Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Longest common substrings with k mismatches

Tomas Flouri [a], Emanuele Giaquinta [b,*], Kassian Kobert [a], Esko Ukkonen [c]

[a] Heidelberg Institute for Theoretical Studies, Germany
[b] Department of Computer Science, Aalto University, Finland
[c] Department of Computer Science, University of Helsinki, Finland

ARTICLE INFO

Article history:
Received 8 September 2014
Received in revised form 16 March 2015
Accepted 16 March 2015
Available online 20 March 2015
Communicated by Ł. Kowalik

Keywords:
Combinatorial problems
String algorithms
Hamming distance
Longest common substring

ABSTRACT

The longest common substring with k-mismatches problem is to find, given two strings S_1 and S_2, a longest substring A_1 of S_1 and A_2 of S_2 such that the Hamming distance between A_1 and A_2 is $\leq k$. We introduce a practical $O(nm)$ time and $O(1)$ space solution for this problem, where n and m are the lengths of S_1 and S_2, respectively. This algorithm can also be used to compute the matching statistics with k-mismatches of S_1 and S_2 in $O(nm)$ time and $O(m)$ space. Moreover, we also present a theoretical solution for the $k = 1$ case which runs in $O(n \log m)$ time, assuming $m \leq n$, and uses $O(m)$ space, improving over the existing $O(nm)$ time and $O(m)$ space bound of Babenko and Starikovskaya [1].

1. Introduction

In this paper we study the longest common substring (or *factor*) with k-mismatches problem (k-LCF for short[1]) which consists in finding the longest common substring of two strings S_1 and S_2, while allowing for at most k mismatches, i.e., the Hamming distance between the two substrings is $\leq k$. This problem is a generalization of the Longest Common Substring problem [2–4] and is similar to the *threshold all-against-all* problem defined by Gusfield [2] and to the *local alignment* problem of biological sequence analysis. In the threshold all-against-all problem the goal is to find all the pairs of substrings of S_1 and S_2 such that the corresponding edit distance is less than a given number d. The difference in the k-LCF problem is that the distance used is the Hamming distance rather than the edit distance, and that we are interested in the pairs of substrings of maximal length only. In the local alignment problem, which can be solved in $O(|S_1| \cdot |S_2|)$ time using the Smith–Waterman algorithm [5], the goal is to compute a pair of substrings of S_1 and S_2 such that the corresponding similarity, according to a suitable scoring function, is maximum over all the pairs of substrings. In particular, if the scoring function is such that the score of a match is 1, the score of a mismatch is 0 and gaps are not allowed, a solution of the local alignment problem is comparable to one of the k-LCF problem, with the difference that there is no bound on the number of mismatches.

Babenko and Starikovskaya [1] studied the case of 1 mismatch only and presented an algorithm for the 1-LCF problem which runs in $O(|S_1| \cdot |S_2|)$ time. A closely related problem is the one of computing the matching statistics with k mismatches. The matching statistics, introduced by Chang and Lawler [6] for the approximate string matching problem, is an array ms of $|S_2|$ integers such that $ms[i]$ is the length of the longest substring of S_2 that starts at position i and matches exactly some substring of S_1, for $i = 0, \ldots, |S_2| - 1$. A natural generalization is obtained by allowing the matching to be approximate, with respect to the Hamming distance. Recently, Leimeister and Morgenstern [7] presented a greedy heuristic for the computation of the matching statistics with k mismatches, which runs

* Corresponding author.
E-mail address: emanuele.giaquinta@aalto.fi (E. Giaquinta).

[1] We use the k-LCF abbreviation as LCS usually refers to the *Longest Common Subsequence* problem.

http://dx.doi.org/10.1016/j.ipl.2015.03.006

in $O(|S_1| \cdot k \cdot z)$ time, where z is the maximum number of occurrences in S_2 of a string of maximal length which occurs in both S_1 and S_2.

In this paper we present two novel contributions. Our first result is an efficient algorithm for the k-LCF problem which runs in time $O(|S_1| \cdot |S_2|)$ and only requires a constant amount of space. This algorithm can also be used to compute the matching statistics with k mismatches with no overhead in the time complexity, i.e., in $O(|S_1| \cdot |S_2|)$ time, and using $O(|S_2|)$ space. Our second result is an algorithm for the 1-LCF problem, i.e., for the $k = 1$ case. We show how to solve this instance in a more time efficient manner by using results from Crochemore et al. [8] for finding the longest generalized repeat(s) with one block of k adjacent don't care symbols. Assuming $|S_2| \leq |S_1|$, our algorithm takes time $O(|S_1| \log |S_2|)$, improving over the previous bound of $O(|S_1| \cdot |S_2|)$.

2. Basic definitions

Let Σ be a finite alphabet of symbols and let Σ^* be the set of strings over Σ. Given a string $S \in \Sigma^*$, we denote by $|S|$ the length of S and by $S[i]$ the i-th symbol of S, for $0 \leq i < |S|$. Given two strings S and S', S' is a substring of S if there are indices $0 \leq i \leq j < |S|$ such that $S' = S[i] \ldots S[j]$. If $i = 0$ ($j = |S| - 1$) then S' is a prefix (suffix) of S. We denote by $S[i..j]$ the substring of S starting at position i and ending at position j. For $i > j$ we obtain the empty string ε. Finally, we denote by $S^r = S[|S|-1]S[|S|-2] \ldots S[0]$ the reverse of the string S.

