Constrained LCS: Hardness and Approximation

Zvi Gotthilf[1], Danny Hermelin[2], and Moshe Lewenstein[1]

[1] Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel
{gotthiz,moshe}@cs.biu.ac.il
[2] Department of Computer Science, University of Haifa,
Mount Carmel, Haifa 31905, Israel
danny@cri.haifa.ac.il

Abstract. The problem of finding the longest common subsequence (LCS) of two given strings A_1 and A_2 is a well-studied problem. The constrained longest common subsequence (C-LCS) for three strings A_1, A_2 and B_1 is the longest common subsequence of A_1 and A_2 that contains B_1 as a subsequence. The fastest algorithm solving the C-LCS problem has a time complexity of $O(m_1m_2n_1)$ where m_1, m_2 and n_1 are the lengths of A_1, A_2 and B_1 respectively. In this paper we consider two general variants of the C-LCS problem. First we show that in case of two input strings and an arbitrary number of constraint strings, it is NP-hard to approximate the C-LCS problem. Moreover, it is easy to see that in case of an arbitrary number of input strings and a single constraint, the problem of finding the constrained longest common subsequence is NP-hard. Therefore, we propose a linear time approximation algorithm for this variant, our algorithm yields a $1/\sqrt{m_{min}|\Sigma|}$ approximation factor, where m_{min} is the length of the shortest input string and $|\Sigma|$ is the size of the alphabet.

1 Introduction

The problem of finding the longest common subsequence (LCS) of two given strings A_1 and A_2 is a well-studied problem, see [3,6,7,1]. The *constrained* longest common subsequence (C-LCS) for three strings A_1, A_2 and B_1 is the longest common subsequence of A_1 and A_2 that contains B_1 as a subsequence. Tsai [10] gave a dynamic programming algorithm for the problem which runs in $O(n^2m^2k)$ where m, n and k are the lengths of A_1, A_2 and B_1 respectively. Improved dynamic programming algorithms were proposed in [2,4] which run in time $O(nmk)$. Approximated results for this C-LCS variant presented in [5].

Many problems in pattern matching are solved with dynamic programming solutions. Among the most prominent of these is the LCS problem. These solutions are elegant and simple, yet usually their running times are quadratic or more, i.e. they are not effective in the case of multiple strings. It is a desirable goal to find algorithms which offer faster running times. One slight improvement, a reduction of a log factor, is the classical Four-Russians trick, see [9]. However, in general, faster algorithms have proven to be rather elusive over the years (and perhaps it is indeed impossible).

P. Ferragina and G. Landau (Eds.): CPM 2008, LNCS 5029, pp. 255–262, 2008.

The classical LCS problem has many applications in various fields. Among them applications in string comparison, pattern recognition and data compression. Another application, motivated from computational biology, is finding the commonality of two DNA molecules. Closely related, Tsai [10] gave a natural application for the C-LCS problem: in the computation of the commonality of two biological sequences it may be important to take into account a common specific structure.

1.1 Our Contribution

We propose to consider two general variants of the C-LCS problem. First, we prove that in case of two input strings and an arbitrary number of constraint strings, it is NP-hard to approximate the C-LCS problem. In addition, we obtain the first approximation algorithm for the case of many input strings and a single constraint. Our algorithm yields a $1/\sqrt{m_{min}|\Sigma|}$ approximation factor, where m_{min} is the length of the shortest input string and $|\Sigma|$ is the size of the alphabet. The running time of our algorithm is linear.

