Minimal Height and Sequence Constrained Longest Increasing Subsequence

Chiou-Ting Tseng⋆, Chang-Biau Yang⋆† and Hsing-Yen Ann⋆
⋆Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
†cbyang@cse.nsysu.edu.tw

Abstract

Given a string $S = a_1 a_2 a_3 \cdots a_n$, the longest increasing subsequence (LIS) problem is to find a subsequence of S such that the subsequence is increasing and its length is maximal. In this paper, we propose and solve two variants of the LIS problem. The first one is the minimal height LIS where the height means the difference between the largest and smallest elements. We propose an algorithm with $O(n \log n)$ time and $O(n)$ space to solve it. The second one is the sequence constrained LIS that given a sequence S and a constraint C, we are to find the LIS of S containing C as its subsequence. We propose an algorithm with $O(n \log(n + |C|))$ time to solve it.

Keywords: bioinformatics, longest increasing subsequence, height, constraint

1 Introduction

Given a string $S = a_1 a_2 a_3 \cdots a_n$, *increasing subsequence (IS)* is a subsequence [23] $IS(S) = a_{i_1} a_{i_2} a_{i_3} \cdots a_{i_k}$ that $a_{i_p} < a_{i_q}$ if $i_p < i_q$, for $1 \le p < q \le k$. For example, consider $S = 41573$. Three of increasing subsequences of $S_1{=}41573$ are 13, 457 and 157. The *longest increasing subsequence (LIS)* problem is to find the longest among all increasing subsequences. Note that the LIS of a given string may not be unique. For example, both 457 and 157 are LIS's of $S = 41573$. The longest decreasing subsequence problem can also be solved similarly.

The LIS problem is interesting in both combinatorial perspective, such as pattern recognition, and biological applications. Delcher *et al.*[8] used LIS to help finding the whole genome alignment.

The rest of this paper is organized as follows. In Section 2, we will review some previous work on the LIS problem. In Section 3, we first review the previous work on the constrained LIS, and then solve the minimal height LIS and sequence constrained LIS problems. Finally, section 4 gives conclusions and some future work.

2 Previous Results

The LIS problem has been widely studied in the past decades. A straightforward method of finding the LIS is to obtain the longest common subsequence of the input string and the sorted input string, with time complexity $O(n^2)$. Schensted [19] is the first one who defined the LIS problem and proposed an algorithm with $O(n \log n)$ time. Hunt and Szymanski [10] improved the algorithm to $O(n \log \log n)$ time. And later, many papers [1, 3–5, 15, 16, 26] studied the LIS problem by using the van Emde Boas priority queue [22], which supports insert, delete, find, predecessor, and successor operations in $O(\log \log |\Sigma|)$ time, where Σ is the alphabet set of the input string. In LIS, the input can be transformed into an integer string where each integer is in $\{1, 2, 3, \cdots, n\}$ and $|\Sigma| = n$, so the LIS algorithm needs only $O(n \log \log n)$ time. Finding the LIS in streaming data has the limitation that the passed data can only be retained a limited number of times, Liben-Nowell *et al.* [14] gave an algorithm with $\log(1 + \frac{1}{\varepsilon})$ - pass, $O(\log l)$ or $O(\log \log \Sigma)$ updating time, and $O(l^{1+\varepsilon} \log \Sigma)$ space, where l is the length of the LIS. The length distribution of the LIS has been analyzed by Aldous and Diaconis [2]. In their result, the average length of the LIS of a string with length n is $2\sqrt{n}$.

Kim showed that finding the LIS is equivalent to finding the maximal independent set in a permutation graph[12]. A variant of the LIS problem is to find the *heaviest increasing subsequence(HIS)*, which is given a string S formed by Σ, and each symbol α in Σ has a weight $w(\alpha)$. The weight of a subsequence is the sum of the weights of all symbols contained in the subsequence. The HIS problem is to find the increasing subsequence with the maximal weight. As

the equivalence of LIS and the maximal independent set of the permutation graph, HIS is equivalent to the maximal weight independent set of the permutation graph. Several papers [6, 9, 13, 26] have devoted to the study of the maximal weight independent set problem in graphs, including permutation graphs.

A simple extension of the LIS problem is to find the LIS of every substring. In our previous work [21], we design an efficient preprocessing method, with $O(n^2)$ time, to solve it. After the preprocessing has been performed, the required answering time is linear to the output size.

