Theoretical Computer Science 638 (2016) 3–10

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Set Intersection and Sequence Matching with mismatch counting [☆]

Ariel Shiftan [*], Ely Porat [*]

Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel

ARTICLE INFO

Article history:
Received 2 April 2015
Received in revised form 3 January 2016
Accepted 7 January 2016
Available online 12 January 2016

Keywords:
Set Intersection Matching
Sequence Matching
Pattern matching
Generalized strings

ABSTRACT

In the classical pattern-matching problem, one is given a text and a pattern both of which are sequences of letters. The requirement is to find all occurrences of the pattern in the text. We studied two modifications of the classical problem, where each letter in the text and pattern is a set (*Set Intersection Matching* problem) or a sequence (*Sequence Matching* problem). Two "letters" are found to match if the intersection of the corresponding sets is not empty or if the two sequences have a common element in the same index. We first show that the two problems are similar by establishing a linear time reduction between them. We then show the first known non-trivial and efficient algorithms for these problems, when the maximum set/sequence size d is small. The first is a Monte Carlo randomized algorithm for *Set Intersection Matching*, that takes $\Theta\left(4^d n \log n \log m\right)$ time, where n and m are the lengths of the text and the pattern, respectively; the failure probability is less than $\frac{1}{n^2}$. This algorithm can also be used, with slight modifications, when up to k mismatches is allowed. In addition, it can be used to maintain an approximation of factor $1 \pm \epsilon$ of the mismatch count in $\Theta\left(\frac{1}{\epsilon^2} 4^d n \log n \log m\right)$ time; the failure probability is bounded by $\frac{1}{n}$. The second is a deterministic algorithm for *Set Intersection Matching* that can be used to count the number of matches at each index of the text in a total running time $\Theta\left(\sum_{i=1}^{d} \binom{\sigma}{i} n \log m\right) = O\left(\sigma^d n \log m\right)$, where σ is the size of the alphabet. The third algorithm, also deterministic, solves the *Sequence Matching* problem in $\Theta\left(4^d n \log m\right)$ time.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction and related work

The classic pattern matching problem is defined as follows: We are given a text $T = t_0, t_1, t_2, \ldots, t_{n-1}$ of size n and a pattern $P = p_0, p_1, p_2, \ldots, p_{m-1}$ of size m — both are sequences of letters belonging to a pre-defined set, i.e., the alphabet Σ of size σ. We are required to find all occurrences of the pattern in the text. Linear time solutions were given in [10,7].

Two forms of approximation for the problem that are commonly researched involve *don't cares* and *mismatch count*. The first is the presence of a wildcard letter that matches any other letter. One of the main approaches for solving that problem, that was first introduced by Fischer and Paterson, is to use convolutions. They showed how the problem can be

[☆] This work is part of Ariel Shiftan's PhD thesis done in the faculty of Computer Science, Bar-Ilan University.
[*] Corresponding authors.
E-mail addresses: shiftaa@cs.biu.ac.il (A. Shiftan), porately@cs.biu.ac.il (E. Porat).

http://dx.doi.org/10.1016/j.tcs.2016.01.003
0304-3975/© 2016 Elsevier B.V. All rights reserved.

solved in $\Theta(n \log m \log \sigma)$ time [4]. Since then many algorithms [11–14] showed how different string matching problems can be solved efficiently by using FFT. Faster algorithms for the string matching with wildcards were presented later, and the time was reduced to $\Theta(n \log m)$ [9,8]. The second form of approximation is match/mismatch count, e.g. the number of matching letters when comparing the pattern to the text at location i for all locations. This mismatch count, at each location compared, is also known as the Hamming distance between the text and the pattern at this location. A well known algorithm for finding the hamming in $\Theta(n\sigma \log m)$ time was given in [4].

The subset matching problem was first introduced by Cole and Hariharan [1]. It extends the classic string-matching problem and defines both the pattern and text to be sequences of sets of characters. Formally, each text location t_i and each pattern location p_i is a set of characters, not a single character, taken from a certain alphabet. Pattern P is said to match text T at location i, if $p_j \subseteq t_{i+j}$ for all j, $0 \le j < m$. Cole and Hariharan proposed a near-linear time randomized algorithm [1] and improved it [2] to a deterministic one. Amihood, Porat and Lewenstein proposed an approximate version with don't cares [3].

