Journal of Computer and System Sciences 78 (2012) 273–278

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

A new efficient indexing algorithm for one-dimensional real scaled patterns ☆

Yung-Hsing Peng, Chang-Biau Yang *, Chiou-Ting Tseng, Chiou-Yi Hor

Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, 80424, Taiwan

ARTICLE INFO

Article history:
Received 15 July 2007
Received in revised form 7 December 2010
Accepted 5 May 2011
Available online 12 May 2011

Keywords:
Algorithm
String matching
Preprocessing
Real-scale

ABSTRACT

Given a pattern string P and a text string T, the one-dimensional real-scale pattern matching problem is to ask for all matched positions in T at which P occurs for some real scales $\geqslant 1$. The real-scale indexing problem, which is derived from the real-scale matching problem, aims to preprocess T, so that all positions of scaled P in T can be answered efficiently. In this paper, we propose an improved algorithm for the real-scale indexing problem. For constant-sized alphabets, our preprocessing takes $O(|T|^2)$ time and space, achieving the answering time $O(|P|+w)$, where U_r denotes the number of matched positions and $w \leqslant U_r$. For the case of large-sized alphabets, our preprocessing can still be implemented with $O(|T|^2)$ time and space, while the answering time is slightly increased to $O(|P|+w+\log|T|)$. Compared to Wang's preprocessing algorithm with $O(|T|^3)$ time, our new indexing algorithm is a significant improvement.

1. Introduction

Pattern matching is a classical problem which asks for all positions of a pattern P in a text T. According to various perspectives on T, algorithms for string matching can be classified into two main types. In the first perspective, both T and P are input strings, which means any algorithm requires $\Omega(|T|+|P|)$ time, where $|T|$ and $|P|$ denote the lengths of T and P, respectively. Many linear time algorithms have been proposed to solve this problem [9,13]. In the second perspective, which is called the *string indexing problem*, T is treated as a database while P is treated as the input target string. That is, every indexing algorithm has two phases, which are the preprocessing phase with T and the searching phase with P. In the preprocessing phase, any algorithm should take $\Omega(|T|)$ time. Besides, the searching phase needs $\Omega(|P|+U)$ time to report all positions, where U denotes the number of reported positions. For the second perspective, suffix trees [21] and suffix arrays [1,15] are well-known approaches with optimal preprocessing and searching time, if the alphabet is fixed.

In addition to the original definition, other extended versions of string matching, such as string matching with don't care characters [10], mismatches [7], or scaling [2,3] are considered not only interesting, but also more realistic. Among them, related problems that involve matching [2–6] or indexing [18,20] scaled patterns have drawn much attention. In this paper, we focus on the one-dimensional *real-scale pattern* [2,20], which means the pattern P can be scaled with real numbers. For example, suppose $T = c^4a^5b^7a^4c^3b^4$ and $P = a^2b^3a^2c^1$, one can easily verify that with the scale $\frac{7}{3}$, the scaled pattern $a^{\lfloor\frac{7}{3}\times 2\rfloor}b^{\lfloor\frac{7}{3}\times 3\rfloor}a^{\lfloor\frac{7}{3}\times 2\rfloor}c^{\lfloor\frac{7}{3}\times 1\rfloor} = a^4b^7a^4c^2$ is a substring of T.

The real-scale matching (r-matching) problem is first defined by Amir et al. [2], who also give an $O(|T|+|P|)$-time solution. In their conclusion [2], they ask for a proper preprocessing on T such that all real-scale matched (r-matched) positions of P in T can be determined efficiently. Afterwards, Wang et al. [20] answer this open question with the first

☆ This research work was partially supported by the National Science Council of Taiwan under contract NSC-95-2221-E-110-102.
* Corresponding author. Fax: +886 7 5254301.
E-mail address: cbyang@cse.nsysu.edu.tw (C.-B. Yang).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2011.05.001

known indexing algorithm. For fixed alphabets, Wang's preprocessing takes $O(|T|^3)$ time and $O(|T|^3)$ space [20], achieving the answering time $O(|P| + U_r)$, where U_r denotes the number of r-matched positions. Besides, they give an $O(|T|^2)$-time and $O(|T|^2)$-space preprocessing on T, so that the decision problem whether P can be r-matched in T can be determined in $O(|P|)$ time [20]. For large alphabets, Wang's preprocessing for the r-matching problem can be implemented with the same cost by using a suffix array [14,15], which achieves the answering time $O(|P| + U_r + \log|T|)$ [20].

