

# Linear Time Algorithm for the Generalised Longest Common Repeat Problem

Inbok Lee\* and Yoan José Pinzón Ardila

King's College London, Dept. of Computer Science,  
London WC2R 2LS, United Kingdom  
inbok@dcs.kcl.ac.uk, Yoan.Pinzon@kcl.ac.uk

**Abstract.** Given a set of strings  $\mathcal{U} = \{T_1, T_2, \dots, T_\ell\}$ , the longest common repeat problem is to find the longest common substring that appears at least twice in each string of  $\mathcal{U}$ , considering direct, inverted, mirror as well as everted repeats. In this paper we define the generalised longest common repeat problem, where we can set the number of times that a repeat should appear in each string. We present a linear time algorithm for this problem using the suffix array. We also show an application of our algorithm for finding a longest common substring which appears only in a subset  $\mathcal{U}'$  of  $\mathcal{U}$  but not in  $\mathcal{U} - \mathcal{U}'$ .

**Keywords:** suffix arrays, pattern discovery, inverted repeats, DNA repeats, DNA Satellites.

## 1 Introduction

Genomic sequences are far from random. One of the more intriguing features of DNA is the extent to which it consists of repeated substrings. Repeated DNA sequences account for large portions of eukaryotic genomes that have been studied to date. The origin of these repeats, as well as their biological function, is not fully understood. Nevertheless, they are believed to play a crucial role in genome organization and evolution [3].

There are basically four repeat sequences types, differing by their orientation and localization. They are: direct, inverted, mirror and everted repeats. The easiest to understand is the direct repeat, in which a substring is duplicated (see Fig. 1(a)). Some multiple direct repeats have been associated with human genetic diseases. For example, the triplet<sup>1</sup> CGG is tandemly repeated 6 to 54 times in a normal FMR-1<sup>2</sup> gene. In patients with the Fragile X syndrome, the pattern occurs more than 200 times. People with this mutation fail to produce a protein involved in making cellular connections in the brain producing mental impediment or retardation. An estimated 1 in 2000 boys (girls are also affected, but the

---

\* This work was supported by the Korea Research Foundation Grant funded by Korean Government(MOEHRD, Basic Research Promotion Fund) M01-2004-000-20344-0.

<sup>1</sup> 3 DNA bp, a.k.a *microsatellites*.

<sup>2</sup> Fragile Mental Retardation 1.

incidence rate is lower) are mentally weakened because of Fragile X. Kennedy's disease, Parkinson's disease, Huntington's disease and Myotonic Dystrophy are examples of other genetic diseases that have been associated with direct repeats [4,19,21].

Inverted repeats are another important element in the Genomes. An inverted repeat, also called palindrome, is except that the second half of the repeat is also found nearby on the opposite strand (complementary strand) of the DNA helix, as shown in Fig. 1(b). This implies that the substring and its reverse complement are both to be found on the same strand, which can thus fold back to base-pair with itself<sup>3</sup> and form a stem-and-loop structure, as shown on the righthand side of Fig. 1(b). Because of their nature, inverted repeats engages in intra- and intermolecular base pairing. The ability to form hairpin, recursive (see Fig. 1(e)), cruciform (see Fig. 1(f)) and pseudo-knot (see Fig. 1(g)) secondary structures is associated with the initiation of DNA replication and frameshift mutations.

An application that makes repeats an interesting research topic is related to the *multiple alignment problem*, producing multiple alignments becomes very complicated when the sequences to be aligned contain multiple repeats, because matches may be present in numerous places. As a precursor to multiple alignment, it is helpful to recognize all multiple repeats within the set of strings that ought to be aligned [15]. Other applications of repetitive DNA analysis range from genetic markers, DNA fingerprinting, mapping genes, comparative genomics and evolution studies [3,8,7].

In this paper we consider the problem which combines two: finding common repetitive substrings in a set of strings. In addition, we want to specify the number of occurrences in each string. We focus on finding the longest substring since the substrings of the longest substring also appear in each string. We also consider reversed and reverse-complemented strings in finding repeats.