Generalized strings are another extensions of the classical string-matching problem. Assuming the alphabet is Σ, *Generalized strings* are sequences of sets, where each set is a subset of Σ that are possibilities for letters in that position. For example, the *generalized string* $[\{a,b\},\{b,d\}]$ matches the strings ab, ad, bb, and bd. This type of string is used in [6] and while specifying the pattern to look for regular expressions.

As in the subset matching and *Generalized strings* problems, the two problems studied in this paper define the pattern and the text to be sequences of sets (or sequences). But here, a match between two 'letters' is when either the intersection of the two sets is not empty, as in the *Set Intersection Matching* problem, or when the two sequences have a common element in the same place, as in the *Sequence Matching* problem. In the former, a search is conducted to find one *generalized string* inside another. Assuming the maximum set (or sequence) size is d, it is trivial that these problems can be easily solved in $\Theta\left(nmd^2\right)$ and $\Theta(nmd)$ time, respectively.

Sample applications for the algorithms in this paper occur when there are possible errors in both the pattern and the text. For example, consider the case when both the pattern and the text were acquired using an Optical Character Recognition (OCR) algorithm that reports few options for each letter. One can create a set of all possible letters for each such letter and then run a *Set Intersection Matching* algorithm to get the occurrences of such faulty patterns in the text.

1.1. Our contribution

The difficulty in both problems lies in the lack of the transitive property, which is the base for fast pattern-matching algorithms. The transitive relation states that if $a = b$ and $b = c$, then $a = c$. This is utilized by classical pattern-matching algorithms to avoid comparing elements in which matching could be concluded. Because this property does not hold in the *Set Intersection Matching* and *Sequence Matching* problems, the existing algorithms cannot be used.

In this paper, we show the first known non-trivial and efficient algorithms for these problems when the maximum set/sequence size d is small. First, we show a Monte Carlo algorithm for *Set Intersection Matching* that takes $\Theta\left(4^d n \log n \log m\right)$ time. It has a failure probability less than $\frac{1}{n^2}$. This algorithm can be used to maintain an approximation of factor $1 \pm \epsilon$ of the mismatch count in $\Theta\left(\frac{1}{\epsilon^2} 4^d n \log n \log m\right)$ time. The failure probability is bounded by $\frac{1}{n}$. It improves upon the trivial solution where $d - 2\log d < \log m - \log\log n - \log\log m$. Assuming $\log n = o(m)$, it improves for $d < \frac{1}{2}\log m$. Next, we present a deterministic algorithm for *Set Intersection Matching*, that can be used to count the number of matches at each index of the text in a total running time $\Theta\left(\sum_{i=1}^{d} \binom{\sigma}{i} n \log m\right) = O\left(\sigma^d n \log m\right)$. This algorithm improves upon the trivial solution where $d < \frac{\log m}{\log \sigma}$. Finally, a $\Theta\left(4^d n \log m\right)$ time deterministic algorithm for the *Sequence Matching* problem is given, which improves upon the trivial algorithm for $d < \frac{1}{2}\log m$.

1.2. Overview and comparison

The structure of this paper is as follows. Section 2 formally defines each of the two problems. Sections 3 and 4 present three algorithms for the problems.

The two algorithms presented in Section 3 are for the *Set Intersection Matching* problem, and both can be used for determining the mismatch count. The first is randomized, approximates the mismatch count with high probability, and is more suitable for cases where the size of the alphabet σ is large, whereas the second is deterministic and gives the exact mismatch count but is less efficient when σ is large. The algorithm given in Section 4 is for the *Sequence Matching* problem, it is deterministic, and efficient for a large alphabet σ, but cannot be used to determine the mismatch count.

2. Preliminary

In the *Set Intersection Matching* problem, each pattern and text location is a set, and two locations match if the intersection between the sets is not empty. More formally: