6D Classification of Pattern Matching Problems[1]

Bořivoj Melichar, Jan Holub

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University
Karlovo náměstí 13
121 35 Prague 2
Czech Republic

e-mail: {melichar, holub}@cs.felk.cvut.cz

Abstract. We present our unified view to pattern matching problems and their solutions. We classify pattern matching problems by using six criteria and therefore we can locate them into six-dimensional space. We also show basic model of nondeterministic finite automaton that can be used for constructing models for all pattern matching problems.

Key words: string matching, sequence matching, classification, finite automata

1 Introduction

Pattern matching (string and sequence matching) appears as a very important component of many applications, including text editing, word processing, data retrieval, symbol manipulation, alignment in genetics, etc. This problem has been extensively studied since beginning of seventies.

The exact string matching is based on the two historical papers by Knuth, Morris and Pratt [KMP77] and by Boyer and Moore [BM77]. Matching of sequences started by Chvátal, Klainer and Knuth [CKK72]. They designed an algorithm for the longest common subsequence problem. Approximate string matching is based on well known paper by Wagner and Fischer [WF74]. Multiple string matching was originated by Aho and Corasick [AC75].

Since this first algorithms, many different problems of pattern matching were studied and many excellent ideas are included in existing text algorithms [CR94].

All one-dimensional pattern matching problems are sequential problems and therefore it is possible to solve them using finite automata. Below we discuss construction of finite automata for many pattern matching problems. These automata are mostly nondeterministic. There are three ways how these automata can be used:

1. To serve as a model of algorithms for solving of different problems.
2. To simulate the nondeterministic automaton in a deterministic way. Some of known pattern matching algorithms use this approach.

[1] This work was supported by grant FRVŠ 0892/97.

3. To construct an equivalent deterministic finite automaton. This approach may lead to the high space complexity in some cases.

The use of finite automata for the modelling of pattern matching algorithms means, that there is a formal method introduced to this part of computer science. It is well known from other areas (e.g. language theory) that introduction of formal approach has positive consequences. The main advantage is that it is possible to describe all problems using an unified view. This leads to the possibility to transfer know-how from another, well developed area, to compare different solutions, to find limitations, etc.

One of the consequences of the introduction of finite automata formalism is:

The possibility of the construction of finite automata for all problems in question shows, that there exist algorithms for all pattern matching problems having the linear time complexity. The space complexity is different for different problems. Existing upper bounds on space complexity of some problems are pessimistic.

Evaluation of some existing algorithms as simulators of nondeterministic finite automata will lead to understanding how some classes of automata can be simulated. This knowledge may serve for improvement of other algorithms and for the design of new ones.

2 Pattern matching problems

2.1 Basic notions and notations

Some basic notions will be used in subsequent sections. This section collects definitions of them. The notion pattern matching is used for string matching and sequence matching.

"Don't care" symbol is a special universal symbol Ø that matches any other symbol including itself.

Definition 1 (Basic pattern matching problems)
Given a text string $T = t_1 t_2 \cdots t_n$ and a pattern $P = p_1 p_2 \cdots p_m$. Then we may define:

1. String matching: verify if string P is a substring of text T.

2. Sequence matching: verify if sequence P is a subsequence of text T.

3. Subpattern matching: verify if a subpattern of P (substring or subsequence) occurs in text T.

4. Approximate pattern matching: verify if pattern P occurs in the text T so that the distance $D(P, X) \leq k$ for given $k < m$, where $X = t_i \cdots t_j$ is a part of text T.

5. Pattern matching with "don't care" symbols: verify if pattern P containing "don't care" symbols occurs in text T.

Definition 2 (Matching a sequence of patterns)
Given a text string $T = t_1 t_2 \cdots t_n$ and a sequence of patterns (string and sequences)

$P_1, P_2, \cdots, P_s$. Matching of sequence of patterns $P_1, P_2, \cdots, P_s$ is a verification whether an occurrence of pattern P_i in text T is followed by an occurrence of P_{i+1}, $1 \leq i < s$.

Definitions 1 and 2 define pattern matching problems as a decision problems, because the output is a Boolean value. A modified version of these problems consists in searching for the first, the last, or all occurrences of pattern and moreover the result may be the set of positions of the pattern in the text. Instead of just one pattern, one can consider a finite or infinite set of patterns.