In this paper, we propose an improved indexing algorithm that takes $O(|T|^2)$ time and $O(|T|^2)$ space in its preprocessing phase. With our indexing algorithm, for fixed alphabets, one can determine whether a pattern P is r-matched in T in $O(|P|)$ time, and find all r-matched positions in $O(|P| + w)$ time, where $w \leqslant U_r$ and w denotes the number of *dominant positions*, which will be further explained in Section 3. With a little modification, we also show how to deal with large alphabets in Section 3.

The rest of this paper is organized as follows. Section 2 provides the review for required techniques. Next, we propose our main algorithm in Section 3, and apply it to other scaling functions [20] in Section 4. Finally, we give our conclusions along with some future work in Section 5.

2. Preliminaries

In this section, we begin with notations used in this paper. After that, we briefly introduce all required techniques.

2.1. Notations

Let T be a string over the alphabet Σ, and $|\Sigma|$ denote the number of distinct symbols in Σ. Also, let $T[i]$ denote the ith character in T, and $T[i, j]$ denote the substring ranging from $T[i]$ to $T[j]$, for $1 \leqslant i \leqslant j \leqslant |T|$. Let T' be the *run-length encoding* (RLE) of T, so that $T' = t_1^{r_1} t_2^{r_2} \cdots t_m^{r_m}$ with $|T'| = m$, where $t_i \in \Sigma$, for $1 \leqslant i \leqslant m$, $t_j \neq t_{j+1}$, $1 \leqslant j \leqslant m-1$ and r_i denotes the run length of t_i. Therefore, one can easily map each character $T'[i]$ in T' to the run $T[\sum_{j=1}^{i-1} r_j + 1, \sum_{j=1}^{i} r_j]$ in T. Let $P' = p_1^{s_1} p_2^{s_2} \cdots p_u^{s_u}$ be the RLE of the pattern string P, where $|P'| = u$ and $|P| = \sum_{j=1}^{u} s_j$. The α-scaling of P, denoted by $\delta_\alpha(P)$, for any real number $\alpha \geqslant 1$, represents the string $p_1^{\lfloor \alpha s_1 \rfloor} p_2^{\lfloor \alpha s_2 \rfloor} \cdots p_u^{\lfloor \alpha s_u \rfloor}$.

Given a text T and a pattern P, the *real-scale matching problem* [2] is to determine every matched position i in T where $\delta_\alpha(P)$ occurs, for some real number $\alpha \geqslant 1$. Considering the previous example for $T = c^4a^5b^7a^4c^3b^4$ and $P = a^2b^3a^2c^1$, one can see that $\delta_{\frac{7}{3}}(P) = a^4b^7a^4c^2$ matches the substring $T[6, 22]$ at position 6 in T.

2.2. Suffix trees and suffix arrays

In the field of string matching, *suffix trees* and *suffix arrays* have been widely studied. Given a text T over the alphabet Σ, the suffix tree T_S of T is an $O(|T|)$-space compacted trie of all suffixes in T [16,19,21]. Given a pattern P along with the suffix tree T_S, one can determine all positions of P in T with $O(|P|\log|\Sigma| + U)$ time, where U denotes the number of reported positions. Different from a suffix tree, a suffix array [15] lexically stores each index (suffix) of T. Since one index is sufficient to represent the suffix starting at $T[i]$, the suffix array T_A of T occupies only $O(|T|)$ space, which is independent of Σ. By keeping the information of the longest common prefix of suffixes, to search a given pattern P in T, one can perform a binary search on T_A, which requires $O(|P| + U + \log|T|)$ time in the searching phase [15].

The details for suffix trees and suffix arrays are omitted here, since they are beyond the scope of this paper. Readers can easily find these material in other papers [1,11,12,14] or some text books for string matching.

2.3. The range minimum query

Given an array A of n numbers, the *range minimum* (*maximum*) *query* (RMQ) asks for the minimum (maximum) element in the subarray $A[i_1, i_2]$, $1 \leqslant i_1 \leqslant i_2 \leqslant n$, where the interval $[i_1, i_2]$ is specified by the query. Let $MIN_A(i_1, i_2)$ and $MAX_A(i_1, i_2)$ be the index of the minimum and maximum element in the subarray $A[i_1, i_2]$, respectively. Bender and Farach-Colton [8] propose the following result.

