

An Efficient Algorithm for the Longest Pattern Subsequence Problem

Xiaodong Wang^{1,2}, Lei Wang³ and Yingjie Wu¹

¹Department of Computer Science, Fuzhou University, Fuzhou, P. R. China

²Department of Computer Science, Quanzhou Normal University, Quanzhou, P. R. China

³College of Computing, Georgia Institute of Technology, Atlanta GA 30332, USA

Abstract - This paper studies the longest pattern subsequence problem based on the dynamic programming algorithm. An $O(n^2)$ time algorithm for the problem is firstly presented. Then the presented algorithm is improved further by generalizing the RSK algorithm and the standard Young tableau. When $|\mathcal{S}| = 1$, the time required by the algorithm is improved to $O(n \log k)$. When $|\mathcal{S}| = 2$, the time required by the algorithm is improved to $O(n)$, an optimal algorithm.

Keywords: \mathcal{S} pattern subsequence, dynamic programming algorithm, RSK algorithm, standard Young tableau.

1 Introduction

Both LIS and LCS are fundamental combinatorial problems which have been well studied in the computer science community [2][3][5][6][7][8]. Among a large number of important applications of both of the problems, we highlight a few that arise in computational biology. The BLAST (Basic Local Alignment Search Tool) [1] database supports queries of the following form: for a sequence \mathcal{S} of amino acids, for example, what segments of known proteins have high local similarity to \mathcal{S} ? An LIS step is also part of the MUMmer system for aligning entire genomes [4], and a straightforward LCS computation gives the value of the optimal alignment of two sequences of DNA. Based on these fundamental problems, a more general longest pattern subsequence problem was proposed [9][10][11]. In the general cases, the length of the longest pattern subsequence of a permutation in the symmetric group S_n is still an open

problem [9]. In this paper we presented a dynamic programming algorithm for the longest pattern subsequence problem. The presented algorithm is improved further by generalizing the RSK algorithm and the standard Young tableau. When $|\mathcal{S}| = 1$, the time required by the algorithm is improved to $O(n \log k)$. When $|\mathcal{S}| = 2$, the time required by the algorithm is improved to $O(n)$, an optimal algorithm.

The longest pattern subsequence problem can be formulated as follows.

Let \mathcal{S} be a finite word in the letters U and D , e.g. $\mathcal{S} = UUDUD$. Let \mathcal{S}^∞ denote the infinite word $\mathcal{S}SS\mathbf{L}$, e.g., $(UUDUD)^\infty = UUDUDUUDUD\mathbf{L}$.

For this example, we have for instance that $UUDUDUU$ is a prefix of \mathcal{S}^∞ of length 7.

Let $\mathbf{t} = t_0 t_1 \mathbf{L} t_{m-1}$ be a word of length $m-1$ in the letters U and D . A sequence $\mathbf{v} = v_0 v_1 \mathbf{L} v_m$ of integers is said to have descent word \mathbf{t} if $v_i > v_{i+1}$ whenever $t_i = D$, and $v_i < v_{i+1}$ whenever $t_i = U$. Thus \mathbf{v} is increasing if and only if $\mathbf{t} = U^m$.

Now let S_n denote the symmetric group of permutations of $1, 2, \dots, n$, and let $a \in S_n$ and define $len_s(a)$ to be the length of longest subsequence of a whose descent word is a prefix of \mathcal{S}^∞ . The longest pattern subsequence problem is

defined to compute $len_s(a)$, the length of longest subsequence of a whose descent word is a prefix of S^∞ for a given pattern S .

2 Dynamic Programming Algorithm

Let S be the given pattern, and $S(k)$ a prefix of S^∞ of length k .

Let $a = a_0 a_1 \mathbf{L} a_{n-1} \in W_n$ be a permutation of $1, 2, \dots, n$.

All the S pattern subsequences ending with a_i is denoted as $a_s(i)$, $0 \leq i < n$. Thus for any $v \in a_s(i)$, $v = v_0 v_1 \mathbf{L} v_{m-1}$, we have, $|v| = m$, $v_{m-1} = a_i$, and the descent word of v is $S(m-1)$, a prefix of S^∞ of length $m-1$.

If for any $j > i$, the descent word of $v + a_j = v_0 v_1 \mathbf{L} v_{m-1} a_j$ is $S(m)$, a prefix of S^∞ of length m , then $v + a_j$ is defined as an extension of v , denoted as $v \mathbf{P} a_j$.

Let $w_s(i) = w_0 w_1 \mathbf{L} w_{m-1}$ be the longest S pattern subsequences of $a_s(i)$, then $w_{m-1} = a_i$. Its length is denoted as $b_i = m$. That is $b_i = \max\{|v| \mid v \in a_s(i)\}$, $0 \leq i < n$. Therefore, $len_s(a) = \max_{0 \leq i < n} \{b_i\}$. The problem is then reduced to compute $b_i, 0 \leq i < n$.