Definition 3 (Distance of patterns)
Three variants of distances between two patterns X and Y are defined as minimal number of editing operations:

1. *replace* (Hamming distance, R-distance),
2. *delete, insert* and *replace* (Levenshtein distance, DIR-distance),
3. *delete, insert, replace* and *transpose* (generalized Levenshtein distance, $DIRT$-distance),

needed to convert pattern X into pattern Y.

The Hamming distance is a metrics on the set of string of equal length. The Levenshtein and the generalized Levenshtein distances are metrics on the set of strings not necessarily of equal length.

2.2 Classification of pattern matching problems

One-dimensional pattern matching problems for a finite size alphabet can be classified according to several criteria. We will use six criteria for classification leading to six-dimensional space in which one point corresponds to particular pattern matching problem.

Let us make a list of all dimensions including possible "values" in each dimension:

1. Nature of the pattern: string, sequence.
2. Integrity of the pattern: full pattern, subpattern.
3. Number of patterns: one, finite number, infinite number.
4. The way of matching: exact, approximate matching with Hamming distance (R-matching), approximate matching with Levenshtein distance (DIR-matching), approximate matching with generalized Levenshtein distance ($DIRT$-matching).
5. Importance of symbols in pattern: take care of all symbols, don't care of some symbols.
6. Number of instances of pattern: one, finite sequence.

The above classification is visualized in Figure 1. If we count the number of possible pattern matching problems, we obtain $N = 2 * 2 * 3 * 4 * 2 * 2 = 192$.

In order to make references to particular pattern matching problem easy, we will use abbreviations for all problems. These abbreviations are summarized in Table 1.

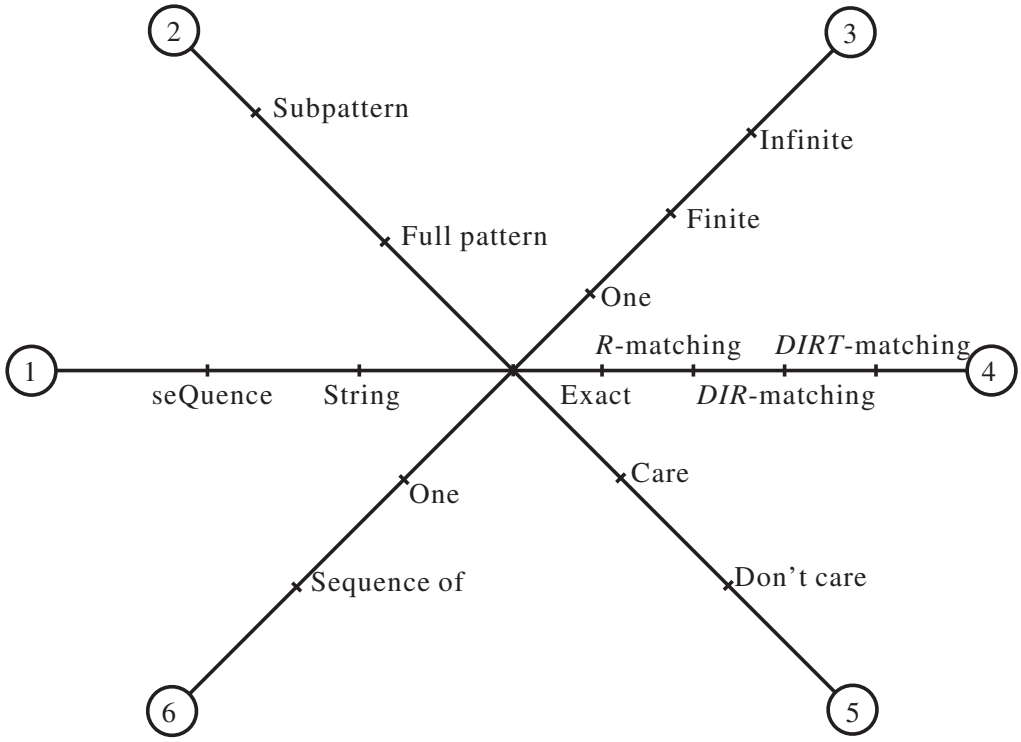

Figure 1: Classification of pattern matching problems.

