. and Lampe, J. (1992). Theoretical and empirical analysis of approximate string matching algorithms. *Proceedings, Combinatorial Pattern Matching, LNCS 644*, pages 175–184.

Chang, W. I. and Lawler, E. L. (1994). Sublinear approximate string matching and biological applications. *Algorithmica*, 12:327–344.

Chen, D. and Bovik, A. (1992). Visual pattern image coding. *IEEE Transactions on Communications*, 38(12):2137–2146.

Chen, J.-M. and Wei, C.-H. (1999). VLSI design for high-speed LZ-based data compression. *IEE Proceedings of Circuits, Devices and Systems*, 146(5):268–278.

Cleary, J. and Witten, I. (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, COM-32:396–402.

Cohn, M. (c. 1994). Data compression. In *Encyclopedia of Computer Science and Technology*.

Constantinescu, C. and Storer, J. (1994). Improved techniques for single-pass adaptive vector quantization. *Proceeding of the IEEE*, 82:933–939.

Cormack, G. and Horspool, R. (1987). Data compression using dynamic Markov modeling. *Computer Journal*, 30(6):541–550.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms*. MIT Press.

Craft, D. J. (1998). A fast hardware data compression algorithm and some algorithmic extensions. *IBM Journal of Research and Development*, 42(6).

Crochemore, M. and et al (1994). Speeding up two string-matching algorithms. *Algorithmica*, 12:247–267.

Crochemore, M. and LeCroq, T. (1996). Pattern-matching and text compression. *ACM Computing Surveys*, 28(1):39–41.

Crochemore, M., Mignosi, F., Restivo, A., and Salemi, S. (2000). Data compression using antidictionaries. *Proceedings of the IEEE*, 88(11):1756–1768.

DeVore, R. A., Jawerth, B., and Lucier, B. J. (1992). Image compression through wavelet transform coding. *IEEE Transactions on Information Theory*, 38(2):719–746.

Eilam-Tzoreff, T. and Vishkin, U. (1988). Matching patterns in strings subject to multi-linear transformations. *Theoretical Computer Science*, 60:231–254.

Ercal, F., Allen, M., and Feng, H. (2000). A systolic image difference algorithm for RLE-compressed images. *IEEE Transactions on Parallel and Distributed Systems*, 11(5):433–443.

Farach, M. and Thorup, M. (1995). String matching in Lempel-Zive compressed strings. In *Proceedings of the twenty-seventh annual ACM symposium on the Theory of Computing*, pages 703–712, New York. ACM.

Fenwick, P. (1996a). Block sorting text compression. In *Proceedings of the 19th Australasian Computer Science Conference*, pages 193–202.

Fenwick, P. (1996b). The Burrows-Wheeler Transform for block sorting text compression. *The Computer Journal*, 39(9):731–740.

Fischer, M. J. and Paterson, M. S. (1974). String matching and other products. *Complexity of Computation, Proceedings, SIAM-AMS.*

Fisher, Y., editor (1995). *Fractal Image Compression.* Springer-Verlag, New York.

Fisher, Y. and Woods, E. W. (1992). Fractal image compression using iterated transforms. In (Storer, 1992), pages 35–62.

Freking, R. A. and Parhi, K. K. (1999). An unrestrictedly parallel scheme for ultra-high rate reprogrammable Huffman coding. *Proc. IEEE ICASP (Int. Conf. Acoustics, Speech, and Signal Processing)*, 4.

Galil, Z. and Giancarlo, R. (1997). Parallel string matching with k mismatches. *Theoretical Computer Science*, 51:341–384.

Galil, Z. and Park, K. (1990). An improved algorithm for approximate string matching. *SIAM Journal of Computing*, 19(6):689–999.

Garris, M., Blue, J., Candela, G., Grother, P., Janet, S., and Wilson, C. (1997). NIST Form-based handprint recognition system (release 2.0).

Gąsieniec, L., Karpinski, M., Plandowski, W., and Rytter, W. (1996). Randomized efficient algorithms for compressed strings: the finger-print approach. *Proceedings, Combinatorial Pattern Matching, LNCS 1075*, pages 39–49.

Gerek, O., Çetin, A. E., and Tewfik, A. H. (1996). Subband domain coding of binary textual images for document archiving. (Submitted to the Computer Journal).

Gersho, A. and Gray, R. M. (1992). *Vector Quantization and Signal Compression.* Kluwa Academic Press, Boston.

