Theoretical Computer Science 638 (2016) 27–32

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Permuted scaled matching ☆

Ayelet Butman [a,b,c], Noa Lewenstein [a,b,c,1], J. Ian Munro [a,b,c,2]

[a] Department of Computer Science, Holon Institute of Technology, Israel
[b] Department of Computer Science, Netanya College, Israel
[c] Cheriton School of Computer Science, University of Waterloo, Canada

ARTICLE INFO

Article history:
Received 9 April 2015
Received in revised form 2 January 2016
Accepted 24 February 2016
Available online 24 May 2016

Keywords:
Pattern matching
Scaled matching
Jumbled matching

ABSTRACT

Scaled matching and permutation matching are two well known paradigms in the domain of pattern matching. Scaled matching refers to finding an occurrence of a pattern which is enlarged proportionally by some scale k within a larger text. Permutation matching is the problem of finding all substrings within a text where the character statistics of the substring and the pattern are the same. Permutation matching is easy, while scaled matching requires innovative solutions. One interesting setting of applications is the merge of the two. The problem of scaled permuted matching (i.e. first permuting and then scaling) has been addressed and solved optimally. However, it was left as an open problem whether there are efficient algorithms for permuted scaled matching. In this paper we solve the problem efficiently in a deterministic setting and optimally in a randomized setting.

1. Introduction

Classical pattern matching for exact matching has been very successfully approached lending a multitude of methods that work in linear time and limited space, e.g. [9,17,18,20–22,28]. However, the definition of a match has been expanded in many directions over the years, taking into account the different types of problems that arise from different domains. Two settings will be of central interest in this paper: *scaled pattern matching* and *permutation matching*, also known as *jumbled pattern matching*.

A k-scaling of a text is when each character of the text is replaced by k copies of the character. For example, if $S = \texttt{abbca}$ then the 3-scaling of S is $\texttt{aaabbbbbbcccaaa}$. *Scaled pattern matching*, where one is given a pattern P and a text T and one seeks all substrings of T which match a k-scaling of P for some $k \geq 1$, was first considered in [8]. Efficient solutions were proposed in that paper also for the two-dimensional version. More efficient algorithms for this problem were proposed in [7]. In [5] it was shown how to remove the alphabet dependence in the solution of [8]. The scaling problem, of course, in real-life, works also with real scales. However, in the setting of matching a good definition is quite elusive and this has been an avenue of research that has been explored quite extensively. For the case of one dimensional real scaled matching there are two natural definitions, appearing in [2,4]. It is interesting to note that the different definitions lead to very different solutions. The two-dimensional case for scaling with real numbers was examined in [3], where interesting solutions were presented.

☆ A preliminary version of this paper appeared in CPM 2014 [13].

E-mail address: noa.lewenstein@gmail.com (N. Lewenstein).

[1] The second author did this research while on sabbatical in the U. of Waterloo.

[2] This research was supported by NSERC of Canada and the Canada Research Chairs Programme.

http://dx.doi.org/10.1016/j.tcs.2016.02.036
0304-3975/

The other domain is that of *permutation matching*, also known as *jumbled pattern matching* [10,12] and in the case of indexing as *histogram indexing*. The problem of permutation matching is when one is given a pattern and a text and needs to find all substrings of a text for which the frequency count of the pattern character set is equal to the frequency count of the substring character set.

For example, if $P = \texttt{abbca}$ and the text is `acababbbcaacca`, there is a permutation match at location 2, because the substring `cabab` has 2 `a`'s, 2 `b`'s, and 1 `c`, exactly as it does in the pattern. The same is true for location 7, where the substring `bbcaa` begins.

Permutations of data are used in various settings. For example, permutations are used when one wants to maintain security of files; see e.g. [27,25]. In pattern matching this was initially used as a filter in approximate pattern matching algorithms [19]. The problem is actually quite simple to solve in linear time. However, there are several points of interest. In [15] it was shown how to obtain non-trivial efficient average-case behavior. In [11] solutions for finding approximate permutation matching were considered.

