Information Processing Letters 114 (2014) 392–396

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

On a compact encoding of the swap automaton

Kimmo Fredriksson [a], Emanuele Giaquinta [b,1,*]

[a] School of Computing, University of Eastern Finland, P.O. Box 1627, FI-70211 Kuopio, Finland
[b] Department of Computer Science, University of Helsinki, Finland

ARTICLE INFO

Article history:
Received 1 July 2013
Received in revised form 13 January 2014
Accepted 13 January 2014
Available online 16 January 2014
Communicated by Ł. Kowalik

Keywords:
Combinatorial problems
String algorithms
Automata theory
Word-level parallelism

ABSTRACT

Given a string P of length m over an alphabet Σ of size σ, a swapped version of P is a string derived from P by a series of local swaps, i.e., swaps of adjacent symbols, such that each symbol can participate in at most one swap. We present a theoretical analysis of the nondeterministic finite automaton for the language $\bigcup_{P' \in \Pi_P} \Sigma^* P'$ (swap automaton, for short), where Π_P is the set of swapped versions of P. Our study is based on the bit-parallel simulation of the same automaton due to Fredriksson, and reveals an interesting combinatorial property that links the automaton to the one for the language $\Sigma^* P$. By exploiting this property and the method presented by Cantone et al. (2012), we obtain a bit-parallel encoding of the swap automaton which takes $O(\sigma^2 \lceil k/w \rceil)$ space and allows one to simulate the automaton on a string of length n in time $O(n \lceil k/w \rceil)$, where $\lceil m/\sigma \rceil \leqslant k \leqslant m$ is the size of a specific factorization of P defined by Cantone et al. (2012) and w is the word size in bits.

1. Introduction

The *Pattern Matching with Swaps* problem (Swap Matching problem, for short) is a well-studied variant of the classic Pattern Matching problem. It consists in finding all occurrences, up to character swaps, of a pattern P of length m in a text T of length n, with P and T sequences of characters over a common finite alphabet Σ of size σ. More precisely, the pattern is said to match the text at a given location j if adjacent pattern characters can be swapped, if necessary, so as to make it identical to the substring of the text ending (or, equivalently, starting) at location j. All swaps are constrained to be disjoint, i.e., each character can be involved at most in one swap.

The Swap Matching problem was introduced in 1995 as one of the open problems in nonstandard string matching [1]. The first result that improved over the naive $O(nm)$-time bound is due to Amir et al. [2], who presented an $O(nm^{\frac{1}{3}} \log m)$-time algorithm for binary alphabets and described how to reduce the case of a general alphabet to that of a binary one with an $O(\log \sigma)$-time overhead. The best theoretical result to date is due to Amir et al. [3]. Their algorithm runs in time $O(n \log m)$ for binary alphabets and can also solve the case of general alphabets in time $O(n \log m \log \sigma)$ by using again the alphabet reduction technique of Amir et al. [2]. Both solutions are based on reducing the problem to convolutions. Note that this problem can also be solved using more general algorithms for Approximate String Matching [4], albeit with worse bounds.

There also exist different practical solutions, based on word-level parallelism. To our knowledge, the first one is due to Fredriksson [5], who presented a generalization of the nondeterministic finite automaton (NFA) for the language $\Sigma^* P$ (prefix automaton) for the Swap Matching problem and a fast method to simulate it using bit-parallelism [6]. The resulting algorithm runs in $O(n \lceil m/w \rceil)$-time and uses $O(\sigma \lceil m/w \rceil)$ space, where w is the machine word size in bits. In the same paper Fredriksson also presented a variant of the BNDM algorithm [7], based on the generalization of the NFA for the language of the suffixes of P (suffix automaton), which achieves sublinear time on average and runs in $O(nm \lceil m/w \rceil)$-time

* Corresponding author.
E-mail address: emanuele.giaquinta@cs.helsinki.fi (E. Giaquinta).
[1] Supported by the Academy of Finland, Grant 118653 (ALGODAN).