It is not difficult to see that the problem satisfies the optimal substructure property. In fact, if $b_i = m$, and $w_s(i) = w_0 w_1 \mathbf{L} w_{m-1}$, $w_{m-2} = a_k$, $w_{m-1} = a_i$, then $b_k = m-1$. Otherwise, $b_k \geq m-1$, since $w_0 w_1 \mathbf{L} w_{m-2}$ is a S pattern subsequence of v ending with a_k . If $b_k > m-1$, then there exists a S pattern subsequence of v ending with a_k of length larger than $m-1$, and then $v a_i$

is a S pattern subsequence of v ending with a_k of length larger than m . Therefore $b_i > m$, a contradiction.

From the optimal substructure property of the problem, b_i can be computed recursively as follows:

$$b_i = \begin{cases} 1 & i = 0 \\ \max_{0 \leq k < i} \{b_k + 1 \mid w_s(k) \mathbf{P} a_i\} & 0 < i < n \end{cases}$$

The dynamic programming algorithm according to above formula can be described as follows.

Dynamic Programming Algorithm:

- 1: $b_0 \leftarrow 1$;
- 2: for $i \leftarrow 1$ to $n-1$ do
- 3: $k \leftarrow 0$;
- 4: for $j \leftarrow 0$ to $i-1$ do
- 5: if ($w_s(j) \mathbf{P} a_i$ and $k < b_j$) then
- 6: $k \leftarrow b_j$;
- 7: $b_i \leftarrow k + 1$;
- 7: $len \leftarrow \max_{0 \leq i < n} \{b_i\}$.

In the algorithm above, the length of $w_s(j)$ is b_j . It can be decided in $O(1)$ time whether a_i is an extension of $w_s(j)$. The two for loops imply that the time required by the algorithm is $O(n^2)$.

The above algorithm computes the length of the longest pattern subsequence only. If we also want to construct the longest pattern subsequence, the algorithm can be modified as follows. For each j , the algorithm maintains the subsequences $w_s(j)$. When we update $b_i \leftarrow b_j + 1$, we reset $w_s(i) \leftarrow w_s(j) a_i$. This adds only a constant amount of extra running

time per update, so the time required by the algorithm remains $O(n^2)$.

3 An Improvement

3.1 The case of $|\mathcal{S}| = 1$

Let $I = (I_1, I_2, \mathbf{L})$ be a partition of $n \geq 0$, denoted $I \mathbf{a} n$ or $|I| = n$. Thus $I_1 \geq I_2 \geq \mathbf{L} \geq 0$, and $\sum I_i = n$. The Young diagram of a partition I is a left justified array of squares with I_i squares in the i th row. For instance, the Young diagram of $(3, 2, 2)$ is given by figure 1.

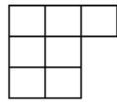


Fig. 1. The Young diagram of $(3, 2, 2)$

A standard Young tableau (SYT) of shape $I \mathbf{a} n$ is obtained by placing the integers $1, 2, \dots, n$ (each appearing once) into the squares of the diagram of I (with one integer in each square) such that every row and column is increasing. For example, an SYT of shape $(3, 2, 2)$ is given by figure 2.

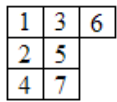


Fig. 2. An SYT of shape $(3, 2, 2)$

The RSK (Robinson - Schensted - Knuth) algorithm gives a bijection between w_n and the SYT of shape $I \mathbf{a} n$. For any given $a = a_0 a_1 \mathbf{L} a_{n-1} \in w_n$, the basic operation of the RSK algorithm is row insertion, i.e., inserting an integer i into a tableau T with distinct entries and with increasing rows and columns. (Thus T satisfies the conditions of an SYT except that its entries can be any distinct integers, not just $1, 2, \dots, n$.) The process of row inserting i into T produces another tableau, denoted $T \leftarrow i$, with increasing rows and columns. If S is the set of entries

of T , then $S \cup \{i\}$ is the set of entries of $T \leftarrow i$. We define $T \leftarrow i$ recursively as follows.

- (1) If the first row of T is empty or the largest entry of the first row of T is less than i , then insert i at the end of the first row.
- (2) Otherwise, i replaces (or bumps) the smallest element j in the first row satisfying $j > i$. We then insert j into the second row of T by the same procedure.

For instance, if $a = 4271536$, then the process of row inserting of the RSK algorithm is given by figure 3.

