Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 98 (2006) 211–218
Information Processing Letters
www.elsevier.com/locate/ipl

Multiple serial episodes matching ☆

Patrick Cégielski [a], Irène Guessarian [b,*], Yuri Matiyasevich [c]

[a] *LACL, UMR-FRE 2673, Université Paris 12, Route forestière Hurtault, F-77300 Fontainebleau, France*
[b] *LIAFA, UMR 7089 and Université Paris 6, 2, Place Jussieu, 75254 Paris Cedex 5, France*
[c] *Steklov Institute of Mathematics, Fontanka 27, St. Petersburg, Russia*

Received 24 June 2005; received in revised form 3 February 2006; accepted 3 February 2006

Available online 20 March 2006

Communicated by L. Boasson

Abstract

Given $q+1$ strings (a text t of length n and q patterns $m_1, \ldots, m_q$) and a natural number w, the *multiple serial episode matching problem* consists in finding the number of size w windows of text t which contain patterns $m_1, \ldots, m_q$ as subsequences, i.e., for each m_i, if $m_i = p_1, \ldots, p_k$, the letters $p_1, \ldots, p_k$ occur in the window, in the same order as in m_i, but not necessarily consecutively (they may be interleaved with other letters). Our main contribution here is an algorithm solving this problem on-line in time $O(nq)$ with an MP-RAM model (which is a RAM model equipped with extra operations).

Keywords: Subsequence matching; Algorithms; Frequent patterns; Episode matching; Datamining

1. Introduction

The recent development of datamining induced the development of computing techniques, among them is episode searching and counting. An example of frequent serial episode search is as follows: let t be a text consisting of requests to a university webserver; assume we wish to count how many times, within at most 10 time units, the sequence $e_1e_2e_3e_4$ appears, where e_1 = 'Computer Science', e_2 = 'Master', e_3 = 'CS318 homepage', e_4 = 'Assignment'. It suffices to count the number of 10-windows of t containing the subsequence $p = e_1e_2e_3e_4$. If e_1, e_2, e_3, e_4 must appear in that same order in the window, the episode is said to be *serial*; if e_1, e_2, e_3, e_4 can appear in any order, the episode is said to be *parallel*; a partial order can also be imposed on the events forming an episode (see [14], which proposes several algorithms for episode searching). Searching serial episodes is more complex than searching parallel episodes. Of course, if one has to scan a log file, it is better to do it for several episodes $e_1e_2\ldots e_n$, $f_1f_2\ldots f_m$, $g_1g_2\ldots g_p$ simultaneously. We will hence investigate the search of several serial episodes in the same window: each serial episode is ordered, but no order is imposed among occurrences of the episodes in the window.

More precisely, the problem we address is the following: given a text t of length n, patterns $m_1, \ldots, m_q$ on the same alphabet A and an integer w, we wish to determine the number of size w windows of text con-

☆ Support by INTAS grant 04-77-7173 is gratefully acknowledged.
* Corresponding author.
E-mail addresses: cegielski@univ-paris12.fr (P. Cégielski), ig@liafa.jussieu.fr, irene.guessarian@liafa.jussieu.fr (I. Guessarian), yumat@pdmi.ras.ru (Y. Matiyasevich).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2006.02.008

taining all q patterns as serial episodes, i.e., the letters of each m_i appear in the window, in the same order as in m_i, but they need not be consecutive because other letters can be interleaved.

