Information Processing Letters 123 (2017) 21–26

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

On approximate pattern matching with thresholds ☆

Peng Zhang [a,*,1], Mikhail J. Atallah [b]

[a] School of Computer Science, Georgia Tech, United States
[b] Department of Computer Science, Purdue University, United States

ARTICLE INFO

Article history:
Received 8 April 2015
Received in revised form 9 January 2017
Accepted 5 March 2017
Available online 9 March 2017
Communicated by X. Wu

Keywords:
Design of algorithms
Pattern matching
Threshold
Recursion

ABSTRACT

In the traditional version of the problem of approximate pattern matching, a pattern symbol is considered to match a text symbol if the two symbols are equal. Such a notion of exact equality is not suitable for situations where the text and pattern symbols are imprecise, e.g., obtained from an analog source, distorted by additive noise, etc. In such situations it is more appropriate to consider two alphabet symbols to match *even if they are not equal*, as long as they do not differ by more than a given threshold θ. The goal is then to compute the number of matches of the length-M pattern with all length-M substrings of the length-N text, i.e., to compute a vector of $N-M+1$ scores, where the ith score is the number of matches between the pattern and the substring that begins at text position i. The main result of this paper is to show that this threshold version of the problem can be solved by recursively solving $3+2\log\theta$ instances of the traditional (i.e., zero-threshold) version of the problem, which is much-studied in the literature and for which there are many efficient (typically randomized) solutions of time complexity close to $O(N\log M)$. This paper's result therefore implies the first randomized $O(N\log M(\log\theta+1))$ solution for the threshold version of the problem. It also implies that any future improvement to the traditional (zero-threshold) version of the problem automatically translates into a similar improvement to the arbitrary-threshold case. Furthermore, we show that the factor $\Omega(\log\theta)$ is tight if use our recursive framework.

1. Introduction

Recall that, in the approximate pattern matching problem, the input is a pair of strings: a text string $T=t_1\cdots t_N$, and a pattern string $P=p_1\cdots p_M$, both over an alphabet Σ. The desired output is a similarity *score vector* S of length $N-M+1$ such that

$$S[i]=\sum_{k=1}^{M}\delta(t_{i+k-1},p_k)$$

for $1\le i\le N-M+1$, where $\delta(x,y)$ is 1 if $x=y$ and is 0 if $x\neq y$. This problem has been widely studied in theory, and has a variety of applications in DNA sequence analysis, web search, image processing, and computer vision, etc.

The best known deterministic algorithm for this problem takes $O(N\sqrt{M\log M})$ time [1,16]. Efficient randomized algorithms of time complexity close to $O(N\log M)$ were given in [3,5]. Besides, [15] gave algorithms which compute each score with an ϵ relative error in time

☆ Portions of this work were supported by National Science Foundation Grant CPS-1329979, Qatar National Research Fund Grant NPRP X-063-1-014, and by sponsors of the Center for Education and Research in Information Assurance and Security. The statements made herein are solely the responsibility of the authors.

* Corresponding author.
E-mail addresses: pzhang60@gatech.edu (P. Zhang), mja@cs.purdue.edu (M.J. Atallah).

[1] The work was done while the first author was at Purdue University.

http://dx.doi.org/10.1016/j.ipl.2017.03.001
0020-0190/

$O((N/\epsilon^2)\log^c M)$ for a small constant c. The entries of S for which the similarity score is at least $M-k$ are known as k-mismatches. Finding only k-mismatches can be done efficiently with an $O(N\sqrt{k\log k})$ time algorithm [2], which was further improved to $O(N+N\sqrt{k/\log N}\log k)$ [12]. Although in the worst case k could be proportional to M, in practice one is often only interested in small values of k. There had been much prior work on this k-mismatch problem (e.g., [13,6], to mention a few). A recent refinement to the shift-add approach of [6] was given in [14].

In [4] a generalization of the pattern matching problem was studied, considering that the text pattern symbols might be imprecise if obtained from an analog source or distorted by additive noise, etc. They computed the score vector S for the case where the input also includes a *threshold* value θ whose significance is that alphabet symbols whose difference does not exceed θ are considered to match. In other words, in the above definition of the score vector S, the δ function is replaced by a δ_θ function where $\delta_\theta(x,y)$ is 1 if $|x-y|\leq\theta$ and is 0 if $|x-y|>\theta$. For example, if the alphabet is $\Sigma=\{0,1,2,\ldots,9\}$, $\theta=2$, $T=6482$, and $P=93$, then $S=(1,0,2)$. The algorithm given in [4] has an $O(N\sqrt{M\log M})$ time complexity. Unlike the situation for the classical (zero-threshold) version of the problem, randomized more efficient (e.g., close to $O(N\log M)$) time bounds for this problem were not known. Such an efficient solution is a corollary to the main result of the present paper.