Theorem 1. *(See [8].) Given an array A of n numbers, one can preprocess A in $O(n)$ time such that for any given interval $[i_1, i_2]$, one can determine both $MIN_A(i_1, i_2)$ and $MAX_A(i_1, i_2)$ in $O(1)$ time.*

With RMQ, Muthukrishnan [17] further obtains the following theorem, which can be used to solve the document listing problem [17].

Theorem 2. *(See [17].) Given an array A of n integers whose absolute values are bounded by $O(n)$, one can preprocess A in $O(n)$ time such that for any given interval $[i_1, i_2]$, one can determine all distinct integers in the subarray $A[i_1, i_2]$ in $O(U_x)$ time, where U_x is the number of reported indices containing distinct integers.*

In Section 3, we will apply Theorems 1 and 2 to achieve our improvement.

2.4. Wang's preprocessing for the r-matching decision problem

In this subsection, we give a brief introduction to Wang's [20] $O(|T|^2)$-time preprocessing for the r-matching decision problem. Though this preprocessing is not feasible for answering general r-matching queries, it is an inspiration of our algorithm. The main idea of Wang's $O(|T|^2)$-time preprocessing is to generate the finite set $\Gamma(T)$ of real scales for the text T, then use these scales to generate all valid patterns, which are then stored in a suffix tree. One can refer to Wang's result [20] for the detailed proofs of the following lemmas.

Lemma 1. *(See [20].) Given $T = t_1^{r_1} t_2^{r_2} \cdots t_m^{r_m}$, $\Gamma(T) = \bigcup_{i=1}^{m} \bigcup_{j=1}^{r_i} \frac{r_i}{j}$ is a set of scales such that any given pattern P can be real-scale matched in T if and only if there exists any $\alpha \in \Gamma(T)$ such that $\delta_\alpha(P)$ occurs in T.*

Based on Lemma 1, Wang et al. constructed $|\Gamma(T)|$ strings $T'_{\alpha_1}, T'_{\alpha_2}, \ldots, T'_{\alpha_{|\Gamma(T)|}}$, where each T'_{α_k} corresponds to scale α_k in $\Gamma(T)$, for $1 \leqslant k \leqslant |\Gamma(T)|$. Given $T' = t_1^{r_1} t_2^{r_2} \cdots t_m^{r_m}$, each T'_{α_k} can be generated from T' by doing the replacement as follows. Let $f(r_j, \alpha_k) = \beta$ be the largest integer such that $\lfloor \beta \alpha_k \rfloor \leqslant r_j$. For each $t_j^{r_j}$ in T', $1 \leqslant j \leqslant m$, it is replaced by $t_j^{f(r_j,\alpha_k)}$ if $\lfloor f(r_j, \alpha_k)\alpha_k \rfloor = r_j$. Otherwise, $t_j^{r_j}$ is replaced by $t_j^{f(r_j,\alpha_k)} \$ t_j^{f(r_j,\alpha_k)}$, where \$ denotes the terminate symbol which does not appear in T. Because $r_j \in Z^+$ and $\alpha_k \geqslant 1$, $f(r_j, \alpha_k)$ can be computed in constant time by checking two integers $\lceil \frac{r_j}{\alpha_k} \rceil$ and $\lfloor \frac{r_j}{\alpha_k} \rfloor$. Therefore, each T'_{α_k} can be constructed in $O(m)$ time.

Lemma 2. *(See [20].) For any given pattern P, $\delta_{\alpha_k}(P)$ occurs in T if and only if P occurs in T_{α_k}.*

By Lemmas 1 and 2, Wang et al. store the string $T_R = T_{\alpha_1} \$ T_{\alpha_2} \$ \cdots \$ T_{\alpha_{|\Gamma(T)|}}$ with a suffix tree, where each T_{α_k} denotes the uncompressed format of T'_{α_k}. Since $|\Gamma(T)|$ is bounded by $|T|$ and the construction of each T_{α_k} takes $O(|T|)$ time in the worst case, Wang's result can be described as follows.

