Engineering Applications of Artificial Intelligence 51 (2016) 109–114

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Linear algorithm for conservative degenerate pattern matching

Maxime Crochemore, Costas S. Iliopoulos, Ritu Kundu, Manal Mohamed *, Fatima Vayani

Department of Informatics, King's College London, London, UK

ARTICLE INFO

Available online 28 January 2016

Keywords:
Degenerate string
Pattern matching
Algorithm

ABSTRACT

A *degenerate symbol* $\tilde{x}$ over an alphabet Σ is a non-empty subset of Σ, and a sequence of such symbols is a *degenerate string*. A degenerate string is said to be *conservative* if its number of non-solid symbols is upper-bounded by a fixed positive constant k. We consider here the matching problem of conservative degenerate strings and present the first linear-time algorithm that can find, for given degenerate strings $\tilde{P}$ and $\tilde{T}$ of total length n containing k non-solid symbols in total, the occurrences of $\tilde{P}$ in $\tilde{T}$ in $O(nk)$ time.

1. Introduction

Degenerate or *indeterminate* strings are found in Biology, Musicology, Cryptography, and Data-mining and Web-mining applications. They are defined by the existence of one or more positions which are represented by sets of symbols, instead of a single symbol. In *conservative degenerate strings*, the number of such positions is bounded by k. In music, single notes may match chords. In encrypted and biological sequences, a position in one string may match exactly with various symbols in other strings. Previous algorithmic research of degenerate strings has been focused on pattern matching.

Typographical mistakes, different spellings of the same words based on locale, different formatting conventions, or data transformation errors give rise to inconsistencies in the string data-sets used by data-mining or web-mining applications. Therefore, one symbol of the data-set may match with a set of symbols in the query string while searching or retrieving information in such cases. For example, 'analyse' and 'analyze' have same semantic meaning but are spelled differently in British and American English; thus a query involving the word can be represented as a degenerate string with the set $\{s, z\}$ at the 6th position. Some other domains where pattern matching using degenerate strings may lead to better performance include, but are not limited to, query suggestions in search-engines, spell-checking, spam filtering, and product-search and recommendation in e-commerce applications.

Pattern matching in degenerate strings is particularly relevant in the context of coding biological sequences. Due to the degeneracy of the genetic code, two dissimilar DNA sequences can be translated into two identical protein sequences. Without taking this degeneracy into account, many associations between biological entities can be overlooked. For example, the following six DNA codons are all translated into the amino acid Leucine: *TTA*, *TTG*, *CTT*, *CTC*, *CTA* and *CTG*. This example highlights the significance of solving problems relating to degeneracy in strings. In fact, special symbols to represent sets of DNA symbols have long been established by the IUPAC-IUBMB Biochemical Nomenclature Committee (Cornish-Bowden, 1985). For example, *R* represents any purine (*A* or *G*), *Y* represents any pyrimidine (*C*, *T* or *U*) and *N* represents any nucleic acid. An example of practical implications of such research is in the design of primers for cloning DNA sequences using PCR (Polymerase Chain Reaction). Degenerate primers are used when their design is based on protein sequences, which can be reverse-translated to n^k different sequences, where n is the length of the sequence.

Another example, focused instead on protein sequences, is sequence motif searching. Sequence motifs are conserved patterns found when aligning proteins which have similar functions. The motifs, therefore, are considered to be important functional domains of the proteins. PROSITE (Sigrist et al., 2013) is a database which stores such motifs, detailing their sequence signatures, listing the proteins in which they occur, and describing their functions (if known). The following is an example of a motif signature, taken from PROSITE (PDOC00930): FED[LV]IA[DE][PA], where a set of possible symbols for a degenerate position is given in square brackets. This motif is found in caveolins, a family of membrane proteins. Currently, this family consists of 80 known proteins, 74 of which contain this motif.

This paper introduces an algorithm which is a significant improvement from those published previously. The first significant contribution for the problem of pattern matching of degenerate strings was in 1974 (Fischer and Paterson, 1974), and was later improved (Muthukrishnan and Palem, 1994). Later still, faster algorithms for the same problem were proposed (Indyk, 1998; Kalai, 2002). Since, many practical methods have been suggested (Holub et al., 2008; Smyth and Wang, 2009; Rahman et al., 2007), as well as variations of the problem considered. For example, a non-practical generalised string matching algorithm was introduced by Abrahamson (1987). Most recently, Crochemore et al. (2014) reported an

* Corresponding author.

http://dx.doi.org/10.1016/j.engappai.2016.01.009
0952-1976/

algorithm to find the shortest solid cover in a degenerate string with time complexity $O(2^k)$. We report here a major improvement in time: $O(kn)$. Further to the problem of pattern matching, the linear algorithm reported here can be applied to many different problems, including finding cover and prefix arrays.