Another extension of the LIS problem is the longest common increasing subsequence (LCIS) problem. Given two strings $A = a_1 a_2 a_3 \cdots a_m$, $B = b_1 b_2 b_3 \cdots b_n$ where each pair of symbols in A and B are comparable, the common increasing subsequence of A and B is $G = g_1 g_2 g_3 \cdots g_l$ where $g_1 = a_{i_1} = b_{j_1}$, $g_2 = a_{i_2} = b_{j_2}$, $\cdots$, $g_l = a_{i_l} = b_{j_l}$ and for all $1 \leq p < q \leq l$, $i_p < i_q$, $j_p < j_q$, $g_p < g_q$. The LCIS of A and B is the longest among all common increasing subsequences of A and B. Yang, Huang and Chao [25] proposed an algorithm for solving the LCIS problem in $O(n^2)$ time. In 2005, several papers tightened the upper bound, Katriel and Kutz [11] gave an algorithm with $O(nl \log n + Sort)$ time, where $Sort$ is the time required for sorting string B into nondecreasing order. Chan et al. [5] gave an algorithm with $O(\min(r \log l, nl + r) \log \log n + Sort))$ time, where r is the number of matched pairs between A and B. Brodal et al. [4] gave an algorithm with $O((m + nl) \log \log |\Sigma| + Sort)$ time. For small $|\Sigma|$, the algorithm has a tighter bound, $O(m)$ when $|\Sigma| = 2$, $O(m + n \log n)$ when $|\Sigma| = 3$. Yoshifumi [18] gives a linear space algorithm for the LCIS problem.

For the LCIS of multiple sequences, Chan et al. [5] gave an algorithm with $O(\min(Nr^2, Nr \log p \log Nr) + NSort_\Sigma(n))$ time, where N is the number of input sequences, and $Sort_\Sigma(n)$ denotes the time required for sorting all sequences. Brodal et al.[4] proposed an algorithm with $O(\min(Nr^2, r \log^{N-1} r \log \log r) + NSort_\Sigma(n))$ time.

3 Constrained LIS

Yang et al. [24] presented the first seen paper on the constrained LIS problem. They defined two types of constraints, the first one is that the difference between two neighboring elements in the increasing subsequence must be in $[L_V, U_V]$ and their positional distance in the original string must be in $[L_I, U_I]$. We call the difference between two neighboring elements as the *cliff* in this paper. They proposed an algorithm with $O(n \log(U_I - L_I))$ time and $O(n)$ space. The second constraint stipulates that the slope of two neighboring elements in LIS must be greater than a predefined value, where the slope is defined as their difference divided by their positional distance in the original string. They solved it in $O(n \log r)$ time and $O(n)$ space where r is the output size.

The *representative increasing subsequence* (RIS) of a string S is the principle row of the row tower proposed by Albert *et al.* [1]. The ith element in RIS of S is the minimal ending number of increasing subsequences with length i. For example, For $S = 41573$, its increasing subsequences are $\{4, 1, 5, 7, 3\}$, $\{45, 47, 15, 17, 13, 57\}$ and $\{457, 157\}$. The ending numbers of increasing subsequences with length 2 are $\{5, 7, 3\}$, thus the minimum is 3. So the RIS of S is 137. If an element is smaller than the ith element of the RIS, it can not be the ending number of increasing subsequence with length $i + 1$. So the length of the LIS ending at a certain element x can be found by searching for the largest element that is smaller than x in the previous RIS and add one to the position. Likewise, insertion of a new element x to RIS R is to find the minimal element in R greater than or equal to x and replace it by x, if x is greater than all elements in R then we append x to the end of R.

The constrained LIS problems we consider in this paper are the minimal height LIS and the sequence constrained LIS, the detail is described in the following subsections.

3.1 Minimal Height LIS

The *height* of an increasing string is defined as the difference between the largest and the smallest elements. The minimal height constraint is in fact the minimal sum version of the cliff constraint. Given a string $S = a_1 a_2 \cdots a_n$, the *minimal height LIS* (MHLIS) problem is to find an LIS with the minimal height. For example $S = 4683571$, its MHLIS is 457 or 467.

Let l_i denote the length of LIS ending at a_i. We record the maximal ending number smaller than a_i of IS's with length $l_i - 1$ in $a_1 a_2 \cdots a_{i-1}$ as the previous element of a_i in MHLIS.

Our algorithm for finding the MHLIS is given as follows.

Step 1: Maintain l binary search trees $q_1, q_2, \cdots, q_l$, where each q_j, $1 \leq j \leq l$,

records the ending numbers of all IS's with length j, and l denotes the length of the LIS.

Step 2: Whenever an element a_i is read, we find the LIS ending at a_i by the following steps.

Step 2.1: Perform binary search on RIS to find the smallest element that is greater than or equal to a_i. The position index of the element is the length l_i of the LIS ending at a_i.

