Milan Šimánek
The factor automaton

Kybernetika, Vol. 38 (2002), No. 1, [105]--111

Persistent URL: http://dml.cz/dmlcz/135449

KYBERNETIKA — VOLUME 38 (2002), NUMBER 1, PAGES 105 - 111

THE FACTOR AUTOMATON[1]

MILAN ŠIMÁNEK

This paper concerns searching substrings in a string using the factor automaton. The factor automaton is a deterministic finite automaton constructed to accept every substring of the given string. Nondeterministic factor automaton is used to achieve new operations on factor automata for searching in non-constant texts.

1. INTRODUCTION

The amount of information processed by computers grows very dramatically. Now every application should contain an information searching engine. Let us deal with exact substring matching for one pattern–text pair. There are a lot of pattern matching algorithms with the same function but each of them is optimal in different field.

Standard algorithms (Aho–Corasick or Boyer–Moore) sequentially compare given pattern with a text to be searched in. These algorithms are relatively slow but they need not any precomputed data structures and have small memory complexity. Time complexity is typically from $O(n+m)$ to $O(n*m)$ where n is the size of the searching text and m is the size of the pattern. The space complexity is equal to the size of the text and the pattern. Algorithms in this category are useful for applications with small texts or in applications which source text changes too often.

Another category of searching algorithms prefer preprocessing the base text and extremely fast searching in special data structures. The most useful data structure is directed acyclic word graph sometimes called DAWG. In this case the searching engine is a finite automaton with the same structure as DAWG has. The space complexity is also $O(n)$ but implementation of this automaton is approximately 4 times larger than algorithms in previous category. The main benefit is the fastest possible searching speed. The time complexity is linear with respect to the size of the pattern, not depending on the size of the searched text. However, the source text must be preprocessed. The preprocessing is an operation with the time and space complexity linear with respect to the size of the text. Any change of the source text causes a new reinitialization of the data structure (or structure of the finite automaton). It seems the cost of this reinitialization is too high but recently the

[1]This research is supported by Grant 201/98/1155 of the Grant Agency of the Czech Republic.

algorithms has been found which can modify internal data structure (or the finite automaton structure) according to changes of the source text. It seems that this algorithms are fast enough if the difference in source text is enough simple.

Algorithms in this category are useful in applications with constant source text with special emphasis on speed (library searching tools, genetics, etc.). They may be used also with non constant source text in conjunction with algorithms changing their internal data structures.

2. THE FACTOR AUTOMATON

The factor automaton is a finite automaton accepting the set of all substrings of the string. The set of all substrings (factors) of the string *text* is Fac(*text*). Each factor automaton is related to some string *text*. The language accepted by automaton is Fac(*text*).

The factor automaton is basically nondeterministic. It is very simple and there is simple formal description of the automaton. The main disadvantage is that it cannot be directly implemented. It can be only simulated but simulation lose the searching speed.

A theorem from automata theory says that each nondeterministic automaton can be converted into the deterministic one with possible exponential grow of the size. The factor automaton also can be converted to the deterministic form but the grow of the size stay linear because the language of the automaton is always a set of related strings (strings are the substrings of the source text). Deterministic factor automaton is suitable for implementation.

2.1. Nondeterministic factor automaton

We can get the nondeterministic automaton accepting all substrings of the string $a_1a_2 \ldots a_n$ from simple automaton accepting only one string $a_1a_2 \ldots a_n$ by adding epsilon transitions from the initial state to all other states and making all states final.

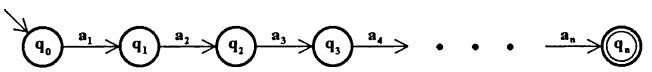

Fig. 1. Finite automaton accepting string $a_1a_2a_3 \ldots a_n$.

The number of states of the nondeterministic factor automaton for string $S = a_1a_2a_3 \ldots a_n$ is $n + 1$. Total number of transitions in the automaton is $2n$. There is only one state with nondeterministic transitions – the initial state q_0. There are possible several modifications of this automaton. It may have no epsilon transitions but all of the states may be initial states. It means that the automaton is configured in all states before start of pattern matching. Another modification is replacing epsilon transitions by regular ones.

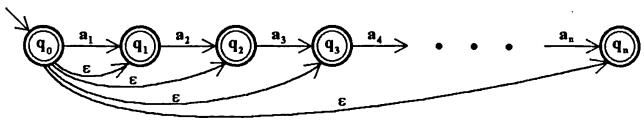

Fig. 2. Nondeterministic factor automaton.

