Information Processing Letters 112 (2012) 562–565

Contents lists available at SciVerse ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Doubly-Constrained LCS and Hybrid-Constrained LCS problems revisited

Effat Farhana [a,b], M. Sohel Rahman [a,*]

[a] AℓEDA Group, Department of CSE, BUET, Dhaka-1000, Bangladesh
[b] Department of CSE, AUST, Dhaka-1208, Bangladesh

ARTICLE INFO

Article history:
Received 23 September 2011
Received in revised form 14 April 2012
Accepted 18 April 2012
Available online 21 April 2012
Communicated by J. Torán

Keywords:
Finite automata
Longest common subsequence
Algorithms
Combinatorial problems

ABSTRACT

We revisit two recently studied variants of the classic Longest Common Subsequence (LCS) problem, namely, the Doubly-Constrained LCS (DC-LCS) and Hybrid-Constrained LCS (HC-LCS) problems. We present finite automata based algorithms for both problems.

1. Introduction

A subsequence of a given sequence s is obtained by deleting zero or more symbols from s. Given two sequences, the longest common subsequence (LCS) problem is to find a common subsequence whose length is the longest. The classic dynamic programming algorithm to compute an LCS of two input strings was invented by Wagner and Fischer [13]. A constrained variant of the longest common subsequence (CLCS) problem, was first proposed by Tsai [12], where the computed LCS must contain a specific constraint (input) string as a subsequence. Subsequently, this problem was addressed in [4,9,6,10]. Among other interesting constrained variants of LCS, *repetition-free LCS* [1], *exemplar LCS* [2], etc., may be cited.

In this paper, we study two new variants of the CLCS problem, namely, the "Doubly-Constrained LCS (DC-LCS)" and the "Hybrid-Constrained LCS (HC-LCS)" problems. These two problems were very recently introduced and studied in [3] and [5], respectively. The problems are formally defined below.

Problem 1 *(DC-LCS).* Given two input strings s_1, s_2, a set of constraint patterns C_s and an occurrence constraint function $C_o : \Sigma \to N$, assigning an upper bound on the number of occurrences of each symbol $\sigma \in \Sigma$, the goal of DC-LCS is to find an LCS s of s_1, s_2 such that s contains at most $C_o(\sigma)$ occurrences of each symbol $\sigma \in \Sigma$ and contains each pattern in C_s as a subsequence.

Problem 2 *(HC-LCS).* Given two input strings s_1, s_2, two constrained patterns P and Q, the goal of HC-LCS is to compute an LCS s of s_1 and s_2 such that s is a supersequence of P but not of Q.

DC-LCS is NP-hard for arbitrary number of constraint strings. Bonizzoni et al. [3] presented a fixed-parameter algorithm where the parameter k is the length of the solution. Their algorithm runs in $k^k T(k, |s_1|, |s_2|)$ time, where $T(k, |s_1|, |s_2|) = (|s_1| \log |s_1| 2^{O(k)}) + O(|s_1||s_2||s_c| \times 2^{O(k)} \log |\widetilde{\Sigma}|)$. Here, s_c is one of the constraint patterns in C_s and $\widetilde{\Sigma}$ is the set containing the pairs (σ, i) for each $\sigma \in \Sigma$ and $i \in \{1, \ldots, C_o(\sigma)\}$. On the other hand, for HC-LCS, two algorithms were presented in [5], of time complexity $O(n^2|P||Q|)$ and $O(|P||Q|r \log\log n + n \log n)$, respectively, where $n = \max(|s_1|, |s_2|)$ and r is the total number of matches between s_1, s_2. Note that, in the worst

* Corresponding author.
E-mail addresses: effat34@gmail.com (E. Farhana), msrahman@cse.buet.ac.bd (M.S. Rahman).

0020-0190/$ – see front matter
http://dx.doi.org/10.1016/j.ipl.2012.04.007

case, $r = O(n^2)$, hence the latter algorithm is slightly worse than the former in the worst case. Notably, finite alphabet was assumed in [5]. In this paper, we devise finite automata based efficient algorithms for both DC-LCS and HC-LCS problems.

2. Preliminaries

To formally describe our algorithms the following definitions are necessary.

Definition 1 *(DFA).* A Deterministic Finite Automaton is represented by 5-tuple notation $A = (Q, \Sigma, \delta, q_0, F)$, where A is the name of the DFA, Q its set of states, Σ its set of input symbols, δ its transition function, q_0 its start state and F its set of accepting states.

Definition 2 *(DASG).* A Directed Acyclic Subsequence Graph (DASG) for a string s of length n is a DFA that accepts the language of all possible 2^n subsequences of s. The DFA is *partial*, that is, each state may not have a transition defined for every symbol [8].

