# Pattern matching and text compression algorithms

Maxime Crochemore
Gaspard Monge Institute
University of Marne-la-Vallée
France

Thierry Lecroq
Laboratoire d'Informatique de Rouen
University of Rouen
France

Pattern matching is the problem of locating a specific pattern inside raw data. The pattern is usually a collection of strings described in some formal language. Applications require two kinds of solution depending on which string, the pattern or the text, is given first. Solutions based on the use of automata or combinatorial properties of strings are commonly implemented to preprocess the pattern. The notion of indexes realized by trees or automata is used in the second kind of solutions.

The aim of data compression is to provide representation of data in a reduced form in order to both save storage place and save transmission time. There is no loss of information, the compression processes are reversible.

Pattern matching and text compression algorithms are two important subjects in the wider domain of text processing. They apply to the manipulation of texts (word editors), to the storage of textual data (text compression), and to data retrieval systems (full text search). They are basic components used in implementations of practical softwares existing under most operating systems. Moreover, they emphasize programming methods that serve as paradigms in other fields of computer science (system or software design). Finally, they also play an important role in theoretical computer science by providing challenging problems.

Although data are memorized in various ways, text remains the main form to exchange information. This is particularly evident in litterature or linguistics where data are composed of huge corpus and dictionaries. This apply as well to computer science where a large amount of data are stored in linear files. And this is also the case, for instance, in molecular biology because biological molecules can often be approximated as sequences of nucleotides or aminoacids. Furthermore, the quantity of available data in these fields tend to double every eighteen months. This is the reason why algorithms should be efficient even if the speed and capacity of storage of computers increase regularly.

## Pattern matching

When the pattern is a single string the problem is known as string matching: locate all occurrences of a string $x$ of length $m$ in a text $y$ of length $n$. The string and the text are built over the same alphabet $\Sigma$ of size $\sigma$. The naive algorithm locates all occurrences in time $O(nm)$. But hashing provides a simple method that avoids the quadratic number of symbol comparisons in most practical situations, and that runs in linear time under reasonnable probabilistic assumptions (Harrison (1971) and Karp and Rabin (1987)).

The first discovered linear-time string-matching algorithm is from Morris and Pratt (1970). It has been improved by Knuth, Morris, and Pratt (1976). The search behaves like a recognition process by automaton, and a character of the text is compared to a character of the pattern no

more than $\log_\Phi(m+1)$ ($\Phi$ is the golden ratio $(1+\sqrt{5})/2$). Hancart (1993) proves that the delay of a related algorithm discovered by Simon (1994) makes no more than $1+\log_2 m$ comparisons per text symbol.

The Boyer and Moore's algorithm (1977) is considered as the most efficient string-matching algorithm in usual applications. A simplified version of it (or the entire algorithm) is often implemented in text editors for the "search" and "substitute" commands. Cole (1995) proves that the maximum number of symbol comparisons is tightly bounded by $3n$ after the preprocessing.

Several variants of Boyer and Moore's algorithm avoid the quadratic behaviour when searching for all occurrences of the pattern. The most efficient solutions in term of number of symbol comparisons have been designed by Apostolico and Giancarlo (1986), Crochemore et alii (Turbo-BM, 1994), and Colussi (1994). Empirical results show that the variations of Boyer and Moore's algorithm designed by Sunday (Quick Search, 1990) and an algorithm based on the suffix automaton by Crochemore et alii (1994) are the most efficient in practice.

Searching for $k$ patterns by repetitive runs of previous algorithms on the text $y$ gives an overall $O(kn)$ running time. In 1975, Aho and Corasick designed an $O(n \log \sigma)$ algorithm to solve this problem, with a running time independent of the number of patterns. It is implemented by the `fgrep` command under the UNIX operating system.

The notion of a longest common subsequence (LCS) of two strings is widely used to compare files. The `diff` command of UNIX system implement an algorithm based of this notion where lines of the files are considered as symbols. Informally, the result of a comparison gives the minimum number of operations (insert a symbol, or delete a symbol) to transform one string into the other. The comparison of molecular sequences is basically done with a closed concept, alignment of strings, which consists in aligning their symbols on vertical lines. This is related to an edit distance, called the Levenshtein distance, with the additional operation of substitution, and with weights associated to operations. Hirchsberg (1975) presents the computation of the LCS in linear space. This is an important result because the algorithm is used on large sequences.