0020-0190/$ – see front matter
http://dx.doi.org/10.1016/j.ipl.2014.01.004

in the worst-case. In 2008 Iliopoulos and Rahman presented a variant of Shift-Or for this problem, based on a Graph-Theoretic model [8]. Their algorithm runs in time $O(n\lceil m/w\rceil \log m)$ and uses $O(m\lceil m/w\rceil)$ space (the $\log m$ term can be removed at the price of $O(\sigma^2\lceil m/w\rceil)$ space). The improvement over the algorithm by Fredriksson is that the resulting bit-parallel simulation is simpler, in that it requires fewer bitwise operations. Later, Cantone and Faro presented an algorithm based on dynamic programming that runs in time $O(n\lceil m/w\rceil)$ and requires $O(\sigma\lceil m/w\rceil)$ space [9]. Subsequently Campanelli et al. presented a variant of the BNDM algorithm based on the same approach which runs in $O(nm\lceil m/w\rceil)$-time in the worst-case [10].

In [11] Cantone et al. presented a technique to encode the prefix automaton in $O(\sigma^2\lceil k/w\rceil)$ space and simulate it on a string of length n in $O(n\lceil k/w\rceil)$ time, where $\lceil m/\sigma\rceil \leqslant k \leqslant m$. In this paper we extend this result to the NFA described in [5]. First, we present a theoretical analysis of this NFA, from which the correctness of the bit-parallel simulation presented in the same paper follows. We then show that, by exploiting the properties of this NFA that we reveal in the following, we can solve the Swap Matching problem in time $O(n\lceil k/w\rceil)$ and space $O(\sigma^2\lceil k/w\rceil)$, where $\lceil m/\sigma\rceil \leqslant k \leqslant m$, using the method presented in [11]. Our result improves over the existing time bound in the word-RAM model by a factor of σ in the best case. Moreover, it can also be applied, with small changes, to the case of the generalized suffix automaton so as to obtain an improved BNDM-like algorithm for the Swap Matching problem.

2. Notions and basic definitions

Given a finite alphabet Σ of size σ, we denote by Σ^m, with $m \geqslant 0$, the set of strings of length m over Σ and put $\Sigma^* = \bigcup_{m\in\mathbb{N}} \Sigma^m$. We represent a string $P \in \Sigma^m$ 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, inclusive, for $0 \leqslant i \leqslant j < m$. For any two strings P and P', we write PP' to denote the concatenation of P and P'.

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

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

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:

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

Definition 1. A *swap permutation* for a string P of length m is a permutation $\pi : \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$ such that:

(a) if $\pi(i) = j$ then $\pi(j) = i$ (characters are swapped);
(b) for all i, $\pi(i) \in \{i-1, i, i+1\}$ (only adjacent characters are swapped);
(c) if $\pi(i) \neq i$ then $P[\pi(i)] \neq P[i]$ (identical characters are not swapped).

For a given string P and a swap permutation π for P, we write $\pi(P)$ to denote the *swapped version* of P, namely $\pi(P) = P[\pi(0)]P[\pi(1)]\ldots P[\pi(m-1)]$.

Definition 2 (*Pattern Matching with Swaps problem*). Given a text T of length n and a pattern P of length m, find all locations $j \in \{m-1, \ldots, n-1\}$ for which there exists a swap permutation π of P such that $\pi(P)$ matches T at location j, i.e. $P[\pi(i)] = T[j-m+i+1]$, for $i = 0\ldots m-1$.

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. 1-factorization encoding of the prefix automaton

A 1-factorization $\boldsymbol{u}$ of size $|\boldsymbol{u}| = k$ of a string P is a sequence $\langle u_1, u_2, \ldots, u_k\rangle$ of nonempty substrings of P such that:

(a) $P = u_1u_2\ldots u_k$;
(b) each factor u_j in $\boldsymbol{u}$ contains at most *one* occurrence of any of the characters in the alphabet Σ, for $j = 1, \ldots, k$.

The size k of a 1-factorization satisfies the condition $\lceil m/\sigma\rceil \leqslant k \leqslant m$. The following result was presented in [11]:

Theorem 1. *(Cf. [11].) Given a string P of length m and a 1-factorization of P of size k, we can encode the automaton $\mathcal{A}(P)$ in $O(\sigma^2\lceil k/w\rceil)$ space and simulate it in time $O(n\lceil k/w\rceil)$ on a string of length n.*

We briefly recall how the encoding of Theorem 1 works. A 1-factorization $\langle u_1, u_2, \ldots, u_k\rangle$ of P induces a partition $\{Q_1, \ldots, Q_k\}$ of the set $Q \setminus \{q_0\}$ of states of the automaton $\mathcal{A}(P)$, where

$$Q_i =_{\text{Def}} \{q_{r_i+1}, \ldots, q_{r_{i+1}}\}, \quad \text{for } i = 1, \ldots, k,$$