Step 2.2: Suppose the predecessor of a_i in q_{l_i-1} is p_i. Add a_i to q_{l_i} and record the predecessor of a_i in the MHLIS ending at a_i as p_i.

Now, we prove the correctness of our algorithm.

Theorem 1. *Our approach finds the MHLIS ending at a_i.*

Proof. Suppose the LIS we find is $b_1 b_2 \cdots b_{l_i-1} a_i$. If we say that $d_1 d_2 \cdots d_{l_i-1} a_i$ has smaller height, for every corresponding b_k, d_k, we have $d_k \geq b_{k+1}$, d_k is not on the right of b_k. Here we use the term "not on the right" because b_k may be the same as d_k. We will show this by induction.

For $k = 1$, we have $a_i - d_1 < a_i - b_1$, so $d_1 > b_1$. If d_1 is on the right of b_1, then $b_1 d_1 d_2 \cdots d_{l_i-1} a_i$ is an IS with length $l_i + 1$, which is a contradiction. Thus, d_1 is not on the right of b_1. Also, b_1 is on the left of b_2, so d_1 is on the left of b_2. Suppose $d_1 < b_2$, and we have $d_1 > b_1$, then b_2 would choose d_1 instead of b_1, which is a contradiction. So $d_1 \geq b_2$.

Suppose the assumption holds for $k = m$, that is $d_m \geq b_{m+1}$ and d_m is not on the right of b_m. For $k = m + 1$, $d_{m+1} > b_{m+1}$ because $d_m \geq b_{m+1}$, $d_{m+1} > d_m$. If d_{m+1} is on the right of b_{m+1}, then $b_1 b_2 \cdots b_{m+1} d_{m+1} \cdots d_{l_i-1} a_i$ is an IS with length $l + 1$, which is a contradiction. So d_{m+1} is not on the right of b_{m+1}. Also, b_{m+1} is on the left of b_{m+2}, so d_{m+1} is on the left of b_{m+2}. Suppose $d_{m+1} < b_{m+2}$, and we have $d_{m+1} > b_{m+1}$, then b_{m+2} would choose d_{m+1} instead of b_{m+1}, which is a contradiction. So $d_{m+1} \geq b_{m+2}$. By the induction hypothesis, this assumption is true for all k.

Suppose that there exists $d_1 d_2 \cdots d_{l_i-1} a_i$ with smaller height. We have $d_{l_i-2} \geq b_{l_i-1}$, and $d_{l_i-1} > d_{l_i-2}$, so $d_{l_i-1} > b_{l_i-1}$. But since we select b_{l_i-1} instead of d_{l_i-1}, we have $b_{l_i-1} \geq d_{l_i-1}$ which is a contradiction. So there is no other IS of length l with smaller height. □

By Theorem 1, we can find the MHLIS ending at a certain element. We maintain the minimal height element in every q. To get the MHLIS, we start from the minimal height element in q_l and trace back the LIS by continuously switching to the predecessor of the current element.

The time complexity is analyzed as follows. For each a_i, we spend $O(\log n)$ time on deciding the length l of the LIS by binary search on the RIS, and $O(\log n)$ time on inserting a_i into the RIS. Finding predecessor in the set of ending numbers of IS's with length $l - 1$ takes $O(\log n)$ time, and inserting into the set of ending numbers of length l IS's also takes $O(\log n)$ time. Finally, calculating the height of the LIS ending at a_i takes constant time if the predecessor and the height of the LIS ending at the predecessor are given. Tracing out the predecessor takes $O(n)$ time by following the predecessor link. So totally we need $O(n \log n)$ time to find the MH-LIS. And the space requirement is $O(n)$.

3.2 Sequence Constrained LIS

Given a string $S = a_1 a_2 \cdots a_n$ and an increasing constraint $C = c_1 c_2 \cdots c_k$, the *sequence constrained longest increasing subsequence* (SCLIS) problem is to find an LIS containing C as its subsequence. Because we are finding an IS and the constraint is a subsequence of the IS, the constraint has to be increasing too.

Note that if there is no duplicated symbol in the input sequence, the problem becomes finding the occurrence of the constraints in the input sequence as follows. Let the constraint C be on positions $p_1, p_2, \cdots, p_k$ in S. To simplify the discussion, we add two dummy constraints $c_0 = -\infty$ in front of S and C and $c_{k+1} = \infty$ at the rear of S and C. Cut the input sequence by p_i for $1 \leq i \leq k$, find the LIS of each substring starting at p_i and ending at the previous element of p_{i+1} with value also starting at c_i and smaller than c_{i+1}, $0 \leq i \leq k$, and then we concatenate the IS's together. The concatenated IS is the answer. So the lower bound of the time complexity is the time required for finding the LIS.