The rest of the paper is organised in the following format: The next section introduces the vocabulary and the notions that will be used in this paper. Section 3 formally defines the problem and presents the algorithm we have proposed. The algorithm is analysed in Section 4 and experimental results are presented in Section 5. Finally, Section 6 concludes the paper.

2. Preliminaries

To provide an overview of our results we begin with a few definitions, generally following Rahman et al. (2007) and Crochemore et al. (2014). An *alphabet* Σ is a non-empty finite set of symbols of size $|\Sigma|$. A *string* over a given alphabet is a finite sequence of symbols. The *length* of a string x is denoted by $|x|$. The *empty string* is denoted by ε. The set of all strings over an alphabet Σ (including empty string ε) is denoted by Σ^*.

A *degenerate symbol* $\tilde{x}$ over an alphabet Σ is a non-empty subset of Σ, i.e., $\tilde{x} \subseteq \Sigma$ and $\tilde{x} \neq \emptyset$. $|\tilde{x}|$ denotes the size of the set and we have $1 \leq |\tilde{x}| \leq |\Sigma|$. A finite sequence $\tilde{X} = \tilde{x}_1\tilde{x}_2\ldots\tilde{x}_n$ is said to be a *degenerate string* if $\tilde{x}_i$ is a degenerate symbol for each i from 1 to n. In other words, a *degenerate string* is built over the potential $2^{|\Sigma|}-1$ non-empty sets of letters belonging to Σ. The number of the degenerate symbols, n here, in a degenerate string $\tilde{X}$ is its *length*, denoted as $|\tilde{X}|$. For example, $\tilde{X} = [a,b][a][c][b,c][a][a,b,c]$ is a degenerate string of length 6 over $\Sigma = [a,b,c]$. If $|\tilde{x}_i| = 1$, that is, $\tilde{x}_i$ represents a single symbol of Σ, we say that $\tilde{x}_i$ is a *solid* symbol and i is a *solid position*. Otherwise $\tilde{x}_i$ and i are said to be *non-solid symbol* and *non-solid position* respectively. For convenience we often write $\tilde{x}_i = c$ ($c \in \Sigma$), instead of $\tilde{x}_i = [c]$, in case of solid symbols. Consequently, the degenerate string $\tilde{X}$ mentioned in the example previously will be written as $[a,b]ac[b,c]a[a,b,c]$. A string containing only solid symbols will be called a *solid string*. Also as a convention, capital letters will be used to denote strings while small letters will be used for representing symbols. Furthermore, the degeneracy will be indicated by a *tilde*, for example, $\tilde{X}$ denotes a *degenerate string* while a plain letter like X represents a *solid string*. The empty degenerate string is denoted by $\tilde{\varepsilon}$.

A *conservative* degenerate string is a degenerate string where its number of non-solid symbols is upper-bounded by a fixed positive constant k. The concatenation of degenerate strings $\tilde{X}$ and $\tilde{Y}$ is $\tilde{X}\tilde{Y}$. A degenerate string $\tilde{V}$ is a *substring* (resp. *prefix*, *suffix*) of a degenerate string $\tilde{X}$ if $\tilde{X} = \tilde{U}\tilde{V}\tilde{W}$ (resp. $\tilde{X} = \tilde{V}\tilde{W}$, $\tilde{X} = \tilde{U}\tilde{V}$) for some degenerate strings $\tilde{U}$ and $\tilde{W}$. By $\tilde{X}[i..j]$, we represent a substring $\tilde{x}_i\tilde{x}_{i+1}\ldots\tilde{x}_j$ of $\tilde{x}$.