4	2	27	17	15	13	136
	4	4	2	27	25	25
			4	4	47	4

Fig. 3. The row inserting of the RSK algorithm

The length of the first row of an SYT is of special importance. It is equal to the length of the LIS for the given sequence $a = a_0 a_1 \mathbf{L} a_{n-1} \in w_n$. Furthermore the LIS of a can be constructed from the first row of the SYT corresponding to a . This is the special case of the longest \mathcal{S} pattern subsequence problem when $\mathcal{S} = U$. It is easy to see that if $\mathcal{S} = D$ an algorithm can be derived similarly. This implies an improved algorithm for the longest \mathcal{S} pattern subsequence problem when $|\mathcal{S}| = 1$ as follows.

Algorithm of $|\mathcal{S}| = 1$:

- 1: $b[1] \leftarrow e[0] \leftarrow 0$
- 2: $j \leftarrow k \leftarrow f \leftarrow 1$
- 3: **for** $i \leftarrow 1$ **to** $n-1$ **do**
- 4: $x \leftarrow b[k]$
- 5: **if** $w_s(x) \mathbf{p} a_i$ **then** $j \leftarrow k \leftarrow k+1$
- 6: **else** $j \leftarrow \text{binary}(i, f, k)$
- 7: $b[j] \leftarrow i$
- 8: $e[i] \leftarrow b[j-1]$
- 9: **construct**(k)
- 10: **return** k

In the algorithm above , the length of $w_s(j)$ is $b[j]$. It can be decided in $O(1)$ time whether a_i is an extension of $w_s(j)$. The position in the first row of the SYT between $b[f]$ and $b[k]$ for a_i to be inserted into is computed by a binary search of $\text{binary}(i,f,k)$. When a_i is inserted into the first row of the SYT, its left element is stored in $e[i]$ which is used to construct the longest pattern subsequence by $\text{construct}(k)$ later. If the length of the longest pattern subsequence is k , then each binary search costs $O(\log k)$ time in the worst case. The total time of the algorithm is clearly $O(n \log k)$ which is a significant improvement.

3.2 The case of $|\mathcal{S}| = 2$

When $|\mathcal{S}| = 2$, there are two cases $\mathcal{S} = DU$ and $\mathcal{S} = UD$ to be distinguished.

For any given permutation $a = a_0 a_1 \dots a_{n-1} \in w_n$, if $a_{i-1} > a_i < a_{i+1}$, then a_i is called a local minimum of a . Similarly, if $a_{i-1} < a_i > a_{i+1}$, then a_i is called a local maximum of a . It is easy to see that the local minimum and the local maximum of a is occurred alternatively in a . If we use a 1 denote a local maximum and a 0 denote a local minimum, then for any given permutation $a = a_0 a_1 \dots a_{n-1} \in w_n$, the descent pattern of a must be $0101\dots$ or $1010\dots$. For instance, the descent pattern of $a = 4326517$ is 10101 as shown in figure 4.

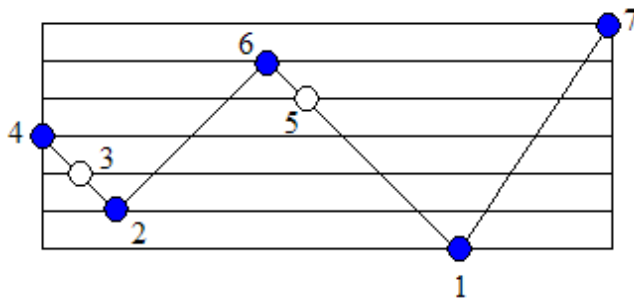


Fig. 4. The descent pattern of $a = 4326517$

When $|\mathcal{S}| = 2$, the \mathcal{S} pattern subsequence is called alternating subsequence. For the longest alternating subsequences, we have follow theorem.

Theorem 1: For the given permutation $a = a_0 a_1 \dots a_{n-1} \in w_n$, the alternating subsequence formed by its local minimum and local maximum is a longest alternating subsequence of a .

Let the alternating subsequence formed by the local minimum and local maximum of a be $v = v_0 v_1 \dots v_{k-1}$, then

- (1) In the case of $\mathcal{S} = DU$, if the descent pattern of $v = v_0 v_1 \dots v_{k-1}$ is $0101\dots$, then the subsequence $v_1 \dots v_{k-1}$ is a longest alternating subsequence of a . If the descent pattern of $v = v_0 v_1 \dots v_{k-1}$ is $1010\dots$, then the subsequence $v_0 v_1 \dots v_{k-1}$ is a longest alternating subsequence of a .
- (2) In the case of $\mathcal{S} = UD$, if the descent pattern of $v = v_0 v_1 \dots v_{k-1}$ is $0101\dots$, then the subsequence $v_0 v_1 \dots v_{k-1}$ is a longest alternating subsequence of a . If the descent pattern of $v = v_0 v_1 \dots v_{k-1}$ is $1010\dots$, then the subsequence $v_1 \dots v_{k-1}$ is a longest alternating subsequence of a .