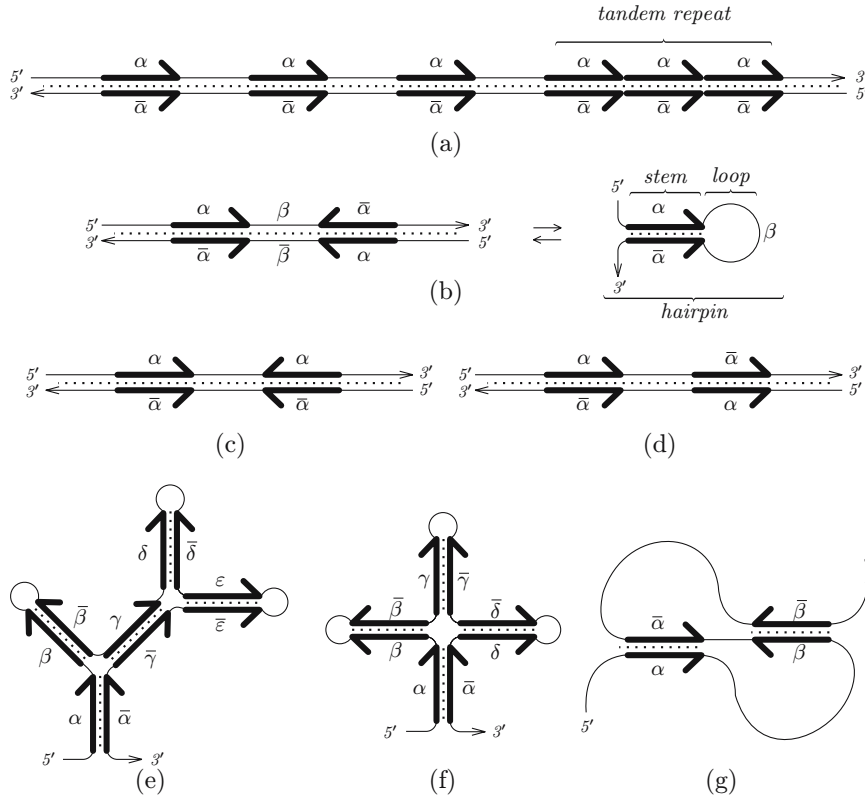
Before beginning, we will establish some notation. We will uniformly adopt the four letter alphabet  $\Sigma_{\text{DNA}} = \{\mathbf{a}, \mathbf{c}, \mathbf{g}, \mathbf{t}\}$ , each letter standing for the first letter of the chemical name of the *nucleotide* in the polymer's chain, and let a bar notation represent an operation corresponding to simple base complementarity, *i.e.* indicating bases that are able to physically base-pair between strands of double-helical DNA:

$$\bar{\mathbf{g}} = \mathbf{c}, \bar{\mathbf{c}} = \mathbf{g}, \bar{\mathbf{a}} = \mathbf{t}, \text{ and } \bar{\mathbf{t}} = \mathbf{a}$$

Let  $A$  be a string over  $\Sigma_{\text{DNA}}$ .  $A[i]$  denotes the  $i$ -th character of  $A$  and  $A[i..j]$  is the substring  $A[i]A[i+1]\cdots A[j]$  of  $A$ . We denote the *reverse* of  $A$  by  $\overleftarrow{A}$  and the *reverse complemented* of  $A$  by  $\overline{A}$ . Sequence  $\overline{A}$  is obtained by reversing  $A$  and mapping each character to its complement. If  $A = \mathbf{gtaac}$ , for example,  $\overleftarrow{A} = \mathbf{caatg}$  and  $\overline{A} = \overline{\mathbf{caatg}} = \mathbf{gttac}$ .

A *repeat* of  $A$  is a substring of  $A$  which appears at least twice in  $A$ . There are four kinds of repeats.

<sup>3</sup> Base-pairings within the same strand are called *secondary structures*.



**Fig. 1.** Different types of repetitions present in genomic sequences. (a) Direct repeat (left: non-tandem direct repeat, right: tandem direct repeats — when two repeats immediately follow each other in the string —). (b) Inverted repeat (left: single-stranded hairpin structure containing an inverted repeat). (c) Mirror repeat. (d) Everted repeat. (e) Recursive secondary structure containing several inverted repeats. (f) Cruciform structure formed at inverted repeats by four hairpins and the four-arm junction. (g) Nested double pseudo-knot structure including two inverted repeats.