The indexing variant turns out to be very difficult; see [6]. However, for the binary case there are new results; see [14]. This is also true for small constant alphabet size [14,23]. The interest in indexing was initiated by the work on text fingerprinting via Parikh mappings [1]. The hardness of the indexing variant is quite surprising because exact matching is harder than permutation matching in the pattern matching setting but seems to be much easier in the indexing setting. An initial result for indexing for permutation matching was presented in [26]. There the authors considered the binary alphabet case and only announced whether there was a match or not. They succeeded in shaving off a double log factor from the quadratic space necessary. In [16] a more sophisticated near-linear *randomized* time construction algorithm was given.

A natural combination of scaled matching and permutation matching was originally considered in [12]. In that paper the combination was defined as first permuting the characters of the string and then allowing a k-scaling. However, first allowing a k-scaling and then permuting the characters is more natural; this makes the problem harder. This happens because the k-scaling gives a handle on the matching because of the repetition of the characters. So, if it is done as the second operation, the shape of the k-scaling allows exploitation of this data. But if the k-scaling happens first, then the permutation jumbles the k-scaling structure. This problem was left as an open question in [12] and we consider it in this paper. It is defined in the preliminaries and algorithms are shown to efficiently solve the problem in a deterministic setting and optimally in a randomized setting. Specifically, the final algorithms for an n-length text and $|\Sigma|$ sized alphabet will run in times $O(n \log |\Sigma|)$ and $O(n)$ time respectively.

The former algorithm assumes the comparison model, whereas the latter assumes the more general word-RAM model. We assume that $\Sigma \subset [1, cn]$, for some constant c.

2. Preliminaries

Let S be a string. Denote with $\#_\sigma(S)$ the number of occurrences of σ in S. Denote with $S[i, j]$ the $j - i + 1$ length substring of S starting at i, i.e. $s_i s_{i+1} \cdots s_j$. The i length prefix of S, denoted $S[0, i-1]$, is $S[0, i-1]$.

Let S be a string and let k be a natural number. A *k-scaling of S* is a string that is obtained by replacing every character σ of S with σ^k, where σ^k denotes σ repeated k times. For example, if $S = \texttt{abca}$ then the 4-scaling of S is `aaaabbbbccccaaaa`.

Let S be a string and k a natural number. We say that a string $\pi(S, k)$ is the *permuted scale of order k of S* if it can be generated by a permutation of the characters of a k-scaling of S.

Say we have a text T. A pattern P is said to have a *permuted k-scaled occurrence at location i of T* if the substring $T[i, i + |P|k - 1]$ is a permuted scale of order k of P.

Permuted Scaled Matching

INPUT: A text $T = t_0 t_1 \cdots t_{n-1}$ and a pattern $P = p_0 p_1 \cdots p_{m-1}$ over alphabet $\Sigma = \{\sigma_0, \sigma_1, \cdots, \sigma_{|\Sigma|-1}\}$
OUTPUT: All permuted k-scaled occurrences of P, for all possible k $(1 \le k \le \lfloor \frac{n}{m} \rfloor)$.

A straightforward algorithm that solves the Permuted Scaled Matching problem works as follows:

1. Construct a table R of size $(n+1) \times |\Sigma|$ such that $R(i, j) = \#_{\sigma_j}(T[0, i])$ for $i \ge 0$ and $R(-1, j) = 0$.
2. For every $0 \le i < j \le n-1$ such that $j - i + 1 = km$ for some natural number $k \ge 1$ do:
   (a) Let $r(l) = \frac{R(j,l) - R(i-1,l)}{\#_{\sigma_l}(P)}$.
   (b) If $r(l) = k$ for each l, $0 \le l \le |\Sigma| - 1$, then announce that P has a scaled occurrence at position i.

The running time for this algorithm is $O(\frac{n^2|\Sigma|}{m})$.