2 Preliminaries

Let $A_1 = \langle a_{1_1}, a_{1_2}, \ldots, a_{1_{m1}} \rangle$, $A_2 = \langle a_{2_1}, a_{2_2}, \ldots, a_{2_{m2}} \rangle$, ..., $A_k = \langle a_{k_1}, a_{k_2}, \ldots, a_{k_{mk}} \rangle$ and $B_1 = \langle b_{1_1}, b_{1_2}, \ldots, b_{1_{n1}} \rangle$, $B_2 = \langle b_{2_1}, b_{2_2}, \ldots, b_{2_{n2}} \rangle$, ..., $B_l = \langle b_{n_1}, b_{n_2}, \ldots, b_{1_{nl}} \rangle$ be an input of the C-LCS problem. The longest constrained subsequence (C-LCS, for short) of A_1, A_2, ..., A_k and B_1, B_2, ..., B_l is the longest common subsequence of A_1, A_2, ..., A_k that contains each of B_1, B_2, ..., B_l as a subsequence. The approximation version of the C-LCS problem is defined as follows. Let OPT_{clcs} be the optimal solution for the C-LCS problem and APP_{clcs} the result of the approximation algorithm APP such that:

- APP_{clcs} is a common subsequence of A_1, A_2, ..., A_k.
- B_1, B_2, ..., and B_l are subsequences of APP_{clcs}.

The approximation ratio of the APP algorithm will be the smallest ratio between $|APP_{clcs}|$ and $|OPT_{clcs}|$ over all possible input strings A_1, A_2, ..., A_k and B_1, B_2, ..., B_l.

Clearly, not every instance of the C-LCS problem must have a feasible solution, i.e. there is no common subsequence of all input strings that contains every constraint string as a subsequence. It can be seen in figure 1 that the left instance is an example of a non-feasible C-LCS instance, while for the right instance "*bcabcab*" is a feasible constrained common subsequence.

3 Arbitrary Number of Constraints

In this section we prove that given two input strings and an arbitrary number of constrains the problem of finding the C-LCS is NP-hard. In addition, we show that it is NP-hard to approximate C-LCS for such instances.

Fig. 1. Non feasible and feasible C-LCS instances

Theorem 1. *The C-LCS problem in case of an arbitrary number of constraints is NP-complete.*

Proof: We prove the hardness of the problem by a reduction from 3-SAT.

Given a 3-SAT instance with variables $x_1, x_2, \ldots, x_k$ and clauses $c_1, c_2, \ldots, c_l$, we construct an instance of C-LCS with two input strings and $k+l-1$ constraints.

The alphabet of A_1 and A_2 is the set of clauses $c_1, c_2, \ldots, c_l$ and a set of separators $\{s_1, s_2, \ldots, s_{k-1}\}$ separating between the variables.

We construct A_1 as follows. For each variable x_i we create a substring X_i by setting all the clauses satisfied with $x_i = true$ followed by all the clauses satisfied with $x_i = false$ (we set the clauses in a sorted order). We then set A_1 to be $X_1 s_1 X_2 s_2 \ldots s_{k-1} X_k$, the X_i substrings separated by the appropriate separators.

We similarly construct A_2. We create a substring $X_i^{'}$ by setting all the clauses satisfied with $x_i = false$ followed by all the clauses satisfied with $x_i = true$ (we set the clauses in a sorted order). We then set A_2 to be $X_1^{'} s_1 X_2^{'} s_2 \ldots s_{k-1} X_k^{'}$, the $X_i^{'}$ substrings separated by the appropriate separators.

Let $c_1, c_2, \ldots, c_l$ and $s_1, s_2, \ldots, s_{k-1}$ be the group of constraints. Note that, all of them are of length one.

See figure 2 as an example of our constriction from the following 3-SAT instance to a C-LCS instance that contains two input strings and $k+l-1$ constraints (all of length one):

$$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x_1} \vee \bar{x_2} \vee x_4) \wedge (x_2 \vee \bar{x_3} \vee \bar{x_4}) \wedge (x_1 \vee \bar{x_3} \vee x_4)$$

Lemma 1. *A 3-SAT instance can be satisfied iff there exists a C-LCS of length* $\geq k+l-1$.

Proof: For simplicity, we assume that there are no clauses that contains both x_i and $\bar{x_i}$.

($\Rightarrow$) Suppose a 3-SAT instance can be satisfied.