- DIRECT REPEAT: A string  $p$  is called a *direct repeat* of  $A$  if  $p = A[i..i + |p| - 1]$  and  $p = A[i'..i' + |p| - 1]$ , for some  $i \neq i'$ .
- INVERTED REPEAT: A string  $p$  is called an *inverted repeat* of  $A$  if  $p = A[i..i + |p| - 1]$  and  $\bar{p} = A[i'..i' + |p| - 1]$ , for some  $i \neq i'$ .
- MIRROR REPEAT: A string  $p$  is called a *mirror repeat* (or *reverse repeat*) of  $A$  if  $p = A[i..i + |p| - 1]$  and  $\overleftarrow{p} = A[i'..i' + |p| - 1]$ , for some  $i \neq i'$ .
- EVERTED REPEAT: This type of repeat is equivalent to the direct repeat described above.

We will only consider direct, mirror and inverted repeats as standard prototypes. It is easy to see, that everted repeats are covered by direct repeats when considering multiple strings.

The generalised longest common repeat problem can subsequently be defined as follows.

*Problem 1.* Given a set of strings  $\mathcal{U} = \{T_1, T_2, \dots, T_\ell\}$ , a set of positive integers  $\mathcal{D} = \{d_1, d_2, \dots, d_\ell\}$ , and a positive integer  $k$ , the GENERALISED LONGEST COMMON REPEAT PROBLEM is to find the longest string  $w$  which satisfies two conditions: (a) There is a subset  $\mathcal{U}'$  of  $\mathcal{U}$  such that  $w$  appears at least  $d_i$  times in every string  $T_i$  in  $\mathcal{U}'$ , and (b)  $|\mathcal{U}'| = k$ .

This definition is an extension of that in [16]. The difference is that we can restrict the number of times that a repeat can appear in each string. Hence the frequency of the repeats will be bound according to our needs. Note that the above definition allows  $w$  to appear just once in some strings.

Karp, Miller, and Rosenberg first proposed an  $O(|T| \log |T|)$  time algorithm for finding the longest normal repeat in a text  $T$ . Also suffix trees [17,20,6] can be used to find it in  $O(|T|)$  time. Landau and Schmidt gave an  $O(k|T| \log k \log |T|)$  time algorithm for finding approximate squares where at most  $k$  edit distance is allowed [14]. Schmidt also gave an  $O(|T|^2 \log |T|)$  time algorithm for finding approximate tandem or non-tandem repeats [18]. Abouelhoda et al. [1] for finding various types of repeats using the enhanced suffix array.

In [16], a linear time algorithm for finding the longest common repeat problem using the generalised suffix tree was derived. Although it is the first approach for this problem, so far as we know, it has some drawbacks. First, it is not easy to implement. Second, it is based on the suffix tree data structure which is not memory-efficient. Besides, it requires a generalised suffix tree for all the strings and a suffix tree for each single string. Our new algorithm is easy to implement and requires only one suffix array<sup>4</sup>. And it can handle a generalised problem.

The outline of the paper is as follows: In Section 2 we explain some basics. In Section 3 we describe our algorithms for the generalised longest repeat problem. In Section 4 we explain an application of our algorithm, *i.e.* finding the longest repeat which appears only in a subset  $\mathcal{U}'$  of  $\mathcal{U}$  but not in  $\mathcal{U} - \mathcal{U}'$ . Conclusions are drawn in Section 5.

## 2 Preliminaries

The *suffix array* of a text  $T$  is a sorted array  $s[1..|T|]$  of all the suffixes of  $T$ . That is,  $s[k] = i$  iff  $T[i..|T|]$  is the  $k$ -th suffix of  $T$ . We also define the auxiliary *LCP array* as an array of the length of the longest common prefix between each substring in the suffix array and its predecessor, and define  $lcp(a, b) = \min_{a \leq i \leq b} lcp[i]$  with the following properties.

**Fact 1.**  $lcp(a, b) \leq lcp(a', b')$  if  $a \leq a'$  and  $b \geq b'$ .

**Fact 2.** The length of the longest common prefix of  $T[s[a]..|T|]$ ,  $T[s[a+1]..|T|]$ ,  $\dots$ ,  $T[s[b]..|T|]$  is  $lcp(a+1, b)$ .

<sup>4</sup> A suffix array is more compact and amenable to store in secondary memory.

We can build the suffix array over a set of strings  $\mathcal{U} = \{T_1, T_2, \dots, T_\ell\}$  as in [5]. First we create a new string  $T'' = T_1\%T_2\%\dots T_\ell$  where  $\%$  is a special symbol which is smaller than any other character in  $\Sigma_{\text{DNA}}$ . Then we compute the *lcp* array using the technique presented in [10] in  $O(|T''|)$  time. By a simple trick we can use only one special character  $\%$  instead of  $\ell$  symbols, that is,  $\%$  does not match itself. We also compute the *ids* array such that  $ids[k] = i$  if  $T''[k]T''[k+1]\dots\%$  is originally a suffix of  $T_i$  (of course,  $\%$  is the symbol that appears first after  $T''[k]$  in  $T''$ ). The *ids* array can be calculated in  $O(|T''|)$  time and space. Fig. 2 is an example of the suffix array over a set of strings. Note that in this example we discarded all the suffixes which begin with  $\%$ .