Dimension	1	2	3	4	5	6
	S	F	O	E	C	O
	Q	S	F	R	D	S
			I	D		
				G		

Table 1: Abbreviations of pattern matching problems.

Using this method, we can, for example, refer to exact string matching of one string as $SFOECO$ problem.

Instead of single pattern matching problem we will use the notion of family of pattern matching problems. In this case we will use symbol '?' instead of particular letter. For example SFO??? is the family of all problems concerning one full string matching.

Each of pattern matching problem can have several instances. For example, $SFOECO$ problem can have the following instances:

1. verify whether given string occurs in text or not,
2. find the first occurrence of given string,
3. find the number of all occurrences of given string,
4. find all occurrences of given string and where they are.

If we take into account all possible instances, the number of pattern matching problems is further growing.

2.3 Pattern matching algorithms

Many algorithms for pattern matching problems were designed using ad hoc approach. But pattern matching problems are sequential problems and therefore it is possible to solve them using finite automata. There is possible to use systematic approach and to create a model of algorithm for each pattern matching problem. This model is nondeterministic finite automaton (*NFA*) in all cases.

We can convert an *NFA* to a deterministic finite automaton (*DFA*) and run it using the text as an input. If we suppose a finite size alphabet, then the running time of *DFA* is $\mathcal{O}(n)$, where n is the length of text.

It is well known, that the *NFA* to *DFA* conversion requires at most $\mathcal{O}(2^m)$ time, where m is the number of states of the *NFA*. The resulting *DFA* may have at most $\mathcal{O}(2^m)$ states.

Latest investigation shows, that this bound does not take place in the area of pattern matching problems [Me95], [Me96]. The bound of number of states of *DFA* is much lower and therefore this approach may have practical applications. Instead of conversion of *NFA* to *DFA*, we can simulate the *NFA* in a deterministic way. Some of known pattern matching algorithms use this approach. The problem of this approach is a high time complexity while the space complexity is low. In case of approximate string matching, "Shift-Or" based algorithms [BG92], [WM92] use this approach as it is shown in [Me95], [Me96].

3 Models of pattern matching algorithms

We will show, in this section, basic model of pattern matching algorithms. Moreover, we will show how to construct some models for more complicated problems using models of simple problems.

3.1 Exact string and sequence matching

In Fig. 1 the basic model of pattern matching algorithms is represented by the circle that meets all the axes in the points which are the closest to the junction of the axes. This model is the model for exact string matching ($SFOECO$ problem). For pattern $P = p_1p_2p_3p_4$ this model is shown in Fig. 2. The $SFOECO$ automaton has $m + 1$ states for the pattern of the length m. This *NFA* can be transformed to *DFA* which has the same number of states as its nondeterministic version. The transformation can be performed in time $\mathcal{O}(m)$.

If we add loops labeled by mismatching characters into states, from which there leads at least one edge, in the model for string matching we obtain a corresponding model for sequence matching. The model of algorithm for exact sequence matching ($QFOECO$ problem) for pattern $P = p_1p_2p_3p_4$ is shown in Fig. 3. Character $\overline{p}$ represents any character mismatching character p. The $QFOECO$ automaton has $m + 1$ states for the pattern of length m.

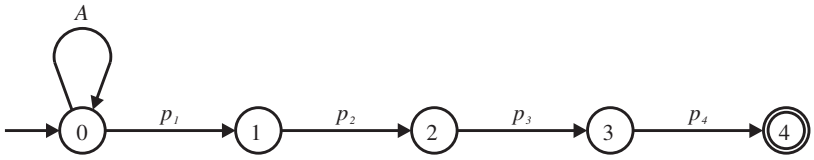

Figure 2: *NFA* for exact string matching ($SFOECO$ automaton)

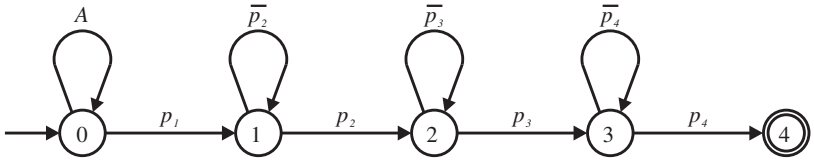

Figure 3: *NFA* for exact sequence matching ($QFOECO$ automaton).