Theorem 3. *(See [20].) Given a pattern P and a text T, one can preprocess T with $O(|T|^2)$ time and space, so that whether $\delta_\alpha(P)$ occurs in T for some real scale $\alpha \geqslant 1$ can be determined in $O(|P|)$ time for fixed alphabets.*

However, one should note that this preprocessing cannot be applied to report all matched positions in T. This is resulted from two main flaws. The first flaw happens in the replacement rule, which only enables one to locate matched positions in T', rather than T. The second flaw is the redundancy of reported positions. One can see that there are $O(|T|^2)$ leaf nodes in the suffix tree of T_R, but there are only m mapping positions, where $m \leqslant |T|$. To report all r-matched positions in time $O(|P| + U_r)$, Wang et al. [20] devise another $O(|T|^3)$-time preprocessing.

In the next section, we will propose an efficient $O(|T|^2)$-time preprocessing, which can be used to report all matched positions in $O(|P| + w)$ time, where $w \leqslant U_r$.

3. An improved indexing algorithm

In this section, we first explain some useful properties for indexing real scaled patterns. Then, we show how to derive a better indexing algorithm by using these properties.

3.1. Properties for indexing

We begin with the concept of *dominant positions*, which enables us to report all matched positions efficiently. The definition of a dominant position is given as follows.

Definition 1. In the r-matching problem, a position i in T is called a dominant position if i is a matched position and $i+1$ is not a matched position.

For a given pattern P, let $DP = \{dp_1, dp_2, \ldots, dp_w\}$ denote the set of dominant positions in T. Since each position in T has its own corresponding position in T', we have $DP' = \{dp'_1, dp'_2, \ldots, dp'_w\}$, which denotes the corresponding set of dominant positions in T'. It is clear that the mapping between DP and DP' is one-to-one.

Observation 1. Given two dominant positions dp_i and dp_j in T, whose corresponding positions in T' are dp'_i and dp'_j, respectively. We have $dp_i = dp_j$ if and only if $dp'_i = dp'_j$.

Suppose $P = p_1^{s_1} p_2^{s_2} \cdots p_u^{s_u}$ is r-matched at position dp'_i in T', then the valid scale of P can be written as $\bigcap_{j=2}^{u-1} [\frac{r_{dp'_i+j-1}}{s_j}, \frac{r_{dp'_i+j-1}+1}{s_j}) \cap [1, \frac{r_{dp'_i}+1}{s_1}) \cap [1, \frac{r_{dp'_i+u-1}+1}{s_u}) = [\alpha_{low}, \alpha_{up})$, where α_{low} and α_{up} denote the lower and upper bounds

of the valid scale, respectively. That is, the matched positions in T derived from $T'[dp'_i]$ are successive indices ranging from $(\sum_{j=1}^{dp'_i} r_j - \lceil \alpha_{up} s_1 \rceil + 2)$ to $(\sum_{j=1}^{dp'_i} r_j - \lfloor \alpha_{low} s_1 \rfloor + 1)$. With an $O(|T|)$-time preprocessing on T, $\sum_{j=1}^{dp'_i} r_j$ can be determined in $O(1)$ time for any given dp'_i. Therefore, one can see the correctness of the following lemma, which is a key point in our indexing algorithm.

Lemma 3. *Given the set of dominant positions $DP' = \{dp'_1, dp'_2, \ldots, dp'_w\}$ and the set of ranges $SOR = \{[low_1, up_1), [low_2, up_2), \ldots, [low_w, up_w)\}$, where $[low_i, up_i)$ denotes the range for the valid scale of dp'_i, all matched positions in T can be determined in $O(w)$ time.*

3.2. New efficient indexing with RMQ

Let W be the set of matched (reported) positions of $\delta_\alpha(P)$ in T. Briefly, the main steps of our indexing algorithm can be described as follows.

Step 1: Determine if there exists any $\delta_\alpha(P)$ in T. If there exists none, set $W = \phi$ and go to Step 4.
Step 2: Obtain DP' and SOR.
Step 3: Construct W with DP' and SOR.
Step 4: Report W.

With Wang's $O(|T|^2)$-time preprocessing, it is clear that Step 1 can be done in $O(|P|)$ time. In addition, based on Lemma 3, W can be constructed in Step 3 and reported in Step 4 with $O(w)$ time once DP' and SOR are given. In the following, we explain how to obtain DP' and SOR in Step 2 with $O(w)$ time by using RMQ queries. One will see that for these RMQ queries, the additional preprocessing takes only $O(|T|^2)$ time, by which we complete our indexing algorithm.