$T_1=acac, T_2=aac, T_3=caac$

$T'' = \overset{1}{a} \overset{2}{c} \overset{3}{a} \overset{4}{c} \overset{5}{\%} \overset{6}{a} \overset{7}{a} \overset{8}{c} \overset{9}{\%} \overset{10}{c} \overset{11}{a} \overset{12}{a} \overset{13}{c} \overset{14}{\%}$

<i>i</i>	1	2	3	4	5	6	7	8	9	10	11
<i>ids</i>	3	2	3	1	2	1	3	1	2	3	1
<i>s</i>	11	6	12	3	7	1	13	4	8	10	2
<i>lcp</i>	-	3	1	2	2	2	0	1	1	1	2
suffix	a c %	a a c %	a c %	a c %	a c %	a c %	c %	c %	c %	c a a c %	c a c %

Fig. 2. A suffix array over strings  $T_1 = acac, T_2 = aac,$  and  $T_3 = caac$

Given a suffix array over a set of strings, a set of positive integers  $\mathcal{D} = \{d_1, d_2, \dots, d_\ell\}$ , and a positive integer  $k$ , a *candidate range* is a range  $(a, b)$  which contains  $k$  distinct values in the set  $\{ids[a], ids[a+1], \dots, ids[b]\}$  and each value  $i$  appears at least  $d_i$  times in the set. A *critical range* is a candidate range that does not properly contain other candidate ranges. Based on these two definitions, it is straightforward to draw the following lemma.

**Lemma 1.** *The answer for the generalised longest common repeat problem is  $T''[s[a']..T''[s[a' + lcp(a' + 1, b')]]$  such that  $(a', b')$  is a critical range and  $lcp(a' + 1, b')$  is the greatest among all critical ranges.*

*Proof.* Fact 2 and the definition of the candidate range implies that if  $(a, b)$  is a candidate range, then  $T''[s[a]..T''[s[b] + lcp(a + 1, b)]]$  is a common string of at least  $k$  strings in  $\mathcal{U}$ . Fact 1 says that we should narrow the range which contains the  $k$  different values in order to find the longest substring, which will correspond to the critical range. Among the critical ranges  $(a', b')$ , we select that one whose  $lcp(a' + 1, b')$  is the greatest as the answer to the problem.

### 3 Algorithm

The algorithm's framework for the generalised longest common repeat problem is as follows.

- **Step 1:** Create a new string  $T'_i$  for each  $1 \leq i \leq \ell$  to consider inverted, mirror and everted repeats, and another string  $T''$  which is the concatenation of all  $T'_i$ 's.
- **Step 2:** Construct the suffix array of  $T''$ .
- **Step 3:** Find the critical ranges.
- **Step 4:** Find the longest common substring for each critical range.

Steps 1,2 and 4 are easy to compute. Step 3 is the key step and needs a bit more or care. Following, we show each step in more detail and show how to use Kim et al.'s algorithm [12] to find the answer for the generalised longest common repeat problem.

**Step 1:** We first modify each string in  $\mathcal{U}$  to consider inverted, mirror and everted repeats. For each  $i = 1, 2, \dots, \ell$ , we create a new string  $T'_i = T_i\% \overleftarrow{T}_i\% \overleftarrow{\overleftarrow{T}}_i$ . And we create a string  $T'' = T'_1\%T'_2\% \dots \%T'_\ell\%$ .

**Step 2:** We build the suffix array of  $T''$ . The construction of the suffix array takes  $O(|T''|)$  time and space [9,11,13]. We also compute  $lcp$  and  $ids$  arrays in  $O(|T''|)$  time and space. Note that the suffixes of  $T_i$ ,  $\overleftarrow{T}_i$  and  $\overleftarrow{\overleftarrow{T}}_i$  have the same value  $i$  in the  $ids$  array.

**Step 3:** We will present an uncomplicated explanation on how to find the critical ranges. For a more exhaustively detailed account, interested readers are directed to [12].