Another related but different variant of the pattern matching problem is δ-matching, studied broadly in [7–11]. Instead of outputting a score vector of length $N-M+1$, δ-matching problem outputs a subset of indices $\mathcal{I}_\delta=\{i:\max_{k=1}^{M}|t_{i+k-1}-p_k|\leq\delta\}$ where δ is a given threshold. Note that if we use δ as threshold in our setting, $\mathcal{I}_\delta$ is exactly the subset of score vector indices without mismatching, that is, $\{i:S[i]=M\}$. Efficient algorithms for δ-matching problem are given in [7–10]. However, these algorithms are not applicable to the problem studied in this paper, since they lack the similarity scores for indices with mismatches.

2. This paper's contribution

The main result of our present paper is to establish that any algorithm of complexity $g(N,M)$ for the traditional (i.e., zero-threshold) version of the problem implies an algorithm of complexity $O(g(N,M)(\log\theta+1))$ for the arbitrary-threshold version of the problem. We establish this by giving an algorithm that solves the with-threshold version of the problem using $3+2\log\theta$ zero-threshold problem instances that have same size (i.e., N and M) as the with-threshold instance (the time spent on creating each instance is linear, hence dominated by $g(N,M)$).

It is easy to see that trying to process, one at a time, the bits of the binary representation of θ, does not work and leads to a dead-end. Our solution bears no resemblance to such an approach. A rough overview of what we do is as follows. We use a recursive approach where, after solving two judiciously defined zero-threshold instances of the problem, we recurse on a problem whose threshold is $\theta/2$. The recursion is on new versions of text and pattern, obtained from the input versions by replacing symbols by other symbols, in a way that resembles an odd–even un-shuffle of a bucketized alphabet (hence some symbols of the alphabet get replaced by much larger ones, whereas other symbols get replaced by much smaller ones). The process we use for replacing a symbol i depends on the *parity* of $\lfloor\frac{i}{\theta/2}\rfloor$, and the arithmetic used in both the even and odd cases involves i, θ, and (for the odd case) $\max_j(\lfloor\frac{j}{\theta/2}\rfloor)$ where the maximization is over all alphabet symbols j that occur in text or pattern. Furthermore, we show that the factor $\Omega(\log\theta)$ is tight if use the above recursive framework.

3. The algorithm

Before stating our algorithm, we formally define the problem of approximate pattern matching with thresholds as follows.

Inputs: A text string $T=t_1\cdots t_N$, a pattern string $P=p_1\cdots p_M$, both over a (possibly large) alphabet. A threshold value θ.

Output: A vector S of length $N-M+1$, where $S[i]=\sum_{k=1}^{M}\delta_\theta(t_{i+k-1},p_k)$ and $\delta_\theta(x,y)$ is 1 if $|x-y|\leq\theta$, 0 otherwise.

This section gives a recursive algorithm that solves the problem of approximate pattern matching with thresholds by making use of $3+2\log\theta$ instances of zero-threshold versions of the problem. As the zero-threshold version is much studied in the literature, and has efficient (typically randomized) algorithms for solving it, our result implies efficient algorithms for the threshold version of the problem.

In what follows we assume alphabet symbols and the threshold to be *integers*. There is no loss of generality in this, because otherwise they can all be "scaled up" to become integers without changing the resulting score vector. For example, if the original alphabet is $\{1.73, 2.55, 5.41, 2.17\}$ and θ is 0.13, then after scaling they become $\{173, 255, 541, 217\}$ and (respectively) 13.

To avoid unnecessarily cluttering the exposition, we present the algorithm when θ is a power of 2; there is no loss of generality in doing so, as the algorithm we give can easily be modified for the general case.

`Algorithm Scores(`T,P,θ`)`

Method: The algorithm is recursive, and works by making two calls to zero-threshold instances of the problem, followed by a recursive call on a problem instance having a threshold of $\theta/2$.

1. If $\theta=1$ then do the following.
   (a) Let S_0 be the score vector returned by `Scores(`T, P,0).
   (b) Obtain T' from T by replacing every occurrence of a symbol i in T with $i+1$. Let S_1 be the score vector returned by `Scores(`T',P,0).
   (c) Obtain P' from P by replacing every occurrence of a symbol i in P with $i+1$. Let S_2 be the score vector returned by `Scores(`T,P',0).
   (d) Return $S_0+S_1+S_2$.