3.2 Substring and subsequence matching

The model of algorithm for exact substring matching ($SSOECO$ problem) for pattern $P = p_1p_2p_3p_4$ is shown in Fig. 4. We can see that this automaton has been created by connecting m $SFOECO$ automata. The $SSOECO$ automaton has $(m+1)+m+(m-1)+\cdots+2 = \frac{m(m+3)}{2}$ states and is called an *initial ε-treelis*.

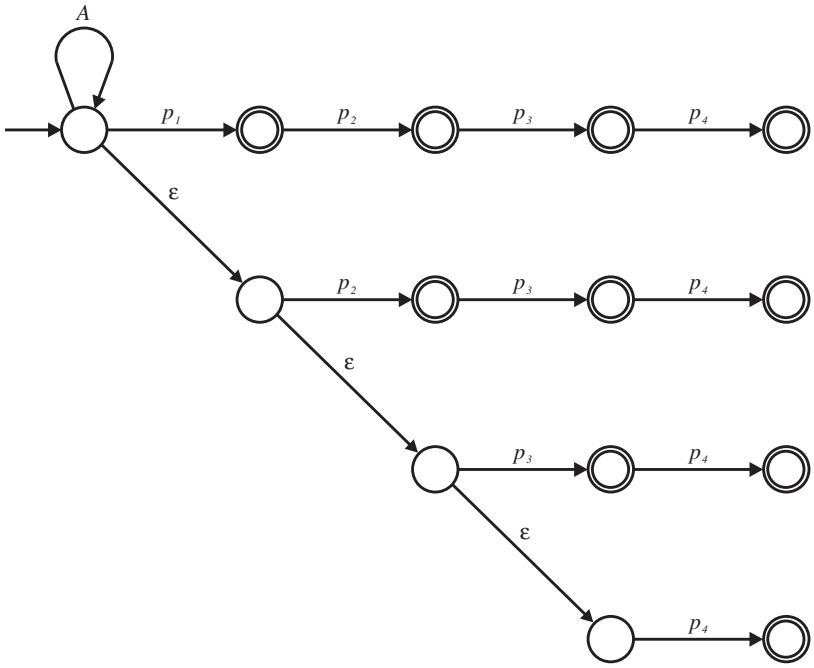

Figure 4: *NFA* for exact substring matching ($SSOECO$ automaton).

The model of algorithm for exact subsequence matching ($QSOECO$ problem) is similar as for exact substring matching. We get this model from $SSOECO$ model by adding loops for mismatching characters and ε-transitions into all states from which just one transition leads. Each ε-transition leads from state q_i to the state q_j such that from state, which is just under state q_i, a matching transition leads to state q_j. This automaton can be also constructed by connecting m $QFOECO$ automata. The $QSOECO$ automaton has $\frac{m(m+3)}{2}$ states and is called *ε-treelis*.

3.3 Approximate string matching

We will discuss three variants of approximate string matching corresponding to three definitions of distances between strings: Hamming distance, Levenshtein distance, and generalized Levenshtein distance.