Let X be an assignment on the variables satisfying the 3-SAT instance. Let Y be the variables assigned *true* values of X and Z be the variables assigned

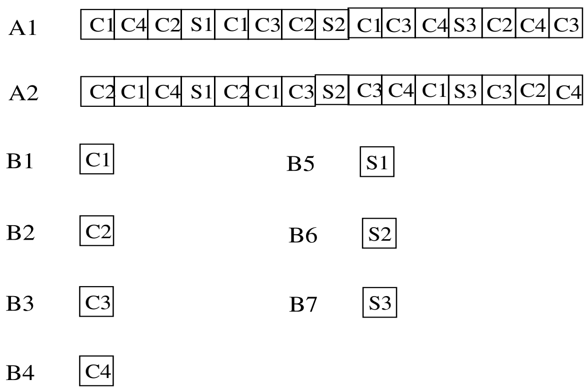

Fig. 2. Construction example

$false$ values. For each variable $x_i \in Y$, let $\{c_{i_j}, \ldots, c_{i_r}\}$ be the clauses which are satisfied by setting x_i to $true$.

We construct a valid C-LCS as follows. Add to the C-LCS the $c'_{i_j}s$ from X_i and $X_i^{'}$. Clearly they cannot cross each other as they are ordered. Likewise for $x_i^{'} \in Z$ we do the same. Moreover, we select $s_1, s_2, \ldots, s_{k-1}$. Note that, since x_i is either $true$ or $false$ we will have:

1. No internal crossings within X_i and $X_i^{'}$.
2. No crossing over the separators.

Obviously, since all clauses are satisfied (by some variable) they appear within the LCS. Since also $s_1, s_2, \ldots, s_{k-1}$ appear, all the C-LCS constraints are satisfied. Therefore, $|C - LCS| \geq l + k - 1$.

($\Leftarrow$) Note that all constraints must be satisfied. Hence, $s_1, s_2, \ldots, s_{k-1}$ appears in the C-LCS. Therefore, any clause c_i appearing in the C-LCS must be within a given X_i, $X_i^{'}$. Thus, there cannot be an inconsistency of the x_i assignments. Because the clauses $c_1, c_2, \ldots, c_l$ are constraints, they must appear in the C-LCS.

Therefore, the assignment of $x_1, x_2, \ldots, x_k$ must satisfy the 3-SAT instance, since every clause must be satisfied in the C-LCS instance. □

The following theorem derived from our reduction.

Theorem 2. *The C-LCS problem in case of an arbitrary number of constraints cannot be approximated.*

Proof: By the C-LCS definition and according to Lemma 1, any valid solution for the C-LCS must satisfy all the constraints (and must be of length $\geq k + l - 1$). Therefore, any approximation algorithm must yield an appropriate solution to the 3-SAT problem. In case that an approximation algorithm fails to find a C-LCS, we can conclude that the corresponding 3-SAT instance could not be satisfied. □

Note that, our reduction is based on a C-LCS instance in which all the constraints are of length one.

4 Single Constraint

In this section we consider the case of an arbitrary number of input strings and a single constraint. It is easy to see that the problem of finding the constrained longest common subsequence is NP-hard. Therefore, we present an approximation algorithm for this case. Our algorithm yields a $1/\sqrt{m_{min}|\Sigma|}$ approximation factor within a linear running time (while m_{min} is the length of the shortest input string). Let $A_1, A_2, \ldots, A_k$ be the input strings. Throughout this section we assume a single constraint string exists, denote it by $B = \langle b_1, b_2, \ldots, b_n \rangle$.

The following result follows from the NP-hardness of the LCS [8] and by setting $B = \epsilon$.

Observation 1. *Given an arbitrary number of input strings and a single constraint, the problem of finding the C-LCS of such instances is NP-hard.*

4.1 Approximation Algorithm

Now we present a linear time approximation algorithm. First we give some useful notations that will be used throughout this subsection.