Giancarlo, R. and Gross, R. (1997). Multi-dimensional pattern matching with dimensional wildcards: Data structures and optimal on-line search algorithm. *Journal of Algorithms*, 24:223–265.

Gibson, J. D., Berger, T., Lookabaugh, T., Lindbergh, D., and Baker, R. L. (1998). *Digital Compression for Multimedia: Principles and Standards.* Morgan Kaufmann, San Francisco.

Gonzalez, G. C. and Woods, R. E. (1992). *Digital Image Processing.* Addison-Wesley, Reading, Mass.

Harrison, M. (1971). Implementation of the substring test by hashing. *Communications of the ACM*, 14(12):777–779.

Hirschberg, D. and Lelewer, D. (1990). Efficient decoding of prefix codes. *Communications of the ACM*, 33(4):449–459.

Horspool, R. and Cormack, G. (1986). Dynamic Markov modeling — A prediction technique. In *Proceedings 19'th HICSS*, pages 700–707, Honolulu.

Huang, W.-J., Saxena, N., and McCluskey, E. J. (2000). A reliable LZ data compressor on reconfigurable coprocessor. *Proc. Symposium on Field Programmable Custom Computing Machine*, pages 249–258.

Huffman, D. (1952). A method for the construction of minimum redundancy codes. *Proc. IRE*, 40(9):1098–1101.

Hume, A. and Sunday, D. (1991). Fast string searching. *Software—Practice and Experience*, 21(11):1221–1248.

Hunter, R. and Robinson, A. H. (1980). International digital facsimile coding standards. *Proc. IEEE*, 68(7):854–867.

Idris, F. and Panchanathan, S. (1995). Image indexing using wavelet vector quantization. *Proceedings of SPIE: Digital Image Storage Archiving Systems*, 2606:269–275.

Idris, F. and Panchanathan, S. (1997). Image and video indexing using vector quantization. *Journal of Machine Vision and Applications*, 10:43–50.

Idury, R. M. (1994). Dynamic multiple pattern matching, PhD Thesis. Technical report, Department of Computer Science, Rice University.

Inglis, S. and Witten, I. (1994). Compression-based template matching. In (Storer and Cohn, 1994), pages 106–115.

Jacquin, A. E. (1992). Image coding based on a fractal theory of iterated contractive image transformations. *IEEE Transactions on Image Processing*, 1(1):18–30.

Jacquin, A. E. (1993). Fractal image coding: A review. *Proceedings of the IEEE*, 81(10).

Jain, A. K. (1981). Image data compression: a review. *Proceedings of the IEEE*, 69:349–289.

Jiang, J. (1995). Novel design for arithmetic coding for data compression. *IEE Proc. on Computers and Digital Techniques*, 142(6):419–424.

Johansen, P. (1994). Pattern recognition by data compression. Technical report, DIKU, Department of Computer Science, University of Copenhagen, Denmark.

Jones, S. (1992). 100-Mbps adaptive data compression design using selectively shiftable CAM. *IEE Proc. Pt.G*, 128(8):498–502.

Jung, B. and Burleson, W. P. (1998). Efficient VLSI for Lempel-Ziv compression in wireless communication network. *IEEE Transactions on VLSI Systems*, 6(3):475–483.

Kärkkäinen, J., Navarro, G., and Ukkonen, E. (2000). Approximate string matching Ziv-Lempel compressed text. *Proceedings, Combinatorial Pattern Matching, LNCS 1848*, pages 195–209.

Karp, R. M. and Rabin, M. O. (1981). Efficient randomized pattern matching algorithms. Technical Report TR-31-8, Aiken Computation Lab, Harvard University.

Karpinski, M., Plandowski, W., and Rytter, W. (1995). The fully compressed string matching for Lempel-Ziv encoding. Technical Report 85132-CS, Department of Computer Science,University of Bonn.

Khan, H. U. and Fatmi, H. A. (1993). A novel approach to data compression as a pattern recognition problem. In (Storer and Cohn, 1993).

Kida, T., Takeda, M., Shinohara, A., and Arikawa, S. (1999). Shift-And approach to pattern matching in LZW compressed text. *Proceedings, Combinatorial Pattern Matching, LNCS 1645*, pages 1–13.

Klein, S. T. (2000). Improving static compression schemes by alphabet extension. *Combinatorial Pattern Matching, LNCS 1848*, pages 210–221.