When searching for a single pattern m, this problem with arguments the window size w, the text t and pattern m is called *serial episode matching problem* in [14], *episode matching* in [8] and *subsequence matching* in [2]. This problem is a generalization of *pattern-matching*. A naive solution exists for *pattern-matching* Its time complexity on RAM is $O(nk)$, where k is the pattern size. Knuth et al. [10] gave a well-known algorithm solving the problem in linear time $O(n + k)$. Without the window size restriction, it is easy to find in linear time whether m is a subsequence of the text: if $m = p_1 \dots p_k$, a finite state automaton with $k + 1$ states $s_0, s_1, \dots, s_k$ will read the text; the initial state is s_0; after reading letter p_1 we go to state s_1, then after reading letter p_2 we go to state $s_2, \dots$; the text is accepted as soon as state s_k is reached. Episode matching within a w-window is harder; its importance is due to potential applications to datamining [13,14] and molecular biology [12,11,16]. For the problem with a single episode in w-windows, a standard algorithm is described in [8,14]. It is close to the algorithms of *pattern-matching* [1,2] and its time complexity is $O(nk)$. Another *on-line* algorithm is described in [8]: the idea is to slice the pattern in $k/\log k$ well-chosen pieces organized in a *trie*; its time complexity is $O(nk/\log k)$. We gave an *on-line* algorithm reading the text t, each text symbol being read only once and whose time complexity is $O(n)$ [6] on a new model, called MP-RAM. Because the abstract models are different and it is difficult to compare them, we implemented the standard algorithm and our algorithm and tested the results for benchmarks: when there is a single pattern, our algorithm on MP-RAM is in average 3 times faster than the standard algorithm, and can even be 10 times faster for large windows and large patterns [6].

The main contribution of this paper is an algorithm (Section 3) for solving problems of simultaneous search of multiple episodes. This algorithm uses the MP-RAM, that we introduced in [6], to model microprocessor basic operations, using only the fast operations on bits (*shifts*), and bit-wise addition; this gives an $O(nq)$-time on-line algorithm that works under assumption that the episode alphabet contains at most $\sqrt{n}/q$ symbols, see Theorem 1.

Our algorithm relies upon three ideas:

(1) preprocess patterns and window size to obtain a finite automaton solving the problem as in [15,10] (the solutions preprocessing the text [20,12,18,21] are prohibitive here because of their space complexity),
(2) code the states of this automaton to compute its transitions very quickly on MP-RAMs, without precomputing, nor storing the automaton: using the automaton itself is also prohibitive, not the least because of the number of states; we emulate the behavior of the automaton without computing the automaton, and
(3) a new implementation of *tries*.

We study: (a) the case when the patterns have no common part and (b) the case when they have similar parts. In each case, an appropriate preprocessing of the set of patterns enables us to build an automaton solving the problem and we show that the behavior of this automaton can be emulated on-line on MP-RAMs. Moreover, the time complexity of the preprocessing is insignificant because it is smaller than the text size by several orders of magnitude: typically, window and patterns will consist of a few dozen characters while the text will consist of several million characters.

The paper is organized as follows: in Section 2, we define the problem, in Section 3 we describe the algorithms which search for multiple serial episodes in parallel.

2. The problem

2.1. The (multiple) episode problem

An *alphabet* is a finite non-empty set A. A *length n word t* on A is a sequence $t_1 t_2 \cdots t_n$ of letters from A. The only length zero word is the *empty word*, denoted by ε. Let $t = t_1 t_2 \cdots t_n$ be a word which will be called the *text* in the paper. The word $p = p_1 p_2 \cdots p_k$ is a *factor* of t iff, there exists an integer j such that $t_{j+i} = p_i$ for $1 \leqslant i \leqslant k$. A size w *window* of on t, in short w-window, is a size w factor $t_{i+1} t_{i+2} \cdots t_{i+w}$ of t; there are $n - w + 1$ such windows in t. The word p is an *episode* (or *subsequence*) of t iff there exist integers $1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n$ such that $t_{i_j} = p_j$ for $1 \leqslant j \leqslant k$. If moreover, $i_k - i_1 < w$, p is an *episode* of t *in a w-window*.

Example 1. If $t =$ "dans ville il y a vie" (a French advertisement) then "vie" is a factor and hence a subsequence of t. "vile" is neither a factor, nor a subsequence of t in a 4-window, but it is a subsequence of t in a 5-window. See Fig. 1.