First, we describe how to obtain DP' in $O(w)$ time. Let T_{SR} be the suffix tree of the concatenated string $T_R = T_{\alpha_1}\$T_{\alpha_2}\$\cdots\$T_{\alpha_{|\Gamma(T)|}}$. Let L be an array of size $|T_R|$ that lexically stores the indices of T_R (lexically stores the leaf nodes of T_{SR}). For each index $L[i]$ in L, we store its corresponding position in T' with another array LAB of size $|T_R|$. Recall that T_R is generated from T', which means for each $T_R[L[i]] \neq \$$, $T_R[L[i]]$ maps to a unique $T'[j]$ in T', for $1 \leqslant j \leqslant m$. Therefore, LAB can be constructed in $O(|T|^2)$ time, since $|T_R| = O(|T|^2)$. For $T_R[L[i]] = \$$, we set $LAB[i] = 0$ and let $T'[0]$ be an empty character. If $\delta_\alpha(P)$ exists in T, then the searching of P in T_{SR} (Step 1) would report some successive leaf nodes $L[i, j]$, whose corresponding positions in T' are stored in $LAB[i, j]$.

Observation 2. For each $k \in DP'$, in $LAB[i, j]$ there exist some $LAB[k'] = k$.

However, recall that $LAB[i, j]$ remains different from DP', because the size of $LAB[i, j]$ is $O(|T|^2)$ and there may be some duplications in $LAB[i, j]$. By Theorem 2, all redundant mappings in $LAB[i, j]$ can be removed by RMQ queries, thus we obtain the following lemma.

Lemma 4. *With an $O(|T|^2)$-time preprocessing on LAB, for any given pattern P that $\delta_\alpha(P)$ exists in T, one can obtain DP' in $O(w)$ time.*

Next, we explain how to obtain SOR in $O(w)$ time. Let $T'_R = T'_{\alpha_1}\$T'_{\alpha_2}\$\cdots\$T'_{\alpha_{|\Gamma(T)|}} = q_1^{z_1} q_2^{z_2} \cdots q_{|T'_R|}^{z_{|T'_R|}}$ be the RLE format of $T_R = T_{\alpha_1}\$T_{\alpha_2}\$\cdots\$T_{\alpha_{|\Gamma(T)|}}$. By Wang's preprocessing, one can map each $T'_R[i']$ to $T'[i'']$ in $O(1)$ time. We associate T'_R with two arrays LOW and UP, both of size $|T'_R|$. We assign the elements in LOW and UP by using $T'_R = q_1^{z_1} q_2^{z_2} \cdots q_{|T'_R|}^{z_{|T'_R|}}$ and $T' = t_1^{r_1} t_2^{r_2} \cdots t_m^{r_m}$ as follows. For $q_{i'} \neq \$$, we assign $LOW[i'] = \frac{r_{i''}}{z_{i'}}$ and $UP[i'] = \frac{r_{i''}+1}{z_{i'}}$, for $1 \leqslant i' \leqslant |T'_R|$. Otherwise, both $LOW[i']$ and $UP[i']$ are set to ∞. We preprocess LOW and UP for the range maximum query and the range minimum query, respectively. It is clear that the construction and preprocessing for LOW and UP can be accomplished in $O(|T|^2)$ time. In the following, we show that SOR can be determined in $O(w)$ time by using RMQ queries on LOW and UP.

Lemma 5. *With the searching result of P in T_{SR}, one can determine SOR in $O(w)$ time.*

Proof. According to Wang's preprocessing, one can map any $T_R[i]$ to $T'_R[i']$ and $T'[i'']$ in $O(1)$ time. For $P = p_1^{s_1} p_2^{s_2} \cdots p_u^{s_u}$ and $i'' \in DP'$, $T'[i'', i'' + u - 1] = t_{i''}^{r_{i''}} t_{i''+1}^{r_{i''+1}} \cdots t_{i''+u-1}^{r_{i''+u-1}}$ is a substring in T' that (1) $t_{i''+j-1} = p_j$, for $1 \leqslant j \leqslant u$, and (2) $r_{i''} \geqslant \lfloor s_1 \alpha \rfloor$, $r_{i''+u-1} \geqslant \lfloor s_u \alpha \rfloor$ and $r_{i''+j-1} = \lfloor s_j \alpha \rfloor$, for $2 \leqslant j \leqslant u-1$ and some real scale $\alpha \in \Gamma(T)$. Therefore, the range of valid scale at position i'' in T' is $\bigcap_{j=2}^{u-1} [\frac{r_{i''+j-1}}{s_j}, \frac{r_{i''+j-1}+1}{s_j}) \cap [1, \frac{r_{i''}+1}{s_1}) \cap [1, \frac{r_{i''+u-1}+1}{s_u}) = [max_{j=2}^{u-1} \frac{r_{i''+j-1}}{s_j}, min_{j=2}^{u-1} \frac{r_{i''+j-1}+1}{s_j}) \cap [1, \frac{r_{i''}+1}{s_1}) \cap [1, \frac{r_{i''+u-1}+1}{s_u})$.