2.1.1. Formal description

Let $M = (\Sigma, Q, \delta, I, F)$ is nondeterministic finite automaton, Σ is the alphabet, Q is the set of states, δ is transition function $\delta : (q, a) \mapsto P$ where $q \in Q$, $a \in \Sigma$ and $P \subset Q$. I is the initial set of states and F is the set of final states. The configuration of the automaton is a pair $(P_i, s_1 s_2 \dots s_k)$ where $P_i \subset Q$ is the set of active states and $s_1 s_2 \dots s_k$ is the string to be read. Automaton pass from the configuration $(P_i, s_1 s_2 s_3 \dots s_k)$ to the configuration $(P_j, s_2 s_3 \dots s_k)$ iff $P_j = \bigcup_{q \in P_i} \delta(q, s_1)$.

The nondeterministic factor automaton for string $S = a_1 a_2 a_3 \dots a_n$ is the nondeterministic finite automaton with $n+1$ states $Q = \{q_0, q_1, q_2, \dots, q_n\}$. The alphabet Σ is the same as the alphabet of the searching string. The transition function δ is defined as

1. $\delta(q_i, a_{i+1}) = \{q_{i+1}\}$ for all $i \geq 0$,
2. $\delta(q_0, \varepsilon) = \{q_i; q_i \in Q,\ q_i \neq q_0\}$,
3. $\delta(q, a) = \emptyset$ otherwise.

$I = \{q_0\}$ is the set containing only one initial state[2] and F is the set of all states $F = Q$ because all states are final.

The automaton being configured in set $\alpha = \{q_{s_1}, q_{s_2}, q_{s_3}, \dots q_{s_i}\}$ of the active states ($\alpha \subset Q$ and $0 \leq i \leq n$) after reading pattern $P = p_1 p_2 p_3 \dots p_j$ means that pattern P occurs in string S i-times at positions $s_1 - j$, $s_2 - j$, $s_3 - j, \dots s_i - j$. This nondeterministic factor automaton provides parallel pattern matching at all possible positions in string S at the same time.

2.2. Deterministic factor automaton

Nondeterministic factor automaton can be split into n chains (one chain per each epsilon transition). First transition of each chain is an epsilon transition starting in the initial state q_0 and leading into a state with only one successor. These transitions can be replaced by regular ones (see Figure 3).

This tree factor automaton is still nondeterministic in the initial state because any two characters in string S may be equal. If the string S contain two or more same characters then the starting part of the corresponding chains can be joined in common prefix (see example on Figure 4). This automaton is deterministic while its searching speed remains the fastest possible. Data structure of this automaton is called suffix tree.

[2]Sometimes it is useful to cut out epsilon transitions and to define the initial set of states as the set of all states $I = Q$.

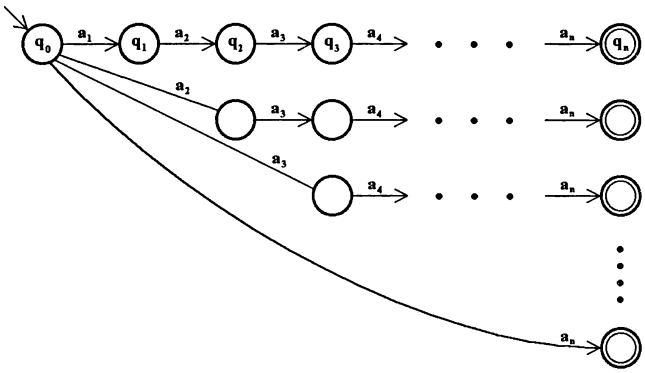

Fig. 3. Nondeterministic tree factor automaton.

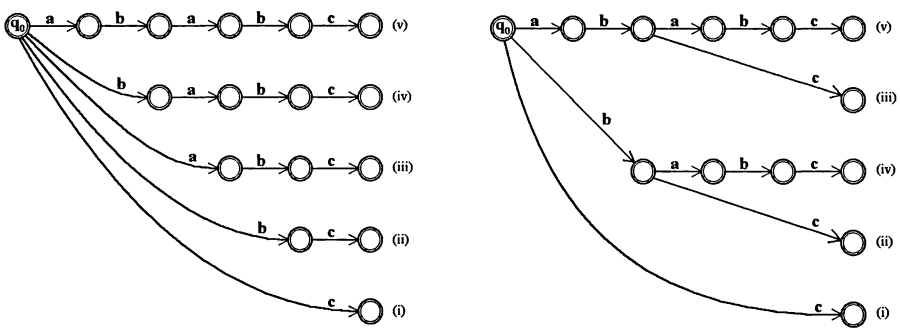

Fig. 4. Example of nondeterministic tree and deterministic suffix tree factor automaton for string *ababc*.