Definition 3 *(DASG for multiple texts).* Let S be a set of strings $T_1, T_2, \dots, T_k$. We say that P is a subsequence of S if and only if there exists $i \in [1, k]$ such that P is a subsequence of T_i. DASG of S is a DFA A which accepts the language $\mathcal{L}(A) = \{w\colon\ i \in [1, k],\ w \text{ is a subsequence of } T_i\}$ [8].

Definition 4 *(Common Subsequence Automaton).* Given a set of strings, a Common Subsequence Automaton (CSA) accepts all common subsequences of the given strings. The language accepted by CSA is a subset of the language accepted by the DASG for a set of strings.

Definition 5 *(Supersequence Automaton).* A Supersequence Automaton is a finite automaton which accepts the set of all supersequences of a given string.

3. A fixed-parameter algorithm for DC-LCS

We present a fixed-parameter algorithm for the DC-LCS problem where the parameter k is the size of a solution of DC-LCS. The algorithm consists of five main stages.

Stage 1: In the first stage, we build a CSA automation which accepts all common subsequences of two input strings. This is done as follows. The DASG M_1 for the two input texts is constructed by using the online algorithm of [8]. Then common subsequence automaton (CSA) of the two strings is obtained from the DASG M_1 by considering the 'match' values computed for each state. The value of 'match' corresponds to the number of input strings that contain a given string, s as a subsequence [8]. So, we will prune all states whose 'match' value is less than two to get the desired CSA M_1'. In the worst case, $\mathcal{R} = O(n^2)$ states are generated for two input strings where $n = \max(|s_1|, |s_2|)$.

Stage 2: In the second stage, we build $|C_s|$ supersequence automata M_2^i, $1 \leqslant i \leqslant |C_s|$, for each constraint pattern in $s_c^i \in C_s$ using the algorithm in [11]. Clearly, M_2^i will accept all the strings containing the pattern s_c^i as a subsequence.

Stage 3: In the third stage, we intersect all $|C_s|$ automata M_2^i, $1 \leqslant i \leqslant |C_s|$, with M_1' using the algorithm in [11]. The resulting automaton accepts all common subsequences of s_1, s_2 including each pattern of C_s as a subsequence. We call the resulting automaton M_3.

Stage 4: We consider character constraint in the fourth stage. As the size of a solution of DC-LCS is k, each $\sigma \in \Sigma$ cannot occur more than k times. For each $\sigma \in \Sigma$, we can construct a DFA which accepts all strings having at most σ_{occ} occurrences, where $\sigma_{occ} = \min(k, C_o(\sigma))$. We intersect all these $|\Sigma|$ automata with M_3 and denote it by M_4. The automaton M_4 accepts the sequences that are accepted by M_3 but do not violate the constraint function $C_o : \Sigma \to N$.

Stage 5: In the final stage, we have to select DC-LCS of length k from M_4. This can be done easily by a modification of maximum length automata (MaxLen automata) [11]. In brief, the algorithm is a modification of the longest path algorithm for DAGs (Directed Acyclic Graph) that works in $O(E)$ time, where E is the number of edges in the input DAG. It can be easily modified to accept strings in a DAG of length k. We call the resulting automata, M_5.

3.1. Time complexity

In our algorithm, we have used the online algorithm of [8] for DASG construction and the algorithms in [11] for supersequence and intersection automata construction. For the sake of convenience, we assume that the length of each input string is n. The length of each constraint pattern may safely be assumed to be k since our solution size is bounded by k. To analyze our algorithm, we need the following result.

Lemma 1. *(See [11].) Given DFA M_1 and M_2 having $\mathcal{R}$ and n states respectively, a DFA M accepting language $\mathcal{L}(M) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$ can be constructed in $O(|\Sigma|\mathcal{R}n)$ time. M has at most $\mathcal{R}n$ states and at most $|\Sigma|\mathcal{R}n$ transitions. Moreover, if M_1 (or M_2) is acyclic, then M is also acyclic.*

We also state the following easy lemma.

Lemma 2. *Given an integer N denoting the upper bound on occurrences of letter $\sigma \in \Sigma$, a DFA M accepting all the strings containing at most N occurrences of σ can be built in $O(N)$ time and M has $O(N)$ states.*

Construction of CSA from DASG M_1 requires $O(|\Sigma| \times (\mathcal{R} + 2) + 2n)$ time [8], as the number of states of DASG is $\mathcal{R} = O(n^2)$ in the worst case. Building supersequence automaton for each M_2^i, $1 \leqslant i \leqslant |C_s|$, needs $O(|\Sigma|k)$ time.