Table 1
Algorithms for finding one-dimensional real scaled patterns.

Algorithm	Complexity	Fixed alphabets	Large alphabets
Amir et al. [2]	Time	$O(\|T\|+\|P\|)$	$O(\|T\|+\|P\|)$
	Space	$O(\|T\|+\|P\|)$	$O(\|T\|+\|P\|)$
	Decision	$O(\|T\|+\|P\|)$	$O(\|T\|+\|P\|)$
	Position	$O(\|T\|+\|P\|)$	$O(\|T\|+\|P\|)$
Wang's method 1 [20]	Time	$O(\|T\|^3)$	$O(\|T\|^3)$
	Space	$O(\|T\|^3)$	$O(\|T\|^3)$
	Decision	$O(\|P\|)$	$O(\|P\|+\log\|T\|)$
	Position	$O(\|P\|+U_r)$	$O(\|P\|+U_r+\log\|T\|)$
Wang's method 2 [20]	Time	$O(\|T\|^2)$	$O(\|T\|^2)$
	Space	$O(\|T\|^2)$	$O(\|T\|^2)$
	Decision	$O(\|P\|)$	$O(\|P\|+\log\|T\|)$
	Position	Disabled	Disabled
This paper	Time	$O(\|T\|^2)$	$O(\|T\|^2)$
	Space	$O(\|T\|^2)$	$O(\|T\|^2)$
	Decision	$O(\|P\|)$	$O(\|P\|+\log\|T\|)$
	Position	$O(\|P\|+w)$	$O(\|P\|+w+\log\|T\|)$

For each $T_R[i]$ that maps to $T'[i'']$ with $i'' \in DP'$ (each $T_R[L[k]]$ with $LAB[k] \in DP'$), we have $T_R[i, i+|P|-1] = P[1,|P|]$, which indicates $T'_R[i'+1, i'+u-2] = P'[2, u-1] = p_2^{s_2} p_3^{s_3} \cdots p_{u-1}^{s_{u-1}}$. Therefore, the range $[max_{j=2}^{u-1} \frac{r_{i''+j-1}}{s_j}, min_{j=2}^{u-1} \frac{r_{i''+j-1}+1}{s_j})$ can be obtained in $O(1)$ time by using a range maximum query and a range minimum query on $LOW[i'+1, i'+u-2]$ and $UP[i'+1, i'+u-2]$, respectively. For any given i'', both $[1, \frac{r_{i''}+1}{s_1})$ and $[1, \frac{r_{i''+u-1}+1}{s_u})$ can be determined in $O(1)$ time. Therefore, for each $i'' \in DP'$, its valid scale $[max_{j=2}^{u-1} \frac{r_{i''+j-1}}{s_j}, min_{j=2}^{u-1} \frac{r_{i''+j-1}+1}{s_j}) \cap [1, \frac{r_{i''}+1}{s_1}) \cap [1, \frac{r_{i''+u-1}+1}{s_u})$ can be determined in $O(1)$ time, which means SOR can be constructed in $O(w)$ time. □

By Lemmas 3, 4 and 5, our result is given as follows.