Knight, J. R. and Myers, E. W. (1999). Super-pattern matching. Technical Report TR-92-29, Department of Computer Science, University of Arizona.

Knuth, D., Morris, J., and Pratt, V. (1977). Fast pattern matching in strings. *SIAM Journal of Computing*, 6(2):323–350.

Kobayashi, M. and Takeda, K. (2000). Information retrieval on the web. *ACM Computing Surveys*, 32(2):144–173.

Kosaraju, S. (1995). Pattern matching in compressed texts. In Thiagarajan, P., editor, *Proceedings, Foundations of Software Technology and Theoretical Computer Science, 15th Conference*, pages 349–362. Springer-Verlag.

Landau, G. M. and Vishkin, U. (1988). Fast string matching with k differences. *Journal of Computer and System Sciences*, 37:63–78.

Landau, G. M. and Vishkin, U. (1994). Pattern matching in a digitized image. *Algorithmica*, 12(4/5):375–408.

Langdon, G. (1984). An introduction to arithmetic coding. *IBM Journal of Research and Development*, 28(2):135–149.

Langdon, G. and Rissanen, J. (1982). A simple general binary source code. *IEEE Transactions on Information Theory*, IT-28:800–803.

Lawrence, S. and Giles, C. (1998). Context and page analysis for improved web search. *IEEE Internet Computin*, 2(4):38–46.

Lee, C. Y. and Yang, R. Y. (1995). High-throughput data compressor design using content addressable memory. *IEE Proc. Pt. G*, 142(1):69–73.

Lee, J. S., Kim, D. K., Park, K., and Cho, Y. (1997). Efficient algorithms for approximate string matching with swaps. *Proceedings, Combinatorial Pattern Matching, LNCS 1264*, pages 28–39.

LeGall, D. (1991). MPEG: a video compression standard for multimedia applications. *Communications of the ACM*, 34(4):47–58.

Lew, M. and Huang, T. (1994). Image compression and matching. In *ICIP-94*, volume 2, pages 720–724. IEEE Computer Society Press.

Li, C.-S., Turek, J., and Feig, E. (1995). Progressive template matching for content-based retrieval in Earth Observing Satellite image database. *SPIE*, 2606:134–144.

Lin, K.-J. and Wu, C.-W. (2000). A low power CAM design for LZ data compression. *IEEE Transactions on Computers*, 49:1139–1145.

Liu, L.-Y., Wang, J.-F., Wang, R.-J., and Lee, J.-Y. (1995). Design and hardware architectures for dynamic Huffman coding. *IEE Proc. on Computers and Digital Techniques*, 142(6):411–418.

Luczak, T. and Szpankowski, W. (1994). A lossy data compression based on string matching: preliminary analysis and suboptimal results. *Proceedings, Combinatorial Pattern Matching, LNCS*.

Luczak, T. and Szpankowski, W. (1997). A suboptimal lossy data compression based on approximate pattern matching. *IEEE Transactions on Information Theory*, 43:1439–1451.

Maa, C.-Y. (1993). Identifying the existence of bar codes in compressed images. In (Storer and Cohn, 1993), page 457.

Manber, U. (1997). A text compression scheme that allows fast searching directly in the compressed file. *ACM Transactions on Information Systems*, 15(2):124–136.

Mandal, M. K., Idris, F., and Panchanathan, S. (1999a). A critical evaluation of image and video indexing techniques in the compressed domain. *Journal of Image and Vision Computing*, 17(7):513–529.

Mandal, M. K., Panchanathan, S., and Aboulnas, T. (1999b). Fast wavelet histogram techniques for image indexing. *Computer Vision and Image Understandin*, 75(1):99–110.

Manjunath, B. S. and Ma, W. Y. (1996). Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–841.

Memon, N., Neuhoff, D. L., and Shende, S. (2000). An analysis of some common scanning techniques for lossless image coding. *IEEE Transactions on Image Processing*, 9(11):1837–1848.

Meng, J. and Chang, S.-F. (1996). Tools for compressed-domain video indexing and editing. In *SPIE Conference on Storage and Retrieval for Image and Video Database*, volume 2670.

Miller, V. and Wegman, M. (1984). Variations on a theme by Ziv and Lempel. Technical report, IBM.