This problem can be solved by the similar strategy solving the *constrained longest common subsequence* (CLCS) problem [7, 17, 20], the layered approach. First, we put the symbol c_i as the symbol of the ith floor, where the 0th floor means no constraint is satisfied. Because all the constraints need be satisfied, in the region after constraint c_i has been satisfied but constraint c_{i+1} has not been satisfied yet, we can only allow this part of IS in the final constrained LIS to be with value larger than or equal to c_i but smaller than c_{i+1}. Otherwise, c_{i+1} is unable to concatenate to the current increasing subsequence. Also, we only insert the element to the

priority queue when all constraints smaller than it is already in the queue.

In the ith floor, we maintain an RIS R_i of the elements greater than or equal to c_i and smaller than c_{i+1}. Originally when we insert an element into the RIS, we replace the smallest element larger than it which is the successor of it in the priority queue. But now, if the successor of the inserted element is one of the constraint, we do not delete it from the priority queue.

Our SCLIS algorithm is to insert the elements of S one by one from left to right into our data structure and the algorithm for inserting a new element is given as follows:

Input: The element to be inserted, e, and the constraint C.

Step1: Find the position of e in C. If $e = c_i$, go to step 2.1. If it is between c_i and c_{i+1}, go to step 2.2.

Step 2.1: If R_{i-1} is not NULL, set predecessor of e to the last element of R_{i-1}. If R_{i-1} is not NULL or $i = 1$, insert e to R_i.

Step 2.2: If c_i is already in R_i or $i = 0$, insert e to R_i, set predecessor of e to the predecessor of e in R_i.

In Step 2.1 we only set the predecessor of c_i and add it to R_i when R_{i-1} is not NULL. This means we only accept constraint c_i when all previous constraints have been satisfied. For example, in Figure 1, the first 7 can not be added since 3 has not been added yet. In Step 2.2, we only add e to R_i when c_i has appeared. It is based on the same reason, if c_i has not appeared yet, the elements with value greater than c_i can never be in the final constrained LIS. Take the first 4 in Figure 1 as an example, it can not be added because 3 is not in R_1 yet. The occurrence of c_i can easily be verified because if c_i is in R_i, it must be the first element of R_i.

The time complexity of this approach is analyzed as follows. If we use arrays to implement this approach. Step 1 takes $O(\log|C|)$ time by binary search on C. Step 2.1 takes constant time. Step 2.2 takes $O(\log n)$ time for doing binary search on R_i. So the time required for inserting one element is $O(\log n)$. There are n elements to be inserted, so the total construction time is $O(n\log(n + |C|))$. The output operation can be achieved in $O(n)$ time by continuously lookup the predecessor table. The space complexity of the R's is $O(n)$ because each element is in at most one of the R's and the predecessor table takes $O(n)$ space.

4   Conclusion and Future Works

In this paper, we propose and solve two variants of the LIS problem. The first one is the minimal height LIS. We propose an algorithm with $O(n\log n)$ time and $O(n)$ space to solve this problem. The second one is the sequence constrained LIS, we proposed an $O(n\log(n + |C|))$ time algorithm to solve it. The possible future work may involve trying to improve the algorithm or to prove that the complexity of our algorithm is optimal, solving other cliff based constraints like min-max, max-min, etc. And among all, exploring the applications of the LIS problems.

References

[1] M. H. Albert, A. Golynski, A. M. Hamel, A. Lopez-Ortiz, S. S. Rao, and M. A. Safari, "Longest increasing subsequences in sliding windows," *Theoretical Computer Science*, pp. 413–432, 2004.

[2] D. Aldous and P. Diaconis, "Longest increasing subsequences: From patience sorting to the baik-deift-johansson theorem," *BAMS: Bulletin of the American Mathematical Society*, Vol. 36, pp. 413–432, 1999.

[3] S. Bespamyatnikh and M. Segal, "Enumerating longest increasing subsequences and patience sorting," *Information Processing Letters*, Vol. 76, No. 1-2, pp. 7–11, 2000.

[4] G. S. Brodal, K. Kaligosi, I. Katriel, and M. Kutz, "Faster algorithms for computing longest common increasing subsequences," Tech. Rep. BRICS-RS-05-37, BRICS, Department of Computer Science, University of Aarhus, Dec. 2005.