The suffix tree factor automaton has optimal time complexity but it is unnecessary large. Each chain is a suffix of the main (the longest) chain. Isomorphic subtrees (branches) can be joined together. The result is acyclic graph representing deterministic factor automaton so-called DAWG (see example on Figure 5).

Another way to get a deterministic version of the factor automaton is to construct all accessible subsets of states of the nondeterministic factor automaton [3]. Total number of all subsets of the states is 2^n but only a few of them are accessible from the initial state. Each subset of states of the nondeterministic automaton relates to one state of the deterministic automaton. However, the initial state of the deterministic automaton relates to the set of all states of the nondeterministic automaton (because of epsilon transitions).

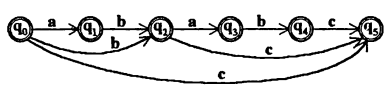

Fig. 5. Example of directed acyclic word graph (DAWG) for $S = ababc$.

Both direct construction methods are equivalent because the results are isomorphic and the constructions have asymptotically same complexity. Another type of construction is offered in [1] and [2].

The number of states of the deterministic factor automaton is greater then n and less than $2n$. Total number of transitions of the deterministic automaton is greater or equal to n and less then $3n$. The proof is in [1, Theorem 6.1].

2.3. The fail function

Let $Fail(q)$ be a fail function defined for each state q of the deterministic factor automaton which is used by some algorithms. The value of this function is a state with this characteristics:

$Fail(q_i) = q_j$ iff $\exists uv \in Fac(S)$ that $\delta^*(q_0, uv) = q_i$ and $\delta^*(q_0, v) = q_j$ and v is the longest suffix of string uv for which $q_i \neq q_j$ (see Figure 6).

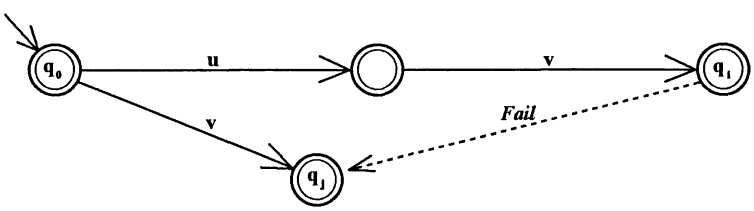

Fig. 6. The fail function.

This means that the factor automaton being in state q_j accepts each suffix which is accepted in state q_i.

3. OPERATIONS ON FACTOR AUTOMATON

An operation on the factor automaton is an operation which make some changes to the factor automaton in order to reflect the changes in searching string.

We can define elementary operations on factor automata for all elementary operations on string. The table below collects basic operations on factor automaton. Each operation has an implicit argument factor automaton for old string and it may have other arguments. The result of the operation is factor automaton for new string.

Operation	Argument	Old string	New string
append	$c \in \Sigma$	$a_1a_2a_3 \dots a_n$	$a_1a_2a_3 \dots a_nc$
insert	$c \in \Sigma$	$a_1a_2a_3 \dots a_n$	$ca_1a_2a_3 \dots a_n$
$R-$ *delete*	—	$a_1a_2a_3 \dots a_{n-1}a_n$	$a_1a_2a_3 \dots a_{n-1}$
$L-$ *delete*	—	$a_1a_2a_3 \dots a_n$	$a_2a_3 \dots a_n$
replace	$c \in \Sigma$, $1 \leq i \leq n$	$a_1 \dots a_{i-1}a_ia_{i+1} \dots a_n$	$a_1 \dots a_{i-1}ca_{i+1} \dots a_n$
join	$b_1b_2 \dots b_m \in \Sigma^*$	$a_1a_2 \dots a_n$	$a_1a_2 \dots a_nb_1b_2 \dots b_m$

4. ON-LINE CONSTRUCTION

The deterministic factor automaton can be constructed by operation *append* from a trivial factor automaton for the empty string accepting only an empty string. Let $DAWG(S)$ be a deterministic factor automaton accepting the set all substrings of the string S. Using operation *append* we can construct $DAWG(S)$ for $S = a_1a_2a_3 \dots a_n$ step by step:

$$DAWG(\varepsilon) \longrightarrow DAWG(a_1) \longrightarrow DAWG(a_1a_2) \longrightarrow \dots \longrightarrow DAWG(S).$$

This method of construction is discussed in [1, 6.3 On-line construction]. The time complexity of construction $DAWG(S)$ is n so that this algorithm has linear time complexity. This algorithm uses fail function.