Miller, V. and Wegman, M. (1985). Variations on a theme by Ziv and Lempel. In Apostolico, A. and Galil, Z., editors, *Combinatorial Algorithms on Words, Volume 12*, NATO ASI Series F, Berlin. Springer-Verlag.

Moffat, A. (1990). Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–1921.

Moffat, A. (1991). Two-level context-based compression of binary images. In (Storer and Reif, 1991), pages 382–391.

Mongeau, M. and Sankoff, D. (1990). Comparison of musical sequencs. *Computers and the Humanities*, 24:161–175.

Moura, E. S., Navarro, G., and Baeza-Yates, R. (2000). Fast and flexible word searching on compressed text. *ACM Transactions on Information Systems*, 18(2):113–139.

MPEG-4 (2000). The Official MPEG Homepage http://www.cselt.it/mpeg/.

Mukherjee, A. and Acharya, T. (1994). Compressed pattern-matching. In (Storer and Cohn, 1994), page 468.

Mukherjee, A. and Acharya, T. (1995). VLSI algorithms for compressed pattern search using tree based codes. In *Proceedings, International Conference on Application Specific Array Processors*, pages 133–136.

Mukherjee, A., Ranganathan, N., and Bassiouni, M. (1991). Efficient VLSI designs for data transformation of tree-based codes. *IEEE Transactions on Circuits and Systems*, 38(3):306–314.

Mukherjee, A., W., F. J., Ranganathan, N., and Acharya, T. (1993). MARVLE: A memory based architecture for variable length encoding. *IEEE Transactions on VLSI Systems*, 1(2):203–214.

Myers, E. W. (1994). A sublinear algorithm for approximate keyword searching. *Algorithmica*, 12:345–374.

Navarro, G. and Raffinot, M. (1999). A general practical approach to pattern matching over Ziv-Lempel compressed text. *Proceedings, Combinatorial Pattern Matching, LNCS 1645*, pages 14–36.

Navarro, G. and Raffinot, M. (2000). Fast and flexible string matching by combining bit-parallelism and suffix automata. *ACM Journal of Experimental Algorithms*, 5(4).

Navarro, G. and Tarhio, J. (2000). Boyer-Moore string matching over Ziv-Lempel compressed text. *Proceedings, Combinatorial Pattern Matching, LNCS 1848*, pages 166–180.

Netravali, A. N. and Haskell, B. G. (1988). *Digital Pictures: Representation and Compression*. Plenum Press, New York.

Netravali, A. N. and Limb, J. O. (1980). Picture coding: A review. *Proceedings of the IEEE*, 68(3):366–406.

Ney, H. and Ortmanns, S. (2000). Progress in dynamic programming search for LVCSR. *Proceedings of the IEEE*, 88(8):1224–1240.

Ngo, C. W., Pong, T. C., and Chin, R. T. (1998). Exploiting image indexing techniques in DCT domain. *Proceedings, MINAR98: Multimedia Information Analysis and Retrieval, LNCS 1464*, pages 195–206.

Nunez, J. L., Feregrino, C., Bateman, S., and Jones, S. (1999). The X-MatchLITE FPGA-based data compressor. *Proc. Euromicro Conference*, 1:126–132.

Oehler, K. and Gray, R. (1993). Combining image classification and image compression using vector quantization. In (Storer and Cohn, 1993), pages 2–11.

Owolabi, O. and McGregor, D. R. (1988). Fast approximate string matching. *Software – Practice and Experience*, 18:387–393.

Parhi, K. K. (1992). High-speed VLSI architectures for Huffman and Viterbi decoders. *IEEE Transactions on Circuits and Systems*, 39(6).

Park, H., Son, J. C., and Cho, S. R. (1995). Area efficient fast huffman decoder for multimedia applications. *Proc. ICASP (Int. Conf. on Acoustics, Speech and Signal Processing*, 5:3279–3281.

Pennebaker, W. and Mitchell, J. (1988). Probability estimation for the Q-coder. *IBM Journal of Research and Development*, 32(6):737–752.

Pennebaker, W., Mitchell, J., Langdon, G., and Arps, R. (1988a). An overview of the basic principles of the Q-coder adaptive binary arithmetic coder. *IBM Journal of Research and Development*, 32(6):717–726.

Pennebaker, W. B., Langdon Jr, G. G., and Arps, R. B. (1988b). An overview of the basic principles of the Q-coder adaptive binary arithmetic coder. *IBM Journal of Research and Development*, 32(6):717–726.