For degenerate strings, the notion of symbol equality is extended to single-symbol *match* between two degenerate symbols in the following way. Two degenerate symbols $\tilde{x}$ and $\tilde{y}$ are said to *match* (represented as $\tilde{x} \approx \tilde{y}$) if $\tilde{x} \cap \tilde{y} \neq \emptyset$. Extending this notion to degenerate strings, we say that two degenerate strings $\tilde{X}$ and $\tilde{Y}$ *match* (denoted as $\tilde{X} \approx \tilde{Y}$) if $|\tilde{X}| = |\tilde{Y}|$ and corresponding symbols in $\tilde{X}$ and $\tilde{Y}$ match, i.e., for each $i = 1,\ldots,|\tilde{X}|$ we have $\tilde{x}_i \approx \tilde{y}_i$. Note that the relation $\approx$ is not transitive. A degenerate string $\tilde{X}$ is said to *occur* at position i in another degenerate (resp. solid) string $\tilde{Y}$ (resp. Y) if $\tilde{X} \approx \tilde{Y}[i\ddot{i}+|\tilde{X}|-1]$ (resp. $\tilde{X} \approx Y[i\ddot{i}+|\tilde{X}|-1]$).

3. Conservative degenerate string matching

Problem 1. Given a conservative degenerate pattern $\tilde{P}$ with k non-solid symbols, and a solid text T, find all positions in T at which $\tilde{P}$ occurs.

Example 1. We consider a degenerate pattern, $\tilde{P} = a[bc]da[bd]$ with $k=2$ and a text, $T=dacdabdadcabdac$. Table 1 shows that $\tilde{P}$ occurs in T at positions 2 and 5.

For convenience, we compute a table $Pre[k, |\Sigma|]$ such that for each non-solid position i ($1 \leq i \leq k$) and each letter $a \in \Sigma$, we have $Pre[i,a] = 1$ if $a \in \tilde{P}[i]$ and 0 otherwise. After such $O(k|\Sigma|)$-time preprocessing, we can check in $O(1)$ time whether a non-solid position in $\tilde{P}$ matches a position in T or not.

3.1. An outline of our approach

Our algorithm to solve Problem 1 is built on the top of an adapted version of the sequential algorithm presented by Landau and Vishkin to find all occurrences of a (solid) pattern P of length m in a (solid) text T of length n with at most e differences each (Landau and Vishkin, 1989), where a difference can be due to either a mismatch between the corresponding characters of the text and the pattern, or a superfluous character in the text, or a superfluous character in the pattern. The modification required for our strategy is to treat only mismatches as the differences in Landau and Vishkin's algorithm. On the lines of the original Landau and Vishkin's algorithm, the modified one works in the following two steps:

1. *Step* 1: Compute the suffix tree of the string obtained after concatenating the text, the pattern and a character # which is not present in $\Sigma \cup \Lambda$, i.e., $TP\#$; using the serial algorithm of Weiner (1973).
2. *Step* 2: Let $Mismatch_{i,j}$ be the position in the pattern at which we have jth mismatch (when defined) between $T[i+1\ddot{i}+m]$ and $P[1\dot{m}]$. In other words, $Mismatch_{i,j} = f$ represents jth mismatch from left to right and implies that $t_{i+f} \neq p_f$. In this step, we find $Mismatch_{i,j}$ for each i and j such that $0 \leq i \leq n-m$ and $1 \leq j \leq c+1$ where c denotes the maximum of the two: e and the total number of mismatches between $T[i+1\ddot{i}+m]$ and $P[1\dot{m}]$. If some $Mismatch_{i,j} = m+1$, it signifies that there is an occurrence of the pattern in the text, starting at $t[i+1]$, with at most e mismatches. $Mismatch_{i,j}$ can be computed from $Mismatch_{i,j-1}$ as follows:
   Let $LCA_{si,sj}$ be the lowest common ancestor (in short LCA) of the leaves of the suffixes $T[si+1,n]$ and $P[sj+1]$ in the suffix tree and $|LCA_{si,sj}|$ denotes its length. $Mismatch_{i,j-1} = f$ implies that $T[i+1\ddot{i}+f]$ and $P[1\ddot{f}]$ is matched with $j-1$ mismatches. We want to find the largest q such that $T[i+f+1\ddot{i}+f+q] = P[f+1\ddot{f}+q]$

Table 1
Occurrence of $\tilde{P}$ in T.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t	d	a	c	d	a	b	d	a	d	c	a	b	d	a	c
Matches		a	$[bc]$	d	a a	$[bd]$ $[bc]$	d	a	$[bd]$						