Let $A_i = \langle A_{i_1}, A_{i_2}, \ldots, A_{i_{m_i}} \rangle$ be an input string of length m_i. Denote with $A_i[s, e]$ the substring of A_i that starts at location s and ends at location e. Denote by $start(A_i, j)$ the leftmost location in A_i such that $b_1, b_2, \ldots, b_j$ is a subsequence of $A_i[1, start(A_i, j)]$. Symmetrically, denote by $end(A_i, j)$ the rightmost location in A_i such that $b_j, b_{j+1}, \ldots, b_n$ is a subsequence of $A_i[end(A_i, j), m_i]$. See Figure 3 as an example of $start(A_i, j)$ and $end(A_i, j)$. For the simplicity of the analysis assume that $start(A_i, 0) + 1 = A_{i_1}$ and $end(A_i, n + 1) - 1 = A_{i_{m_i}}$.

Let OPT_{clcs} be an optimal C-LCS solution. By definition, B must be a subsequence of OPT_{clcs} and a subsequence of every input string A_i $(1 \leq i \leq k)$.

Choose an arbitrary embedding of B over OPT_{clcs} (as a subsequence) and denote with $p_1, p_2, \ldots, p_n$ the positions of $b_1, b_2, \ldots, b_n$ in OPT_{clcs}. For simplicity assume $p_0 + 1$ and $p_{n+1} - 1$ are the positions of the first and the last characters of OPT_{clcs} respectively. Note that there may be many possible embeddings of B over OPT_{clcs}.

The following lemma and corollaries are instrumental in achieving the desirable approximation ratio.

Lemma 2. *Let $B = \langle b_1, b_2, \ldots, b_n \rangle$ be the constraint string and OPT_{clcs} be an optimal C-LCS, then for any assignment of B over OPT_{clcs} and for every $0 \leq i \leq n$ the following statement holds:*

$$|LCS(A_1[start(A_1, i)+1, end(A_1, i+1)-1], A_2[start(A_2, i)+1, end(A_2, i+1)-1], \ldots, A_m[start(A_m, i) + 1, end(A_m, i + 1) - 1])| \geq |OPT_{clcs}[p_i + 1, p_{i+1} - 1]|.$$

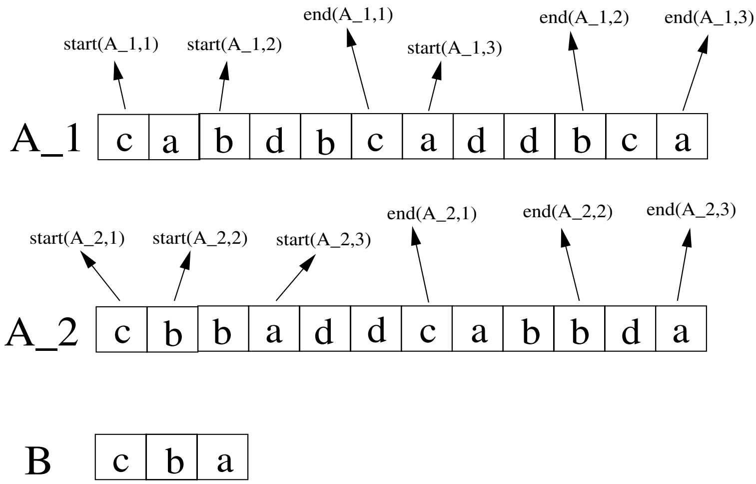

Fig. 3. An example of $start(A_i, j)$ and $end(A_i, j)$

Proof: Let us assume that there is an assignment of B over OPT_{clcs} such that: $|LCS(A_1[start(A_1,i)+1, end(A_1,i+1)-1], A_2[start(A_2,i)+1, end(A_2,i+1)-1], \ldots, A_m[start(A_m,i)+1, end(A_m,i+1)-1])| < |OPT_{clcs}[p_i+1, p_{i+1}-1]|$.

Note that, $OPT_{clcs}[p_i+1, p_{i+1}-1]$ must be a common subsequence of substrings of A_1, A_2, …, A_m. For every $j \leq m$, those substrings must start at a location $\geq start(A_j,i)+1$ and end at a location $\leq end(A_j, i+1)-1$. This contradicts the fact that the LCS of the substrings cannot be longer than the LCS of the original complete strings. □

The next two corollaries follows from Lemma 2.