We maintain a range  $(a, b)$  during this step. At first  $a = 1$  and  $b = 0$ . We maintain  $\ell$  counters  $c_1, c_2, \dots, c_\ell$  (initially  $c_1 = c_2 = \dots = c_\ell = 0$ ) and a counter  $h$  which contains the number of  $c_i$ 's ( $1 \leq i \leq \ell$ ) that are  $\geq d_i$ . Initially  $h = 0$ .

We now define the following two sub-steps: *expanding* and *shrinking*. In the expanding sub-step, we find a candidate range. We expand the range from  $(a, b)$  to  $(a, b + 1)$ . We check  $ids[b]$  and set  $c_{ids[b]} = c_{ids[b]} + 1$ . If  $c_{ids[b]} = d_i$ , then  $h = h + 1$ . We stop the expanding sub-step if  $h = k$ . Now  $(a, b)$  is a candidate range and we move to the shrinking sub-step. To speed up, we use the following idea, if  $lcp[b] = 0$ , then we reset all counters  $c_1, c_2, \dots, c_\ell$  and  $h$  to 0. We then start expanding sub-step again with  $a = b$  and  $b = b - 1$ .

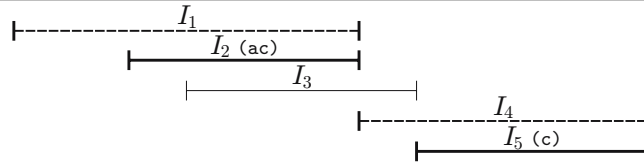
In the shrinking sub-step, we find a critical range from the candidate range  $(a, b)$  found in previous sub-step. We start by shrinking the candidate range downwards. First, we set  $c_a = c_a - 1$  and  $a = a + 1$ . If  $c_a < d_a$ , then  $h = h - 1$ . If  $h < k$ , then  $(a - 1, b)$  is a critical range. We report it and go back to the expanding sub-step with  $a$  and  $b$ . All these steps run in  $O(|T''|)$  time and space.

**Step 4:** For each critical range  $(a', b')$  found in Step 3, we use Bender and Farach-Colton's technique [2] to compute  $lcp(a' + 1, b')$ . After  $O(|T''|)$  time and

$$T_1=acac, T_2=aac, T_3=caac \quad \mathcal{D}=\{2,1,1\}$$

$$T'' = \overset{1}{a} \overset{2}{c} \overset{3}{a} \overset{4}{c} \overset{5}{\%} \overset{6}{a} \overset{7}{a} \overset{8}{c} \overset{9}{\%} \overset{10}{c} \overset{11}{a} \overset{12}{a} \overset{13}{c} \overset{14}{\%}$$

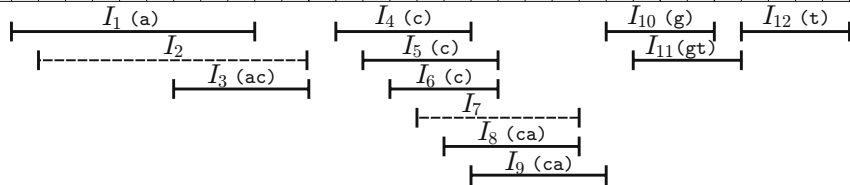
<i>i</i>	1	2	3	4	5	6	7	8	9	10	11	
<i>ids</i>	3	2	3	1	2	1	3	1	2	3	1	
<i>s</i>	11	6	12	3	7	1	13	4	8	10	2	
<i>lcp</i>	-	3	1	2	2	2	0	1	1	1	2	
suffix	a a c %	a a c %	a c %	a c %	a c %	a c %	a c %	c %	c %	c %	c a a c %	c a c %



$$T'_1=acac\%caca\%gtgt, T'_2=aac\%caa\%ggt, T'_3=caac\%caac\%ggtg$$

$$T'' = \overset{1}{a} \overset{2}{c} \overset{3}{a} \overset{4}{c} \overset{5}{\%} \overset{6}{c} \overset{7}{a} \overset{8}{c} \overset{9}{a} \overset{10}{\%} \overset{11}{g} \overset{12}{t} \overset{13}{g} \overset{14}{t} \overset{15}{\%} \overset{16}{a} \overset{17}{a} \overset{18}{c} \overset{19}{\%} \overset{20}{c} \overset{21}{a} \overset{22}{a} \overset{23}{\%} \overset{24}{g} \overset{25}{t} \overset{26}{\%} \overset{27}{c} \overset{28}{a} \overset{29}{a} \overset{30}{c} \overset{31}{\%} \overset{32}{c} \overset{33}{a} \overset{34}{a} \overset{35}{c} \overset{36}{\%} \overset{37}{g} \overset{38}{t} \overset{39}{t} \overset{40}{g} \overset{41}{\%}$$