Pennebaker, W. B. and Mitchell, J. L. (1993). *JPEG: Still Image Data Compression Standard.* Van Nostrand Reinhold, New York.

Pentland, A., Picard, R. W., and Scaroff, S. (1996). Photobook: Content-based manipulation of image databases. *International Journal of Computer Vision*, 18(3):233–254.

Pevzner, P. A. and Waterman, M. S. (1993). A fast filtration algorithm for the substring matching problem. *LNCS 684, Combinatorial Pattern Matching*, pages 197–214.

Pratt, W. K. (1991). *Digital Image Processing.* John Wiley, New York.

Ranganathan, N. and Henriques, S. (1993). High speed VLSI design for Lempel-Ziv-based data compression. *IEEE Transcations on Circuits and Systems*, 40:96–106.

Reeves, R., Kubik, K., and Osberger, W. (1997). Texture characterization of compressed aerial images using DCT coefficients. *Proceedings of SPIE: Storage and Retrieval for Image and Video Databases*, 3022:398–407.

Rigoutsos, I. and Califano, A. (1994). Searching in parallel for similar strings. *IEEE Computational Science and Engineering*, pages 60–67.

Rui, Y., Huang, T. S., and Chang, S. F. (1999). Image retrieval: current techniques, promising directions, and open issues. *International Journal of Visual Communications and Image Representation*, 10:39–62.

Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustic, Speech and Signial Processing*, 26(2):43–49.

Salomon, D. (1998). *Data Compression: The Complete Reference*. Springer.

Samet, H. (1984). The quadtree and related hierarchical data structures. *ACM Computing Survey*, 16(2):187–260.

Schmalz, M. (1992). General theory for the processing of compressed and encrypted imagery, with taxonomic analysis. *Proceedings, SPIE: Hybrid Image and Signal Processing III*, 1702:250–263.

Schmalz, M. (1995a). An introduction to the recognition of patterns in compressed data. 1. Image-template operations over block-, transform-, runlength-encoded, and vector-quantized data. *Proceedings of the SPIE*, 2484:256–269.

Schmalz, M. (1995b). An introduction to the recognition of patterns in compressed data. 2. Optical processing of data transformed by block-, transform-, and runlength-encoding, as well as vector-quantization. *Proceedings of the SPIE*, 2490:334–349.

Sellers, P. (1980). The theory of computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1:359–373.

Shannon, C. (1948). A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423, 623–656.

Shannon, C. and Weaver, W. (1949). *The Mathematical Theory of Communication*. The University of Illinois Press, Urbana, Illinios.

Sharman, N. (1992). An empirical comparison of progressive image compression techniques. Master's thesis, University of Canterbury, Christchurch, New Zealand.

Sharman, N., Bell, T., and Witten, I. (1992). Compression of pyramid coded images for progressive transmission. In *Proc 7'th New Zealand Image Processing Workshop*, pages 171–176, University of Canterbury, Christchurch, New Zealand.

Shibata, Y., Kida, T., Fukamachi, S., Takeda, T., Shinohara, A., Shinohara, S., and Arikawa, S. (1999a). Byte-pair encoding: A text compression scheme that accelerates pattern matching. Technical report, Department of Informatics, Kyushu University, Japan.

Shibata, Y., Matsumoto, T., Takeda, M., Shinohara, A., and Arikawa, S. (2000). A Boyer-More type algorithm for compressed pattern matching. *Proceedings, Combinatorial Pattern Matching, LNCS 1848*, pages 181–194–13.

Shibata, Y., Takeda, M., Shinohara, A., and Arikawa, S. (1999b). Pattern matching in text compressed by using antidictionaries. *Proceedings, Combinatorial Pattern Matching, LNCS 1645*, pages 37–49.

Sikora, T. (1997). The MPEG-4 video standard verification model. *IEEE Transactions on Circuit and Systems for Video Technology*, 7(1).

Sikora, T., Bauer, S., and Makai, B. (1995). Efficiency of shape–adaptive 2D transforms for coding of arbitrarily shaped image segments. *IEEE Transactions on Circuits and Systems for Video Technolog*, 5:254–258.

Silberg, E. and Etzion, O. (1997). The metacrawler architecture for resource aggregation on the web. *IEEE Expert*, 12(1):11–14.

Sleator, D. and Tarjan, R. (1985). Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208.