Theorem 4. *Given a pattern P and a text T, one can preprocess T with $O(|T|^2)$ time and $O(|T|^2)$ space, so that for $\alpha \geqslant 1$, all matched positions of $\delta_\alpha(P)$ in T can be determined in $O(|P|+w)$ time, where $w \leqslant U_r$ denotes the number of dominant positions and U_r denotes the number of matched positions.*

For large alphabets, which means that $|\Sigma|$ is not a constant, our algorithm can be implemented by replacing T_{SR} with T_{AR}, where T_{AR} denotes the suffix array of T_R. After an $O(|T|\log|T|)$-time conversion of T into a string over the integer alphabet $\{1, 2, \ldots, |T|\}$, based on related techniques of suffix array [12,14], one can construct T_{AR} in $O(|T|^2)$ time. As a result, for large alphabets, our indexing algorithm takes $O(|T|^2)$ time and space in its preprocessing phase, and reports all matched positions in $O(|P|+w+\log|T|)$ time.

4. Implementation of other scaling functions

In this section, we shall consider two other important scaling functions, which are $\delta_\alpha(P) = p_1^{\lceil \alpha s_1 \rceil} p_2^{\lceil \alpha s_2 \rceil} \cdots p_u^{\lceil \alpha s_u \rceil}$ and $\delta_\alpha(P) = p_1^{\lfloor \alpha s_1 + 0.5 \rfloor} p_2^{\lfloor \alpha s_2 + 0.5 \rfloor} \cdots p_u^{\lfloor \alpha s_u + 0.5 \rfloor}$. One can easily verify that with a minor adaptation, our indexing algorithm is still applicable. Therefore, in the following, we only give our adaptation, but omit detailed proofs.

For $\delta_\alpha(P) = p_1^{\lceil \alpha s_1 \rceil} p_2^{\lceil \alpha s_2 \rceil} \cdots p_u^{\lceil \alpha s_u \rceil}$, the valid scale for P at the ith position in T' is $\bigcap_{j=2}^{u-1} (\frac{r_{i+j-1}-1}{s_j}, \frac{r_{i+j-1}}{s_j}] \cap [1, \frac{r_i}{s_1}] \cap [1, \frac{r_{i+u-1}}{s_u}] = (max_{j=2}^{u-1} \frac{r_{i+j-1}-1}{s_j}, min_{j=2}^{u-1} \frac{r_{i+j-1}}{s_j}] \cap [1, \frac{r_i}{s_1}] \cap [1, \frac{r_{i+u-1}}{s_u}]$. Based on this change, the function $f(r_j, \alpha_k) = \beta$ should be redefined as the largest integer such that $\lceil \beta \alpha_k \rceil \leqslant r_j$.

In the case $\delta_\alpha(P) = p_1^{\lfloor \alpha s_1 + 0.5 \rfloor} p_2^{\lfloor \alpha s_2 + 0.5 \rfloor} \cdots p_u^{\lfloor \alpha s_u + 0.5 \rfloor}$, for P at the ith position in T', one can obtain the valid scale $\bigcap_{j=2}^{u-1} [\frac{r_{i+j-1}-0.5}{s_j}, \frac{r_{i+j-1}+0.5}{s_j}) \cap [1, \frac{r_i+0.5}{s_1}) \cap [1, \frac{r_{i+u-1}+0.5}{s_u}) = [max_{j=2}^{u-1} \frac{r_{i+j-1}-0.5}{s_j}, min_{j=2}^{u-1} \frac{r_{i+j-1}+0.5}{s_j}) \cap [1, \frac{r_i+0.5}{s_1}) \cap [1, \frac{r_{i+u-1}+0.5}{s_u})$. Therefore, the function $f(r_j, \alpha_k) = \beta$ should be redefined as the largest integer such that $\lfloor \beta \alpha_k + 0.5 \rfloor \leqslant r_j$. In addition, we have $\Gamma(T) = (\bigcup_{i=1}^{m} \bigcup_{j=1}^{r_i} \frac{r_i - 0.5}{j}) \cap [1, \infty]$.

5. Conclusions and future work

To summarize, in Table 1 we list various algorithms for finding one-dimensional real scaled patterns, including previous and our results. In this table, the terms "Time" and "Space" denote the required time and space for preprocessing, respec-

tively. Also, the term "Decision" denotes the required time for the decision problem whether P can be r-matched in T. Finally, we use the term "Position" to represent the time spent on finding all positions where P can be r-matched in T. One should note that the algorithm proposed by Amir et al. [2] is not an indexing algorithm. In Table 1, one can see that our preprocessing is more efficient than the previous results proposed by Wang et al. We also improve the searching time of all matched positions from $O(|P| + U_r)$ to $O(|P| + w)$, where $w \leqslant U_r$. For other scaling functions, such as rounding to the nearest integer [20], our indexing algorithm is still applicable.