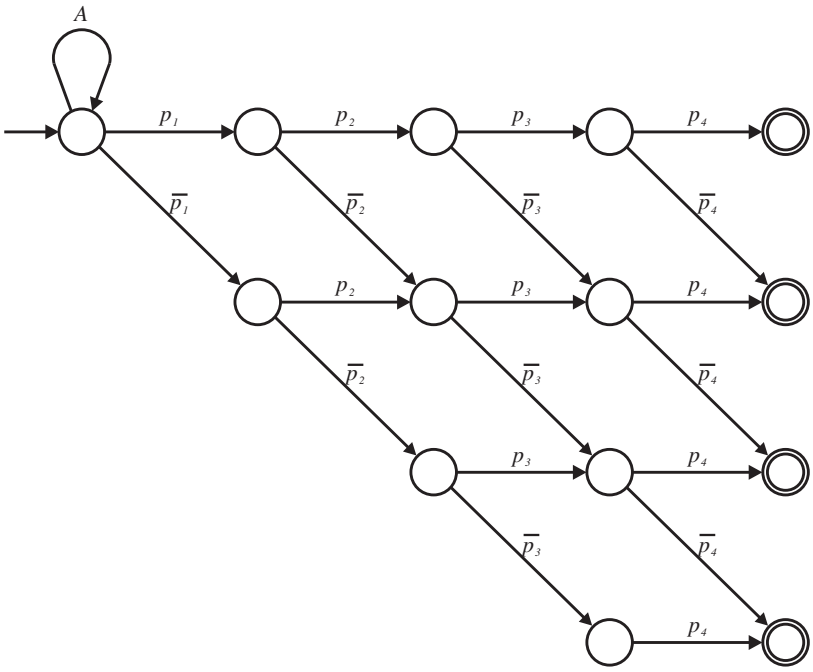

Figure 5: *NFA* for string R-matching ($SFORCO$ automaton).

Hamming distance Let us note, that Hamming distance between strings x and y is equal to the minimal number of editing operations *replace* which are necessary to convert string x into string y. Therefore this type of string matching is called string R-matching. The model of algorithm for string R-matching ($SFORCO$ problem) was presented in [Me95] and in Fig. 5 it is shown for string $P = p_1p_2p_3p_4$ and Hamming distance $k = 3$.. This automaton has been created by connecting $k + 1$ $SFOECO$ automata by edges that represent editing operation *replace*. The $SFORCO$ automaton has $(m + 1) + m + (m - 1) + \cdots + (m - k + 1) = (k + 1)(m + 1 - \frac{k}{2})$ states. This automaton is called $R - treelis$.

Levenshtein distance Let us note, that Levenshtein distance between strings x and y is equal to the minimal number of editing operations *delete*, *insert* and *replace* which are necessary to convert string x into string y. Therefore this type of string matching is called string DIR-matching. The model of algorithm for string DIR-matching ($SFODCO$ problem) was presented in [Me96], [Ho96] and in Fig. 6 it is shown for string $P = p_1p_2p_3p_4$. It has been created from $SFORCO$ model by adding edges representing editing operations *insert* and *delete*.

Generalized Levenshtein distance Let us note, that generalized Levenshtein distance between strings x and y is equal to the minimal number of editing operations *delete*, *insert*, *replace* and *transpose* which are necessary to convert string x into string y. Therefore this type of string matching is called string $DIRT$-matching. The model of algorithm for string $DIRT$-matching ($SFOGCO$ problem) for string $P = p_1p_2p_3p_4$

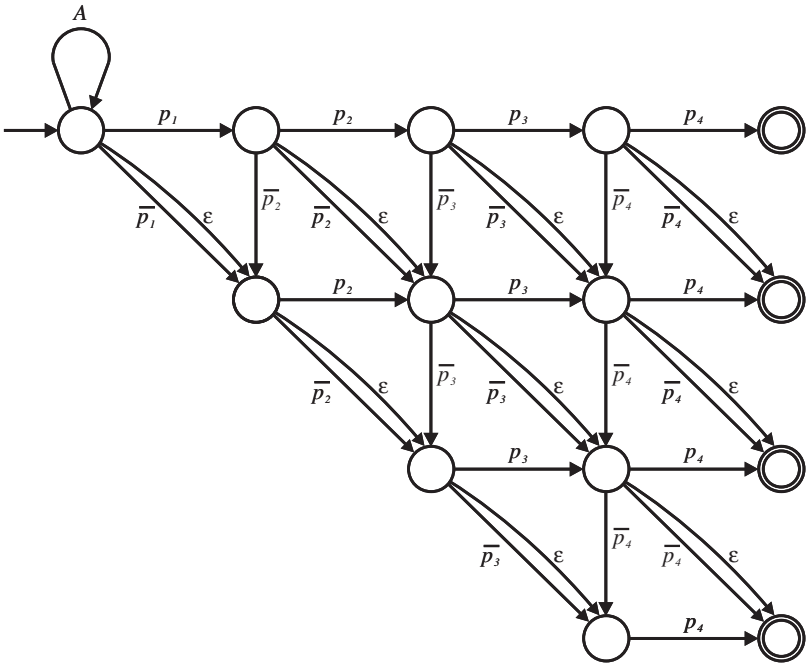

Figure 6: *NFA* for string *DIR*-matching ($SFODCO$ automaton).

is shown in Fig. 7. It has been constructed from $SFODCO$ model by adding states for editing operation *transpose* and corresponding edges.