Sloan, A. D. (1994). Retrieving database contents by image recognition: New fractal power. *Advanced Imaging*, 9(5):26–30.

Smeulders, A. W. M., Worring, M., Santini, S., Gupter, A., and Jain, R. (2000). Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intellingence*, 22(12):1349–1380.

Smith, J. and Chang, S.-F. (1994a). Quad-tree segmentation for texture-based image query. In *Proceedings of the Second ACM International Conference on Multimedia (MULTIMEDIA '94)*, pages 279–286, New York. ACM Press.

Smith, J. R. and Chang, S. F. (1994b). Transform features for texture classification and discrimination in large image databases. *Proceedings, IEEE International Conference on Image Processing*, 3:407–411.

Song, J. and Yeo, B. L. (1999). Fast extraction of spatially reduced image sequences from MPEG-2 compressed video. *IEEE Transactions on Circuits and Systems for Video Technolog*, 9(7):1100–1114.

Steinberg, Y. and Gutman, M. (1993). An algorithm for source coding subject to a fidelity criterion, based on string pattern matching. *IEEE Transactions on Information Theory*, 39:877–886.

Storer, J. (1988). *Data Compression: Methods and theory.* Computer Science Press, Rockville, Maryland.

Storer, J., editor (1992). *Image and Text Compression.* Kluwer Academic, Norwell, Massachusetts.

Storer, J. and Cohn, M., editors (1992). *Proc. IEEE Data Compression Conference*, Snowbird, Utah. IEEE Computer Society Press, Los Alamitos, California.

Storer, J. and Cohn, M., editors (1993). *Proc. IEEE Data Compression Conference.* IEEE Computer Society Press, Los Alamitos, California, Snowbird, Utah.

Storer, J. and Cohn, M., editors (1994). *Proc. IEEE Data Compression Conference.* IEEE Computer Society Press, Los Alamitos, California, Snowbird, Utah.

Storer, J. and Cohn, M., editors (1997). *Proc. IEEE Data Compression Conference*, Snowbird, Utah. IEEE Computer Society Press, Los Alamitos, California.

Storer, J. and Cohn, M., editors (1998). *Proc. IEEE Data Compression Conference*, Snowbird, Utah. IEEE Computer Society Press, Los Alamitos, California.

Storer, J. and Reif, J., editors (1991). *Proc. IEEE Data Compression Conference*, Snowbird, Utah. IEEE Computer Society Press, Los Alamitos, California.

Storer, J. and Syzmanski, T. (1982). Data compression via textual substitution. *Journal of the ACM*, 29:928–951.

Storer, J. A. (2000). Special issue on lossless data compression. *Proceedings of the IEEE*, 88(11):1685–1809.

Sutinen, E. and Tarhio, J. (1996). Filtration with q-samples in approximate string matching. *Proceedings, Combinatorial Pattern Matching, LNCS 1075*, pages 50–63.

Swanson, M. D., Hosur, S., and Tewfik, A. H. (1996). Image coding for content-based retrieval. *Proceedings of SPIE: VCIP*, 2727:4–15.

Szpankowski, W. (1993). Asympotic properties of data compression and suffix trees. *IEEE Transactions on Information Theory*, 39:1647–1659.

Takaoka, T. (1994). Approximate pattern matching with samples. In *Lecture Notes in Computer Science, Springer-Verlag*, pages 234–242.

Takaoka, T. (1996). Approximate pattern matching with grey scale values. In *Proceedings, CATS 96 (Computing: the Australasian Theory Symposium)*, pages 196–203.

Topiwala, P. N., editor (1998). *Wavelet Image and Video Compression*. Kluwa Academic Publishers.

TREC (2000). Official webpage for TREC – Text REtrieval Conference series. http://trec.nist.gov.

Tsai, W. S. and Yu, S. S. (1985). Atributed string matching with merging for shape recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(4):453–463.

Ukkonen, E. (1985). Finding approximate patterns in strings. *Journal of Algorithms*, 6:132–137.

Vasconcelos, N. and Lippman, A. (1997). Library-based coding: a representation for efficient video compression and retrieval. In (Storer and Cohn, 1997), pages 121–130.

Vellaikal, A., Kuo, C. C. J., and Dao, S. (1995). Content-based retrieval of remote-sensed images using vector quantization. *Proceedings of SPIE*, 2488:178–189.