5. LZ77 AND SLIDING WINDOW

One of the famous compression algorithms is known as LZ77. Its working memory is split into two regions – the sliding window and the buffer (see Figure 7).

The main part of this compression method scans the text in the sliding window for a pattern contained in buffer. The longest prefix of the buffer found in the sliding window is used for compression. After each search operation the matching prefix of the buffer will be encoded to a single character. The sliding window will be shifted to the left and buffer will be filled in from input stream. Then the next searching operation will continue until the end of the input stream.

The size of the sliding window is constant in range of a few thousands bytes while the size of the buffer is only a few characters. The searching speed of the standard implementation is linear with respect to the size of the sliding window.

We can speed up the search operation applying the factor automaton. The pair of *L-delete* and *append* operation enables fast moving of the sliding window without recomputation of the factor automaton (see Figure 8).

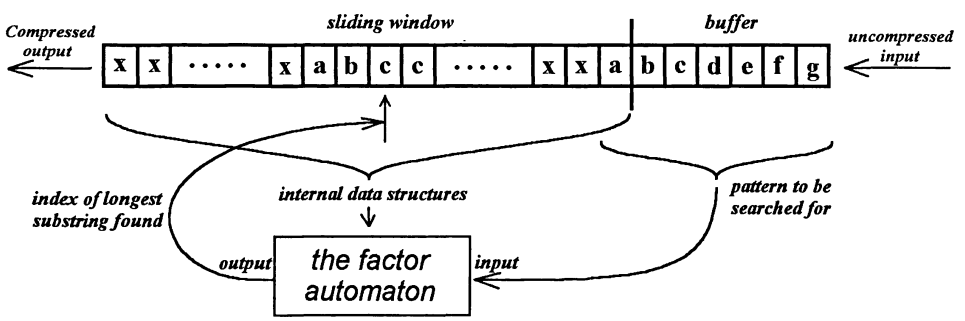

Fig. 7. The LZ77 compression algorithm.

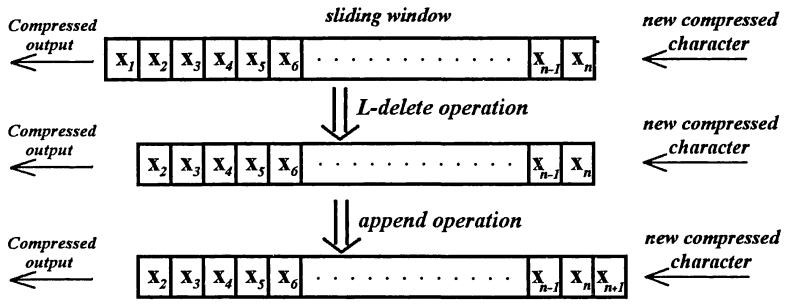

Fig. 8. Recomputation of the sliding window.

At first we apply k-times operation *append* to get a deterministic factor automaton for sliding window with size k characters. Then we will apply repetitively a couple of operations *L-delete* and *append* and perform the searching operation by factor automaton. We will get a moving window for fast searching in this part of the text. The speed of searching is independent on size of the searching window and depends only on the size of pattern looking for. The recomputation of factor automaton (operations *L-delete* and *append*) take in average constant time per one shift.

The main part of this algorithm has a linear-time complexity but the time complexity is constant independent of the size of sliding window. It seems that the speed up is essential. Moreover the size of sliding window can be increased to get better compression while the time of the compression stay the same.

(Received May 12, 2000.)

REFERENCES

[1] M. Crochemore and W. Rytter: Text Algorithms, Chapter 6, Subword graphs. Oxford University Press, Oxford 1994.

[2] M. T. Chen and J. Seiferas: Efficient and elegant subword tree construction. In: Combinatorial Algorithms on Words, NATO Advanced Science Institutes, Series F, vol. 12, Springer–Verlag, Berlin 1985, pp. 97–107

[3] B. Melichar: The construction of factor automata. In: Workshop'98, vol. 1, Czech Technical University, Prague 1997, pp. 189–190.

[4] M. Šimánek: Operations on factor automaton. In: Workshop '98, vol. 1, Czech Technical University, Prague 1997, pp. 207–208.

[5] M. Šimánek: The factor automaton. In: Proceedings of the Prague Stringology Club Workshop'98, Czech Technical University Prague, 1998, pp. 102–106.

[6] M. Šimánek: Operations on Factor Automata. Postgraduate Study Report DC-PSR-98-02, Czech Technical University Prague 1998, 38 pp.

Ing. Milan Šimánek, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, Karlovo nám. 13, 121 35 Praha 2. Czech Republic.

e-mail: simanek@fel.cvut.cz