[5] W. T. Chan, Y. Zhang, S. P. Y. Fung, D. Ye, and H. Zhu, "Efficient algorithms for finding a longest common increasing subsequence," *The 16th Annual International Symposium on Algorithms and Computation, Hainan, China*, pp. 665–674, 2005.

[6] M. S. Chang and F. H. Wang, "Efficient algorithms for the maximum weight clique and maximum weight independent set problems on permutation graphs," *Information Processing Letters*, Vol. 43, No. 6, pp. 293–295, 1992.

[7] F. Y. L. Chin, A. D. Santis, A. L. Ferrara, N. L. Ho, and S. K. Kim, "A simple algorithm for

	7	4	3	7	3	1	5	7	9	8
						1				
3			3		3		35			
7				7				7	79	78

Figure 1: The constrained LIS for $S = 7437315798$ and $C = 37$.

the constrained sequence problems," *Information Processing Letters*, Vol. 90, No. 4, pp. 175–179, 2004.

[8] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg, "Alignment of whole genomes," *Nucleic Acids Research*, Vol. 27, No. 11, pp. 2369–2376, 1999.

[9] W. L. Hsu, "Maximum weight clique algorithms for circular-arc graphs and circle graphs," *SIAM Journal on Computing*, Vol. 14, No. 1, pp. 224–231, 1985.

[10] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," *Communications of the ACM*, Vol. 20, No. 5, pp. 350–353, 1977.

[11] I. Katriel and M. Kutz, "A faster algorithm for computing a longest common increasing subsequence," Research Report MPI-I-2005-1-007, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, Mar. 2005.

[12] H. Kim, "Finding a maximum independent set in a permutation graph," *Information Processing Letters*, Vol. 36, No. 1, pp. 19–23, 1990.

[13] D. T. Lee and M. Sarrafzadeh, "Maximum independent set of a permutation graph in k tracks," *International Journal of Computational Geometry and Applications*, Vol. 3, No. 3, pp. 291–304, 1993.

[14] D. Liben-Nowell, E. Vee, and A. Zhu, "Finding longest increasing and common subsequences in streaming data," *11th International Computing and Combinatorics Conference, Kunming, China*, pp. 263–272, 2005.

[15] E. Mäkinen, "On the longest upsequence problem for permutations," Tech. Rep. A-1999-7, Department of Computer Science, University of Tampere, 1999.

[16] F. Malucelli, T. Ottmann, and D. Pretolani, "Efficient labelling algorithms for the maximum noncrossing matching problem," *Discrete Applied Mathematics*, Vol. 47, No. 2, pp. 175–179, 1993.

[17] C.-L. Peng, "An approach for solving the constrained longest common subsequence problem," *Master Thesis, Department of Computer Science and Engineering, National Sun Yat-sen University, Taiwan*, July 2003.

[18] Y. Sakai, "A linear space algorithm for computing a longest common increasing subsequence," *Information Processing Letters*, Vol. 99, No. 5, pp. 203–207, 2006.

[19] C. Schensted, "Longest increasing and decreasing subsequences," *Canadian Journal of Mathematics*, Vol. 13, pp. 179–191, 1961.

[20] Y.-T. Tsai, "The constrained longest common subsequence problem," *Information Processing Letters*, Vol. 88, pp. 173–176, 2003.

[21] C. T. Tseng, S. H. Shiau, and C. B. Yang, "An optimal algorithm for finding the longest increasing subsequence of every substring," *Proceeding of the 5th Conference on Information Technology and Applications in Outlying Islands*, p. 14, 2006.

[22] P. van Emde Boas, R. Kaas, and E. Zijlstra, "Design and implementation of an efficient priority queue," *Mathematical Systems Theory*, Vol. 10, pp. 99–127, 1977.

[23] C. B. Yang and R. C. T. Lee, "Systolic algorithms for the longest common subsequence problem," *Journal of the Chinese Institute of Engineers*, Vol. 10, No. 6, pp. 691–699, 1987.

[24] I. H. Yang and Y. C. Chen, "Fast algorithms for the constrained longeest increasing subsequence problems," *Proceeding of the 25th Workshop on Combinatorial Mathematics and Computation Theory*, pp. 226–231, 2008.

[25] I. H. Yang, C. P. Huang, and K. M. Chao, "A fast algorithm for computing a longest common increasing subsequence.," *Information Processing Letters*, Vol. 93, No. 5, pp. 249–253, 2005.

[26] M. S. Yu, L. Y. Tseng, and S. J. Chang, "Sequential and parallel algorithms for the maximum-weight idependent set problem on permutation graphs," *Information Processing Letters*, Vol. 46, No. 1, pp. 7–11, 1993.