Veltkamp, R. C. and Tanase, M. (2000). Content-based image retrieval systems: A survey. Technical Report, Department of Computer Science, Utretch University, The Netherlands. *Technical Report*.

Wagner, A. and Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM*, 21:168–173.

Wallace, G. (1991). The JPEG still picture compression standard. *Communications of the ACM*, 34(4):31–44.

Wang, H. and Chang, S.-F. (1995). Adaptive image matching in the subband domain. In *Proceedings, SPIE/IEEE Visual Communications and Image Processing '96*.

Waterman, M. S. (1989). *Mathematical Methods for DNA Sequences*. CRC Press, Boca Raton, Florida.

Wei, G., Li, D., and Sethi, I. K. (1998). Web-wise: compressed image retrieval. *Proceedings, MINAR'98: Multimedia Information Analysis and Retrieval, LNCS 1464*, pages 33–46.

Weinberger, M., Seroussi, G., and Sapiro, G. (2000). The LOCO-I lossless image compression algorithm: Principles and standandization into JPEG-LS. *IEEE Transactions on Image Processing*, 9(8):1309–1326.

Welch, T. (1984). A technique for high performance data compression. *IEEE Computer*, 17:8–20.

Wickerhauser, M. V. (1994). Two fast approximate wavelet algorithms for image processing, classification, and recognition. *Optical Engineering*, 33(7):2225–2235. Special issue on Adapted Wavelet Analysis.

Wilson, B., Wilson, S., Varakantam, S., Puramandia, S., and Ananthamurthy, H. M. A. (2000). A memory efficient architecture for compressed-domain feature extraction. *Proc. Workshop on Signal Processing Systems*, pages 305–314.

Witten, I. and Cleary, J. (1983). Picture coding and transmission using adaptive modeling of quad trees. In *Proc. Int. Electrical, Electronics Conf.*, pages 222–225, Toronto.

Witten, I., Neal, R., and Cleary, J. (1987). Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–541.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufman, second edition edition.

Woods, J. W., editor (1985). *Subband Image Coding*. Kluwa Academic Publishers, Armsterdam.

Wu, S. and Manber, U. (1992a). agrep — A fast approximate pattern-matching tool. In USENIX Association, editor, *Proceedings of the Winter 1992 USENIX Conference: January 20 — January 24, 1992, San Francisco, California*, pages 153–162, Berkeley, CA, USA. USENIX.

Wu, S. and Manber, U. (1992b). Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91.

Yang, E. H. and Keiffer, J. (1996). Simple universal lossy data compression schemes derived from Lempel-Ziv algorithm. *IEEE Transactions on Information Theory*, 42:239–245.

Yang, E. H. and Kieffer, J. (1995). On the performance of data compression algorithms based on string pattern matching. *IEEE Transactions on Information Theory*, 41.

Yeo, B. L. and Liu, B. (1996). Rapid scene analysis in compressed video. *IEEE Transactions on Circuits and Systems for Video Technology*, 5(6):533–.

Yoshida, T., Endoh, T., Kawamura, N., and Kato, H. (1992). Image reduction system. US Patent 5,159,468.

Zhang, A., Cheng, B., and Acharya, R. (1995a). An approach to query-by-texture in image database systems. *Proceedings, SPIE*, 2606:339–349.

Zhang, A., Cheng, B., and Acharya, R. (1995b). Texture-based image retrieval using fractal codes. Technical Report 95-19, Department of Computer Science, SUNY Buffalo. Tue, 19 Sep 1995.

Zhang., A., Cheng, B., Acharya, R., and Manon, R. (1996). Comparison of wavelet transfroms and fractal coding in texture-based image retrieval. *Proceedings, SPIE Conference on Visial Data Exploration and Analysis III*, pages 116–125.

Zhang, Y.-Q., Pereira, F., Sikora, T., and Reader, C. (1997). Special Issue on MPEG-4. *IEEE Transactions on Circuit and Systems for Video Technology*, 7(1).

Zhu, R. and Takaoka, T. (1989). A technique for two-dimensional pattern matching. *Communications of the ACM*, 32(9):1110–1120.

Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23:337–343.

Ziv, J. and Lempel, A. (1978). Compression of individual sequences via variable rate coding. *IEEE Transactions on Information Theory*, IT-24:530–536.

Ziviani, N., Moura, E. S., Navarro, G., and Baeza-Yates, R. (2000). Compression: A key for next generation text retrieval systems. *IEEE Computer*, 33(11):37–44.