Approximate string matching consists in finding all approximate occurrences of pattern $x$ in text $y$. Approximate occurrences of $x$ are segments of $y$ that are close to $x$ according to a specific distance: their distance to $x$ must be not greater than a given integer $k$. Two common distances are the Hamming distance and the Levenshtein distance.

With the Hamming distance related to the number of mismatches between the pattern and its approximate occurrences, the problem is also called the approximate string matching with $k$ mismatches. With the Levenshtein distance (or edit distance) the problem is known as the approximate string matching with $k$ differences. The approximate string searching is a lively domain of research. It includes for instance the notion of regular expressions to represent sets of strings. Algorithms based on regular expression are commonly found in book related to compiling techniques. The Shift-Or algorithm by Baeza-Yates and Gonnet (1992), and by Wu and Manber (1992) is a method that is both very fast in practice and very easy to implement. It adapts to the two above problems.

In applications where the text is to be searched for several patterns, it is the text that needs to be preprocessed. Even if no further information is known on their syntactic structure, it is possible and indeed extremely efficient to built an index that supports searches. Data structures to represent indexes on text files are: suffix trees (Weiner 1973, McCreight 1976, Ukkonen 1994), direct acyclic word graph (Blumer et alii, 1985), suffix automata (Crochemore, 1986), and suffix arrays (Manber and Myers, 1993). All algorithms (except for suffix arrays) build the index in time $O(n \log \sigma)$.

## Text compression

The following methods yield two basic data compression algorithms, which produce good compression ratios and run in linear time.

The first strategy is a statistical encoding that takes into account the frequencies of symbols to built a uniquely decipherable code optimal with respect to the compression criterion. Huffman method (1951) provides such an optimal statistical coding. It admits a dynamic version where symbol counting is done at coding time. The command `compact` of UNIX implements this version.

Ziv and Lempel (1977) designed a compression method using encoding segments. These segments are stored in a dictionary that is built during the compression process. When a segment of the dictionary is encountered later while scanning the original text it is substituted by its index in the dictionary. In the model where portions of the text are replaced by pointers on previous occurrences, the Ziv and Lempel's compression scheme can be proved to be asymptotically optimal (on large enough texts satisfying good conditions on the probality distribution of symbols). The dictionary is the central point of the algorithm. Furthermore, a hashing technique makes its implementation efficient. This technique improved by Welch (1984) is implemented by the `compress` command of the UNIX operating system.

The problems and algorithms discussed above give a sample of text processing methods. Several other algorithms improve on their performance when the memory space or the number of processors of a parallel machine are considered for example. Methods also extend to other discrete objects such as trees and images.

## References

Listed below are either books entirely devoted to pattern matching or text compression algorithms, or books on the design of general algorithms that contain a whole chapter on the topic. All references mentionned in the text may be found in these books, in (Crochemore and Rytter, 1994) for example.


Aho, A.V. 1990. Algorithms for finding patterns in strings. In: *Handbook of Theoretical Computer Science, Algorithms and Complexity, Vol. A, ch. 5, pp 255–330.* J. van Leeuwen ed., Elsevier, Amsterdam.

Bell, T.C., Cleary J.G. and Witten, I.H. 1990. *Text compression.* Prentice Hall, Englewood Cliffs, New Jersey.

Cormen, T.H., Leiserson C.E. and Rivest, R.L. 1990. *Introduction to algorithms, ch. 34, pp 853–885.* MIT Press.

Crochemore, M. and Rytter W. 1994. *Text algorithms.* Oxford University Press.

Gonnet, G.H. and Baeza-Yates, R.A. 1991. *Handbook of algorithms and data structures, ch. 7, pp 251–288.* Addison-Wesley.

Nelson, M. 1992. *The data compression book.* M&T Books.

Sedgewick R. 1990. *Algorithms in C, ch. 19 and 22.* Addison-Wesley.

Stephen, G.A. 1994. *String searching algorithms.* World Scientific Press.