<i>i</i>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	
<i>ids</i>	1	2	2	2	3	3	2	3	1	3	1	1	2	3	1	3	1	2	3	3	1	1	3	1	1	2	3	1	2	3	1	2	3	
<i>su</i>	9	22	21	16	29	34	17	30	3	35	7	1	18	31	4	36	8	20	28	33	2	6	41	13	11	24	38	14	26	40	12	25	39	
<i>lcp</i>	-	1	1	2	3	3	1	2	2	2	2	3	0	1	1	1	1	2	3	4	2	3	0	1	2	2	3	0	1	1	2	1	2	
suffix	a %	a %	a %	a a c %	a a c %	a a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %	a c %



**Fig. 3.** An example of the generalised longest common repeat problem. The table at the top only considers direct repeats. The table at the bottom considers direct, inverted, mirror and everted repeats. The intervals depicts dashed lines for candidate ranges and bold lines for critical ranges.

space preprocessing, each query takes  $O(1)$  time. Since it is easy to show that the number of critical ranges is  $O(|T''|)$ , the time complexity of Step 4 is  $O(|T''|)$ .

**Theorem 1.** *The generalised longest common repeat problem can be solved in  $O(\sum_{i=1}^{\ell} |T_i|)$  time and space.*

*Proof.* We showed that all the steps run in  $O(\sum_{i=1}^{\ell} |T'_i|)$  time and space. And  $|T'_i| = O(|T_i|)$  for  $1 \leq i \leq \ell$ .

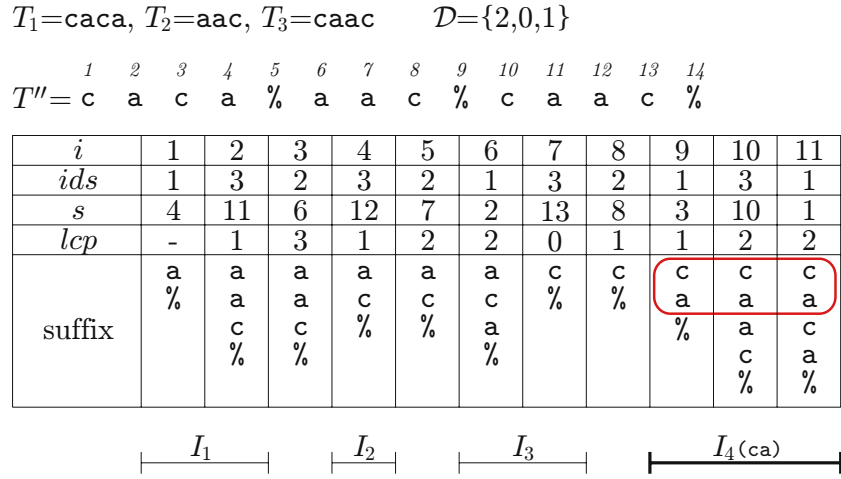
Fig. 3(top) is an example. We have three strings  $T_1 = \text{acac}$ ,  $T_2 = \text{aac}$ , and  $T_3 = \text{caac}$ ,  $k = 2$ , and  $\mathcal{D} = \{2, 1, 1\}$ . For simplicity, we do not consider the inverted repeats here. We first build the suffix array, then we find the critical ranges. At first we find a candidate range  $I_1 = (1, 6)$ . Then we shrink this range to find a critical range  $I_2 = (3, 6)$ . Here  $\text{lcp}(4, 6) = 2$ , therefore this range shares a common prefix  $\text{ac}$ . Then we try to find the next candidate range from position 5, but we meet position 7 where  $\text{lcp}[7] = 0$ . So we discard  $I_3 = (5, 7)$  and move to position 7 to find a new candidate range. From position 7, we find candidate range  $I_4 = (7, 11)$ . Then we shrink to find critical range  $I_5 = (8, 11)$ .  $\text{lcp}(8, 11) = 2$ , meaning that they share a common prefix  $\text{c}$ . Since  $\text{ac}$  is longer than  $\text{c}$ , the answer is  $\text{ac}$ .