Conclusion

We have presented unified view to pattern matching. We have also shown the basic model for pattern matching and several methods of constructing other models. These models can be very useful in designing new methods or improving existing methods for pattern matching. Since for each pattern matching problem there exists *NFA* there exists algorithm running in linear time for each such problem.

References

[AC75] Aho, A. V., Corasick, M. J.: Efficient String Matching: An Aid to Bibliographic Search. CACM, Vol. 18, No. 6, pp. 333–340, 1975.

[BG92] Baeza-Yates, R., Gonnet, G. H.: A New Approach to Text Searching. Communications of the ACM, October 1992, Vol. 35, No. 10, pp. 74–82.

[BM77] Boyer, R. S., Moore, J. S.: A Fast String Searching Algorithm. Commun. ACM, Vol. 20, No. 10, October 1977, pp. 762–772.

[CKK72] Chvátal, V., Klarner, D. A., Knuth, D. E.: Selected Combinatorial Research Problems. STAN-CS-72-292, Stanford University, June 1972, 26.

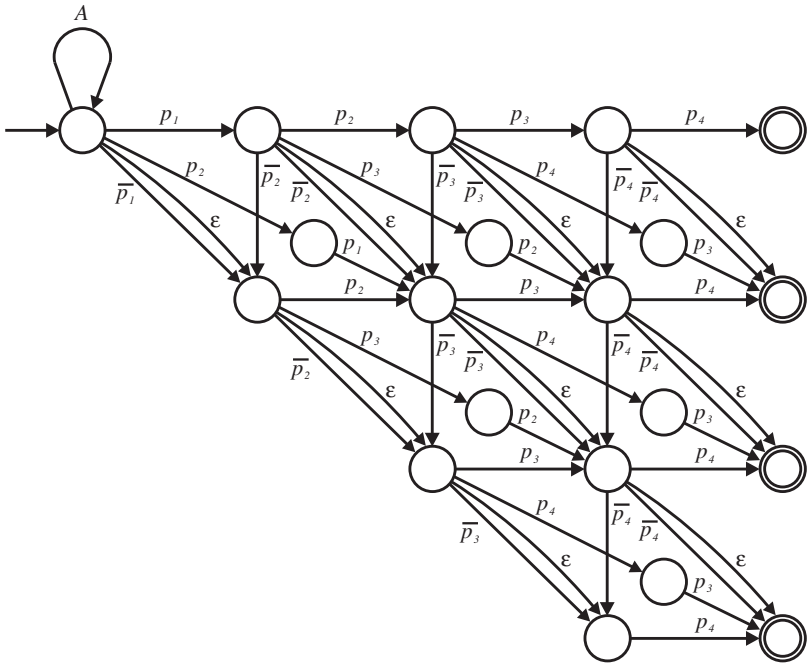

Figure 7: *NFA* for string $DIRT$-matching ($SFOGCO$ automaton).

[CR94] Crochemore, M., Rytter, W.: Text Algorithms. Oxford University Press, New York 1994, p. 414.

[Ho96] Holub, J.: Reduced Nondeterministic Finite Automata for Approximate String Matching. Proceedings of the Prague Stringology Club Workshop '96, Czech Technical University, August 1996, pp. 19–27.

[KMP77] Knuth, D. E., Morris, J. H., Pratt, V. R.: Fast Pattern Matching in Strings, SIAM J. Comput., Vol. 6, No. 2, June 1977, pp. 322–350.

[Me95] Melichar, B.: Approximate String Matching by Finite Automata. Computer Analysis of Images and Patterns. LNCS 970, Springer 1995, pp. 342–349.

[Me96] Melichar, B.: String Matching with k Differences by Finite Automata. Proceedings of the 13^{th} ICPR, Vol. II, August 1996, pp. 256–260.

[WF74] Wagner, R. A., Fisher, M. J.: The String-to-String Correction Problem. Journal of ACM, January 1974, Vol. 21, No. 1, pp. 168–173.

[WM92] Wu, S., Manber, U.: Fast Text Searching Allowing Errors. Communications of the ACM, October 1992, Vol. 35, No. 10, pp. 83–91.