Corollary 1. *Let $B = \langle b_1, b_2, \ldots, b_n\rangle$ be the constraint string and OPT_{clcs} be an optimal C-LCS. If we can find the LCS of $A_1, A_2, \ldots, A_m$, then we can approximate the C-LCS with a $\frac{1}{n+1}$-approximation ratio.*

Proof: Choosing the maximal LCS of $A_1[start(A_1,i)+1, end(A_1,i+1)-1]$, $A_1[start(A_1,i)+1, end(A_1,i+1)-1]$, …, $A_m[start(A_m,i)+1, end(A_m,i+1)-1]$ (over $0 \leq i \leq n$). W.L.O.G. let LCS_j be the maximal LCS and let j be the corresponding index. By Lemma 2 we get that $\langle b_1, b_2, \ldots, b_j\rangle \cdot LCS_j \cdot \langle b_{j+1}, b_{j+2}, \ldots, b_n\rangle \geq \frac{|OPT_{clcs}|}{(n+1)}$, where '·' denotes string concatenation. □

Corollary 2. *Let $B = \langle b_1, b_2, \ldots, b_n\rangle$ be the constraint string and OPT_{clcs} be an optimal C-LCS. If we can find an approximate LCS of $A_1, A_2, \ldots, A_m$, within an approximation ratio $\frac{1}{r}$, then we can approximate the C-LCS with a $\frac{1}{r(n+1)}$-approximation ratio.*

Proof: Using similar arguments to Corollary 1 and according to Lemma 2. □

Now, we give a short description of our algorithm (see Algorithm 1 for details). The structure of our algorithm is derived from Corollary 2. For every $i \leq n$, we simply compute an approximated LCS between $A_1[start(A_1, i) + 1, end(A_1, i + 1) - 1]$, $A_1[start(A_1, i) + 1, end(A_1, i + 1) - 1], \ldots, A_m[start(A_m, i) + 1, end(A_m, i + 1) - 1]$. We find the approximate LCS as follows:

For every $\sigma \in \Sigma$ and for every input string, denote with $C_{A_i}(\sigma, e, f)$ the number of $\sigma's$ in $A_i[e, f]$. For every $i \leq n$, let $C[\sigma, e_i, f_i] = \min(C_{A_i}(\sigma, e_i, f_i))$ and let $C^*(e_i, f_i) = \max C[\sigma, e_i, f_i]$ over all $\sigma \in \Sigma$.

With the use of $C[\sigma, e_i, f_i]$ and some additional arrays, the following lemma can be straightforwardly be seen to be true.

Lemma 3. *$C^*(e_i + 1, f_i)$ and $C^*(e_i, f_i + 1)$ can be computed from $C^*(e_i, f_i)$ in $O(k)$ time, given $O(\sum_{i=1}^{k} m_i)$ space.*

Our algorithm, perform one scan of A_i $(1 \leq i \leq k)$, from left to right. We can use two pointers for every string in order to scan it appropriately.

Algorithm 1. Linear Time Approximation Algorithm

$Occ \leftarrow 0$;
$bLoc \leftarrow 0$;
for $j \leftarrow 0$ **to** n **do**
  /* $1 \leq i \leq k$ */
  if $|C^*[start(A_i, j) + 1, end(A_i, j + 1) - 1]| > Occ$ **then**
    $Symbol \leftarrow$ The corresponding symbol of the above C^* ;
    $Occ \leftarrow |C^*[start(A_i, j) + 1, end(A_i, j + 1) - 1]|$;
    $bLoc \leftarrow j$;
return $B[1, bLoc] \cdot \langle Symbol^{Occ} \rangle \cdot B[bLoc + 1, n]$;

Time and Correctness Analysis:
Let C_{out} be the output string of the Algorithm 1, note that:

1) C_{out} is common subsequence of $A_1, A_2, \ldots, A_m$.
2) C_{out} contains B as a subsequence.

Thus, C_{out} is a feasible solution.

The running time is linear. The computation of $C^*[start(A_i, j)+1, end(A_i, j+1)-1]$ is a process of $2(\Sigma_{i=1}^{k}|m_i|)$ updates operations (we insert and delete every character of the input strings exactly once). Moreover, according to Lemma 3, we can perform k update operations in $O(k)$ time. Thus, the total running time remains linear.

Lemma 4. *Algorithm 1 yields an approximation ratio of $\frac{1}{\sqrt{m_{min}|\Sigma|}}$.*

Proof: We divide the proof into three cases. If $n \leq \sqrt{\frac{m_{min}}{|\Sigma|}} - 1$, then according to Lemma 2 and since the approximate LCS provide a $1/\Sigma$ approximation ratio, the length of the C-LCS returned by Algorithm 1 is at least $|OPT_{clcs}|/\sqrt{m_{min}|\Sigma|}$. Therefore, it is sufficient to prove that Algorithm 1 also yields an approximation ratio of $\frac{1}{\sqrt{m_{min}|\Sigma|}}$ in case that $n > \sqrt{\frac{m_{min}}{|\Sigma|}} - 1$.

Note that, if $n \geq \sqrt{\frac{m_{min}}{|\Sigma|}}$ any valid solution for the C-LCS must also provide an approximation ratio of $\frac{1}{\sqrt{m_{min}|\Sigma|}}$. Moreover, if $OPT_{clcs} > n$, we can see that Algorithm 1 returns at least one extra character over B. Thus, in case that $\sqrt{\frac{m_{min}}{|\Sigma|}} - 1 \leq n < \sqrt{\frac{m_{min}}{|\Sigma|}}$, our algorithm also yields an approximation ratio of $\frac{1}{\sqrt{m_{min}|\Sigma|}}$. □

5 Open Questions

A natural open question is whether there are better approximation algorithms for the single constraint C-LCS problem, which improves the above approximation factor ? Another interesting question is regarding the existence of a lower bound for this C-LCS variant.

References

1. Aho, A.V., Hirschberg, D.S., Ullman, J.D.: Bounds on the Complexity of the Longest Common Subsequence Problem. Journal of the ACM 23(1), 1–12 (1976)
2. Arslan, A.N., Egecioglu, Ö.: Algorithms For The Constrained Longest Common Subsequence Problems. International Journal of Foundations of Computer Science 16(6), 1099–1109 (2005)
3. Bergroth, L., Hakonen, H., Raita, T.: A Survey of Longest Common Subsequence Algorithms. In: Proc. SPIRE 2000, pp. 39–48 (2000)
4. Chin, F.Y.L., De Santis, A., Ferrara, A.L., Ho, N.L., Kim, S.K.: A simple algorithm for the constrained sequence problems. Information Processing Letters 90(4), 175–179 (2004)
5. Gotthilf, Z., Lewenstein, M.: Approximating Constrained LCS. In: Ziviani, N., Baeza-Yates, R. (eds.) SPIRE 2007. LNCS, vol. 4726, pp. 164–172. Springer, Heidelberg (2007)
6. Hirschberg, D.S.: A Linear Space Algorithm for Computing Maximal Common Subsequences. Communications of the ACM 18(6), 341–343 (1975)
7. Hirschberg, D.S.: Algorithms for the Longest Common Subsequence Problem. Journal of the ACM 24(4), 664–675 (1977)
8. Maier, D.: The Complexity of Some Problems on Subsequences and Supersequences. Journal of the ACM 25(2), 322–336 (1978)
9. Masek, W.J., Paterson, M.: A Faster Algorithm Computing String Edit Distances. Journal of Computer and System Sciences 20(1), 18–31 (1980)
10. Tsai, Y.-T.: The constrained longest common subsequence problem. Information Processing Letters 88(4), 173–176 (2003)