Fig. 3(bottom) consider the previous example but taking into consideration all type of repeats. Once the suffix array is built, we proceed to find the critical ranges. At first we find a candidate range  $I_1 = (1, 9)$ . It is also a critical range and  $\text{lcp}(2, 9) = 1$ . Hence the suffixes in  $I_1$  shares a common prefix  $\text{a}$ . Then we try to find the next candidate range from position 2. We find a candidate range  $I_2 = (2, 11)$ . Next, we shrink it to find a critical range  $I_3 = (7, 11)$ .  $\text{lcp}(8, 11) = 2$  and it shares a common prefix  $\text{ac}$ , which is longer than  $\text{a}$  we found before. We begin again from position 9, but we meet position 13 where  $\text{lcp}[13] = 0$  before finding a candidate range. So we begin again from position 13. We find a candidate range  $I_4 = (13, 17)$  which is also a critical range.  $\text{lcp}(14, 17) = 1$  and the common prefix is  $\text{c}$ . Next we find a candidate range  $I_5 = (14, 18)$  which contains a critical range  $I_6 = (15, 18)$ , whose common prefix is  $\text{c}$  again. Then we find a candidate range  $I_7 = (16, 21)$  containing a critical range  $I_8 = (17, 21)$ . Its common prefix is  $\text{ca}$ . We find another critical range  $I_9 = (18, 22)$  and again we find  $\text{ca}$ . Since  $\text{lcp}[23] = 0$ , we start the search for the critical range from here. We find two critical ranges  $I_{10} = (23, 26)$  and  $I_{11} = (24, 27)$ . Their common prefix is  $\text{g}$  and  $\text{gt}$ , respectively. From position 28, we have one critical range  $I_{12} = (28, 31)$ . We find  $\text{t}$  from here. After finding all the critical ranges, we find seven substrings  $\text{a}$ ,  $\text{c}$ ,  $\text{g}$ ,  $\text{t}$ ,  $\text{ac}$ ,  $\text{ca}$ , and  $\text{gt}$ . In this case, the answer is  $\text{ac}$ ,  $\text{ca}$ , and  $\text{gt}$ .

## 4 An Application

We extend our algorithm for the generalised longest common repeat problem, finding a longest *feature* from a set. A feature of a subset  $\mathcal{U}' \subset \mathcal{U}$  is a common string which appears only in  $\mathcal{U}'$  but not in  $\mathcal{U} - \mathcal{U}'$ . More formally, this problem can be defined as follows.

*Problem 2.* Given a set of strings  $\mathcal{U} = \{T_1, T_2, \dots, T_\ell\}$ , and a set of non-negative integers  $\mathcal{D} = \{d_1, d_2, \dots, d_\ell\}$ , THE GENERALISED LONGEST FEATURE PROBLEM is to find the longest string  $w$  which satisfies two conditions: (a) There is a subset  $\mathcal{U}'$  of  $\mathcal{U}$  such that  $w$  appears at least  $d_i$  times in every string  $T_i$  in  $\mathcal{U}'$  if  $d_i > 0$ , (b)  $|\mathcal{U}'| = k$ , and (c)  $w$  should not appear in  $T_i$  if  $d_i = 0$ .



**Fig. 4.** An example for the generalised longest feature problem. Bold lines where used to indicate critical ranges.

To find the longest feature, we modify our algorithm for the generalised longest repeat problem. First, we partition the suffix array into intervals so that each partition would contain only suffixes of strings in  $\mathcal{U}'$ . Then we find the candidates of the longest feature. We find critical ranges using the algorithm in Section 3. Finally, we check if the candidates are valid or not. A candidate is not valid if it appears in some string  $T_i$  where  $d_i = 0$ . For an interval  $[a, b]$  and a critical range  $[a', b']$  where  $a \leq a'$  and  $b \geq b'$ , we compute  $lcp(a, b')$  and  $lcp(a' + 1, b + 1)$ . If either of the two equals to  $lcp(a' + 1, b')$ , it means that it appears in some string  $T_i$  where  $d_i = 0$ .

Fig. 4 is an example. We have three strings,  $T_1 = \text{caca}$ ,  $T_2 = \text{aac}$ , and  $T_3 = \text{caac}$ . And  $\mathcal{D} = \{2, 0, 1\}$ , meaning that a feature should appear at least twice in  $T_1$ , and at least once in  $T_3$ , but it should not appear in  $T_2$ . We partition the suffix array into four intervals,  $I_1 = [1, 2]$ ,  $I_2 = [4, 4]$ ,  $I_3 = [6, 7]$ , and  $I_4 = [9, 11]$ . There is no critical range in  $I_1$ ,  $I_2$ , and  $I_3$ .  $I_4$  contains a critical range  $[9, 11]$  which represents a candidate  $\text{ca}$ . Since  $lcp(9, 11) = 1$  ('c') and  $lcp(10, 11) = 2$ , we say that  $\text{ca}$  never appears in  $T_2$ .

