Information and Computation 213 (2012) 3–12

Contents lists available at SciVerse ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

A compact representation of nondeterministic (suffix) automata for the bit-parallel approach

Domenico Cantone, Simone Faro, Emanuele Giaquinta *

Università di Catania, Dipartimento di Matematica e Informatica, Viale Andrea Doria 6, I-95125 Catania, Italy

ARTICLE INFO

Article history:
Available online 2 February 2012

ABSTRACT

We present a novel technique, suitable for bit-parallelism, for representing both the nondeterministic automaton and the nondeterministic suffix automaton of a given string in a more compact way. Our approach is based on a particular factorization of strings which on the average allows to pack in a machine word of w bits automata state configurations for strings of length greater than w. We adapted the Shift-And and BNDM algorithms using our encoding and compared them with the original algorithms. Experimental results show that the new variants are generally faster for long patterns.

1. Introduction

The string matching problem consists in finding all the occurrences of a pattern P of length m in a text T of length n, both defined over an alphabet Σ of size σ. The Knuth–Morris–Pratt (KMP) algorithm was the first linear-time solution (cf. [7]), whereas the Boyer–Moore (BM) algorithm provided the first sublinear solution on average [2]. Subsequently, the BDM algorithm reached the $\mathcal{O}(n \log_\sigma(m)/m)$ lower bound time complexity on the average (cf. [4]). Both the KMP and the BDM algorithms are based on finite automata; in particular, they simulate, respectively, a deterministic automaton for the language $\Sigma^* P$ and a deterministic suffix automaton for the language of the suffixes of P.

The bit-parallelism technique, introduced in [1], has been used to simulate efficiently the nondeterministic version of the KMP automaton. The resulting algorithm, named Shift-Or, runs in $\mathcal{O}(n\lceil m/w \rceil)$, where w is the number of bits in a computer word. A variant of the Shift-Or algorithm, called Shift-And, was described in [11]. Later, a very fast BDM-like algorithm (BNDM), based on the bit-parallel simulation of the nondeterministic suffix automaton, was presented in [8].

Bit-parallelism encoding requires one bit per pattern symbol, for a total of $\lceil m/w \rceil$ computer words. Thus, as long as a pattern fits in a computer word, bit-parallel algorithms are extremely fast, otherwise their performances degrade considerably as $\lceil m/w \rceil$ grows. Though there are a few techniques to maintain good performance in the case of long patterns, such limitation is intrinsic.

In this paper we present an alternative technique, still suitable for bit-parallelism, to encode both the nondeterministic automaton and the nondeterministic suffix automaton of a given string in a more compact way. Our encoding is based on factorizations of strings in which no character occurs more than once in any factor. This is the key towards separating the nondeterministic part from the deterministic one of the corresponding automata. It turns out that the nondeterministic part can be encoded with k bits, where k is the size of the factorization. Though in the worst case $k = m$, on the average k is much smaller than m, making it possible to encode large automata in a single or few computer words. As a consequence,

* Corresponding author.
E-mail addresses: cantone@dmi.unict.it (D. Cantone), faro@dmi.unict.it (S. Faro), giaquinta@dmi.unict.it (E. Giaquinta).

0890-5401/$ – see front matter
doi:10.1016/j.ic.2011.03.006

bit-parallel algorithms based on such approach tend to be faster in the case of sufficiently long patterns. We will illustrate this point by comparing experimentally different implementations of the Shift-And and the BNDM algorithms.

2. Basic notions and definitions

Given a finite alphabet Σ of size σ, we denote by Σ^m, with $m \geqslant 0$, the collection of strings of length m over Σ and put $\Sigma^* = \bigcup_{m\in\mathbb{N}} \Sigma^m$. We represent a string $P \in \Sigma^m$, also called an m-gram, as an array $P[0 \ldots m-1]$ of characters of Σ and write $|P| = m$ (in particular, for $m = 0$ we obtain the empty string ε). Thus, $P[i]$ is the $(i+1)$-st character of P, for $0 \leqslant i < m$, and $P[i \ldots j]$ is the substring of P contained between its $(i+1)$-st and $(j+1)$-st characters, for $0 \leqslant i \leqslant j < m$. Also, we define $first(P) = P[0]$ and $last(P) = P[|P|-1]$. For any two strings P and P', we say that P' is a suffix of P if $P' = P[i \ldots m-1]$, for some $0 \leqslant i < m$, and write $Suff(P)$ for the set of all suffixes of P. Similarly, P' is a prefix of P if $P' = P[0 \ldots i]$, for some $0 \leqslant i < m$. In addition, we write $P.P'$, or more simply PP', for the concatenation of P and P', and P^r for the reverse of the string P, i.e. $P^r = P[m-1]P[m-2]\ldots P[0]$.