Recently, Amir et al. propose a new scaling function [4,5], which is considered more precise and realistic in the field of computer vision. For this new scaling function, Amir et al. give the best known matching algorithm [5], which takes $O(|T|\log|P| + \sqrt{|T|\log|P|}|P|^{\frac{3}{2}})$ time. Then, they extend their result to two-dimensional real-scale matching [5], deriving an $O(|T||P|^3 + |T|^2|P|\log|P|)$-time matching algorithm, where the text and the pattern are represented by a $|T| \times |T|$ matrix and a $|P| \times |P|$ matrix, respectively. Hence, for this new scaling function, to improve the matching algorithms and to propose efficient indexing algorithms are both important future studies.

References

[1] M.I. Abouelhoda, E. Ohlebusch, S. Kurtz, Optimal exact string matching based on suffix arrays, in: Proceedings of the 9th International Symposium on String Processing and Information Retrieval, Lisbon, Portugal, 2002, pp. 31–43.
[2] A. Amir, A. Butman, M. Lewenstein, Real scaled matching, Inform. Process. Lett. 70 (4) (1999) 185–190.
[3] A. Amir, G.M. Landau, U. Vishkin, Efficient pattern matching with scaling, J. Algorithms 13 (1992) 2–32.
[4] A. Amir, A. Butman, M. Lewenstein, E. Porat, Real two-dimensional scaled matching, Algorithmica 53 (3) (2009) 314–336.
[5] A. Amir, A. Butman, M. Lewenstein, E. Porat, D. Tsur, Efficient one dimensional real scaled matching, J. Discrete Algorithms 5 (2) (2007) 205–211.
[6] A. Amir, G. Călinescu, Alphabet-independent and scaled dictionary matching, J. Algorithms 36 (1) (2000) 34–62.
[7] A. Amir, M. Lewenstein, E. Porat, Faster algorithms for string matching with k mismatches, J. Algorithms 50 (2) (2004) 257–275.
[8] M.A. Bender, M. Farach-Colton, The LCA problem revisited, in: Proceedings of Latin American Theoretical Informatics, Punta del Este, Uruguay, 2000, pp. 88–94.
[9] R. Boyer, J. Moore, A fast string searching algorithm, Commun. ACM 20 (10) (1977) 762–772.
[10] P. Clifford, R. Clifford, Simple deterministic wildcard matching, Inform. Process. Lett. 101 (2007) 53–54.
[11] M. Farach-Colton, P. Ferragina, S. Muthukrishnan, On the sorting-complexity of suffix tree construction, J. ACM 47 (6) (2000) 987–1011.
[12] D.K. Kim, J.S. Sim, H. Park, K. Park, Constructing suffix arrays in linear time, J. Discrete Algorithms 3 (2–4) (2005) 126–142.
[13] D. Knuth, J. Morris, V. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (1) (1977) 323–350.
[14] P. Ko, S. Aluru, Space efficient linear time construction of suffix arrays, J. Discrete Algorithms 3 (2–4) (2005) 143–156.
[15] U. Manber, G. Myers, Suffix arrays: A new method for on-line string searches, SIAM J. Comput. 22 (5) (1993) 935–948.
[16] E.M. McCreight, A space-economical suffix tree construction algorithm, J. ACM 23 (2) (1976) 262–272.
[17] S. Muthukrishnan, Efficient algorithms for document retrieval problems, in: Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2002, pp. 657–666.
[18] Y.-H. Peng, C.-B. Yang, K.-S. Huang, H.-Y. Ann, Efficient preprocessings on the one-dimensional discretely scaled pattern matching problem, in: Proceedings of the 24th Workshop on Combinatorial Mathematics and Computation Theory, Puli, Nantou, Taiwan, 2007, pp. 35–43.
[19] E. Ukkonen, On-line construction of suffix trees, Algorithmica 14 (3) (1995) 249–260.
[20] B.-F. Wang, J.-J. Lin, S.-C. Ku, Efficient algorithms for the scaled indexing problem, J. Algorithms 52 (1) (2004) 82–100.
[21] P. Weiner, Linear pattern matching algorithms, in: Proceedings of the 14th IEEE Symposium on Switching and Automata Theory, University of Iowa, 1973, pp. 1–11.