**Theorem 2.** *The generalised longest common repeat problem can be solved in  $O(\sum_{i=1}^{\ell} |T_i|)$  time and space.*

*Proof.* We showed that all the steps run in  $O(\sum_{i=1}^{\ell} |T'_i|)$  time and space. And  $|T'_i| = O(|T_i|)$  for  $1 \leq i \leq \ell$ .

## 5 Conclusions

We have defined the generalised longest common repeat problem and presented a linear time algorithm for the problem, allowing direct and inverted repeats.

A remaining work is to devise a more space-efficient algorithm for the problem. Another possibility is to incorporate the detection of degenerated repeats (which are called approximate repeats in the stringology literature).

## References

1. M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. The enhanced suffix array and its application to genome analysis. In *Proceedings of the 2nd Workshop on Algorithms in Bioinformatics (WABI 2002)*, pages 449–463, 2002.
2. M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proceedings of the Fourth Latin American Symposium*, pages 88–94, 2000.
3. J. Beckman and M. Soller. Toward a unified approach to genetic mapping of eukaryotes based on sequence tagged microsatellite sites. *Biotechnology*, pages 8:930–932, 1990.
4. C. T. Caskey, *et al.* An unstable triplet repeat in a gene related to Myotonic Dystrophy, *Science*, pages 255:1256–1258, 1992.
5. S. Dori and G. M. Landau. Construction of aho-corasick automaton in linear time for integer alphabets. In *Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching (CPM 2005)*, page to appear, 2005.
6. M. Farach-Colton, P. Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *Journal of the ACM*, 47(6):987–1011, 2000.
7. K. Inman and N. Rudin. An introduction to forensic DNA analysis. CRC press, Boca Raton, Florida, 1997.
8. A. Jeffreys, D. Monckton, K. Tamaki, D. Neil, J. Armour, A. MacLeod, A. Collick, M. Allen, and M. Jobling. Minisatellite variant repeat mapping: application to DNA typing and mutation analysis. In *DNA Fingerprinting: State of the Science*, pages 125–139, Basel, 1993.
9. J. Kärkkäinen and P. Sanders. Simpler linear work suffix array construction. In *Proceedings of the 13th International Colloquium on Automata, Languages and Programming (ICALP 2003)*, pages 943–945, 2003.
10. T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching (CPM 2001)*, pages 181–192, 2001.
11. D. K. Kim, J. S. Sim, H. Park, and K. Park. Linear-time construction of suffix arrays. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003)*, pages 186–199, 2003.
12. S.-R. Kim, I. Lee, and K. Park. A fast algorithm for the generalised  $k$ -keyword proximity problem given keyword offsets. *Information Processing Letters*, 91(3):115–120, 2004.
13. P. Ko and S. Aluru. Space-efficient linear time construction of suffix arrays. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003)*, pages 200–210, 2003.
14. G. M. Landau and J. P. Schmidt. An algorithm for approximate tandem repeats. In *Proceedings of the Fourth Combinatorial Pattern Matching*, pages 120–133, 1993.
15. G. M. Landau, J. P. Schmidt and D. Sokol. An algorithm for approximate tandem repeats. In *Journal of Computational Biology*, pages 8(1): 1–18 , 2001.
16. I. Lee, C. S. Iliopoulos, and K. Park. Linear time algorithm for the longest common repeat problem. In *Proceedings of the 11th String Processing and Information Retrieval (SPIRE 2004)*, pages 10–17, 2004.

17. E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, April 1976.
18. J. P. Schmidt. All highest scoring paths in weighted grid graphs and its application to finding all approximate repeats in strings. *SIAM Journal on Computing*, 27(4):972–992, 1998.
19. R. H. Singer. Triplet-repeat transcripts: A role for RNA in disease. *Science*, 280(5364):696–697, 1998.
20. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995.
21. K. J. Woo, K. Sang-Ho, and C. Jae-Kwan. Association of the dopamine transporter gene with Parkinsons disease in Korean patients *Journal of Korean Medical Science*, 15(4), 2000.