Given a string $P \in \Sigma^m$, we indicate with $\mathcal{A}(P) = (Q, \Sigma, \delta, q_0, F)$ the nondeterministic automaton for the language $\Sigma^* P$ of all words in Σ^* ending with an occurrence of P, where:

- $Q = \{q_0, q_1, \ldots, q_m\}$ (q_0 is the initial state);
- the transition function $\delta : Q \times \Sigma \to \mathscr{P}(Q)$ is defined by

$$\delta(q_i, c) =_{\text{Def}} \begin{cases} \{q_0, q_1\} & \text{if } i = 0 \text{ and } c = P[0], \\ \{q_0\} & \text{if } i = 0 \text{ and } c \neq P[0], \\ \{q_{i+1}\} & \text{if } 1 \leqslant i < m \text{ and } c = P[i], \\ \emptyset & \text{otherwise;} \end{cases}$$

- $F = \{q_m\}$ (F is the set of final states).

Likewise, for a string $P \in \Sigma^m$, we denote by $\mathcal{S}(P) = (Q, \Sigma, \delta, I, F)$ the nondeterministic suffix automaton with ε-transitions for the language $Suff(P)$ of the suffixes of P, where:

- $Q = \{I, q_0, q_1, \ldots, q_m\}$ (I is the initial state);
- the transition function $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathscr{P}(Q)$ is defined by

$$\delta(q, c) =_{\text{Def}} \begin{cases} \{q_{i+1}\} & \text{if } q = q_i \text{ and } c = P[i] \ (0 \leqslant i < m), \\ Q & \text{if } q = I \text{ and } c = \varepsilon, \\ \emptyset & \text{otherwise;} \end{cases}$$

- $F = \{q_m\}$ (F is the set of final states).

The valid configurations $\delta^*(q_0, S)$ which are reachable by the automaton $\mathcal{A}(P)$ on input $S \in \Sigma^*$ are defined recursively as follows

$$\delta^*(q_0, S) =_{Def} \begin{cases} \{q_0\} & \text{if } S = \varepsilon, \\ \bigcup_{q' \in \delta^*(q_0, S')} \delta(q', c) & \text{if } S = S'c, \text{ for some } c \in \Sigma \text{ and } S' \in \Sigma^*. \end{cases}$$

Much the same definition of reachable configurations holds for the automaton $\mathcal{S}(P)$, but in this case one has to use $\delta(I, \varepsilon) = \{q_0, q_1, \ldots, q_m\}$ for the base case.

Finally, we recall the notation of some bitwise infix operators on computer words, namely the bitwise `and` "&", the bitwise `or` "|", the `left shift` "$\ll$" operator (which shifts to the left its first argument by a number of bits equal to its second argument), and the unary bitwise `not` operator "$\sim$".

3. The bit-parallelism technique

Bit-parallelism is a technique introduced by Baeza-Yates and Gonnet in [1] that takes advantage of the intrinsic parallelism of the bit operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w, where w is the number of bits in the computer word. Bit-parallelism is particularly suitable for the efficient simulation of nondeterministic (suffix) automata; the first algorithms based on it are the well-known Shift-And [11] (a variant of Shift-Or [1]) and BNDM [8]. The Shift-And algorithm simulates the nondeterministic automaton (NFA, for short) that recognizes the language $\Sigma^* P$, for a given string P of length m. Its bit-parallel representation uses an array B of σ bit-vectors, each of size m, where the i-th bit of $B[c]$ is set iff $\delta(q_i, c) = q_{i+1}$ or equivalently iff $P[i] = c$, for $c \in \Sigma$, $0 \leqslant i < m$. Automaton configurations $\delta^*(q_0, S)$ on input $S \in \Sigma^*$ are then encoded as a bit-vector D of m bits (the initial state does not need to be represented, as it is always active), where the i-th bit of D is set iff state q_{i+1} is active, i.e.