The suffix tree $\mathcal{T}(S)$ of a string S is a rooted directed tree with $|S'|$ leaves and edge labels over $(\Sigma \cup \{\$\})^* \setminus \{\varepsilon\}$, where $\$ \notin \Sigma$ and $S' = S\$$. Each internal node has at least two children and is such that the edge labels of the children have different first symbols. For each leaf i, the concatenation of the edge labels on the path from the root to leaf i is equal to $S'[i..|S'|-1]$. Assuming a constant size alphabet, the suffix tree can be built in $O(|S|)$ time [2]. For any node u in $\mathcal{T}(S)$, *depth*(u) denotes the length of the string labeling the path from the root to u. For any pair of nodes u, v in $\mathcal{T}(S)$, *LCA*(u, v) denotes the lowest common ancestor of u and v, i.e., the deepest node in $\mathcal{T}(S)$ that is ancestor of both u and v. The suffix tree can be preprocessed in $O(|S|)$ time so as to answer LCA queries in constant time [9]. We denote by $\mathcal{B}(S)$ the binary suffix tree obtained by replacing each node u in $\mathcal{T}(S)$ with out-degree at least 2 with a binary tree with $d-1$ internal nodes (whose *depth* values are equal to *depth*(u)) and $d-2$ internal edges, where the d leaves are the d children of u. The binary suffix tree can be built in $O(|S|)$ time [8]. The generalized suffix tree $\mathcal{T}(S_1, S_2)$ of two strings S_1 and S_2 is the suffix tree built over $S' = S_1\$_1S_2\$_2$, where $\$_1, \$_2 \notin \Sigma$, such that the leaves are numbered with a pair (s-index) and for each leaf (j, l) the concatenation of the edge labels on the path from the root to the leaf is equal to $S_j[l..|S_j|-1]\$_j$. The index of a leaf (j, l) is the starting position of $S_j[l..|S_j|-1]\$_j$ in $S_1\$_1S_2\$_2$. We use the notation $\mathcal{B}(S_1, S_2)$ to denote the binary generalized suffix tree of S_1 and S_2.

3. The longest common substring with *k* mismatches problem

Let S_1 and S_2 be two strings with $n = |S_1|$, $m = |S_2|$. W.l.o.g. we assume that $n \geq m$. Given an integer k, let $\phi(i, j)$ be the length of the longest substring of S_1 and S_2 ending at position i and j, respectively, such that the two substrings have Hamming distance at most k. Formally, $\phi(i, j)$ is equal to the largest integer $l \leq \min(i, j) + 1$ such that

$$|\{0 \leq h \leq l-1 \mid S_1[i-h] \neq S_2[j-h]\}| \leq k,$$

for $0 \leq i < n, 0 \leq j < m$. The *longest common substring with k-mismatches* problem consists in, given two strings S_1 and S_2 and an integer k, finding the length of the longest substrings of S_1 and S_2 with Hamming distance at most k, i.e., $\max_{i,j} \phi(i, j)$.

4. A practical algorithm for arbitrary *k*

In this section we present a practical algorithm for the k-LCF problem. By definition, $\phi(i, j)$ is also the length of the longest suffixes of $S_1[0..i]$ and $S_2[0..j]$ with Hamming distance at most k. Our algorithm computes all the values $\phi(i, j)$ based on this alternative formulation. The idea is to iterate over the ϕ matrix diagonal-wise and compute, for a fixed $(i, j) \in \{(0,0), (0,1), \ldots, (0, m-1)\} \cup \{(1,0), (2,0), \ldots, (n-1, 0)\}$, the values $\phi(i+p, j+p)$, for $0 \leq p < \min(n-i, m-j)$, i.e., the diagonal starting at (i, j), in $O(m)$ time. Let Q be an (empty) queue data structure and $s = 0$, for a given pair (i, j). The algorithm iterates over p maintaining the invariant that $p - s$ is the length of the longest common suffix of $S_1[i..i+p-1]$ and $S_2[j..j+p-1]$ up to k-mismatches, i.e., $p - s = \phi(i+p-1, j+p-1)$, and that Q contains exactly the positions in S_1 of the mismatches between $S_1[i+s..i+p-1]$ and $S_2[j+s..j+p-1]$ with the order of elements in the queue matching their natural order.

At the beginning the invariant holds since Q is empty, $p - s = 0$ and $S_1[i+s..i+p-1] = S_2[j+s..j+p-1] = \varepsilon$. Suppose that the invariant holds up to position p. If $S_1[i+p] = S_2[j+p]$ then the invariant trivially holds also for $p+1$ with $s' = s$ and $Q' = Q$. Otherwise, we have a mismatch between $S_1[i+p]$ and $S_2[j+p]$. If $|Q| < k$, then the invariant also holds for $p+1$ with $s' = s$ and Q' equal to Q after an ENQUEUE(Q, p) operation. Instead, if $|Q| = k$, the pair of suffixes $S_1[i+r..i+p]$ and $S_2[j+r..j+p]$, for $r = s, \ldots, \min Q$, match with $k+1$ mismatches and $r = \min Q + 1$ is the minimum position for which the corresponding suffixes match with k mismatches. Hence, in this case the invariant also holds for $p+1$ with $s' = \min Q + 1$ and Q' equal to Q after a DEQUEUE operation followed by an ENQUEUE(Q, p) operation.

The algorithm maintains the largest length found up to the current iteration and the starting positions of the corresponding substrings in S_1 and S_2, such that the position in S_1 is minimal, in three integers ℓ, r_1, and r_2. Each time $p - s > \ell$ it updates their values accordingly. The code of the algorithm is shown in Fig. 1. The time complexity of one iteration of the algorithm is $O(1)$ if the queue operations take constant time, which yields $O(m)$ time for a