Proof:

The proof is by induction on n . The theorem can be verified easily if $n \leq 3$. We now assume that the theorem is true when the length of a is less than n .

- (1) In the case of $\mathcal{S} = DU$, it can be proved that there exists a longest alternating subsequence of $a = a_0 a_1 \dots a_{n-1}$ containing the first local maximum of a . In fact, if the descent pattern of $v = v_0 v_1 \dots v_{k-1}$ is $1010\dots$, then $v_0 = a_0$ is the first local maximum of a .

Let $u = u_0 u_1 \mathbf{L}$ be a longest alternating subsequence of a and $u_0 \neq v_0$. There are two cases to discuss. Firstly, if $u_0 \leq v_0$ we can replace u_0 with v_0 in the subsequence u getting another longest alternating subsequence of a which contains v_0 , the first local maximum of a . Secondly, if $u_0 > v_0$ then it is easy to see that u_0 must lie behind v_1 and $u_0 > v_0 > v_1$. Therefore, $v_0 v_1 u_0 u_1 \mathbf{L}$ is a longer alternating subsequence of a , a contradiction.

Similarly, if the descent pattern of $v = v_0 v_1 \mathbf{L} v_{k-1}$ is $0101\mathbf{L}$, then v_1 is the first local maximum of a . Let $u = u_0 u_1 \mathbf{L}$ be a longest alternating subsequence of a . If u_0 lies between v_0 and v_1 or between v_1 and v_2 , we can replace u_0 with v_1 in the subsequence u getting another longest alternating subsequence of a which contains v_1 , the first local maximum of a . If u_0 lies behind v_2 , a contradiction can be conducted similarly.

When we get the first local maximum of a , the problem of finding a longest alternating subsequence of a in pattern $\mathcal{S} = DU$ is reduced to the problem of finding a longest alternating subsequence of a subsequence of a in pattern $\mathcal{S} = UD$. By the induction hypothesis, the part (1) of the theorem follows.

The proof for the part (2) of the theorem is similar. \blacklozenge

From theorem 1, we can generalize the standard Young tableau and the RSK algorithm as follows. The two basic operations of the RSK algorithm are replace and insert at the end. When $|\mathcal{S}| = 2$, the rules for the operations replace and insert at the end can be changed to: if the descent pattern changes, then performs insert at the end, otherwise performs replace. For instance, the process of the generalized RSK algorithm for permutation $a = 4326517$ in pattern $\mathcal{S} = DU$ is given by figure 5.

4	43	42	426	4265	4261	42617
		3	3	3	35	35

Fig. 5. The generalized RSK algorithm

It is easy to see that the first row of the generalized standard Young tableau is a longest alternating subsequence of a in pattern $\mathcal{S} = DU$.

The algorithm description for the generalized RSK algorithm is very simple. Only one extra statement added to the algorithm for the case of $|\mathcal{S}| = 1$ as follows.

Algorithm of $|\mathcal{S}| = 1$:

- 1: $b[1] \leftarrow e[0] \leftarrow 0$
- 2: $j \leftarrow k \leftarrow f \leftarrow 1$
- 3: **for** $i \leftarrow 1$ **to** $n-1$ **do**
- 4: $x \leftarrow b[k]$
- 5: **if** $w_{\mathcal{S}}(x) \mathbf{p} a_i$ **then** $j \leftarrow k \leftarrow k+1$
- 6: **else** $j \leftarrow \text{binary}(i, f, k)$
- 7: $b[j] \leftarrow i$
- 8: $e[i] \leftarrow b[j-1]$
- 9: **if** $(k < n-1 \text{ and } \mathcal{S}_{k-1} \neq \mathcal{S}_k)$ **then** $f \leftarrow k$
- 10: $\text{construct}(k)$
- 11: **return** k

The new statement $\text{if}(k < n-1 \text{ and } \mathcal{S}_{k-1} \neq \mathcal{S}_k) \text{ then } f \leftarrow k$, makes the length of search range for the binary search become 1. Thus no binary search performed. Therefore the time complexity of the algorithm for the case of $|\mathcal{S}| = 2$ is $O(n)$, an optimal algorithm.

4 Open Problems

We proposed an $O(n^2)$ time dynamic programming algorithm for the longest pattern subsequence problem. Then the presented algorithm is improved further by generalizing the RSK algorithm and the standard Young

tableau. When $|S| = 1$, the time required by the algorithm is improved to $O(n \log k)$. When $|S| = 2$, the time required by the algorithm is improved to $O(n)$, an optimal algorithm.

The following problems are still open.

(1) Can the RSK algorithm and the standard Young tableau be generalized to solve the longest pattern subsequence problem when $|S| \geq 3$?

(2) What is the lower bound of the computing time for the longest pattern subsequence problem in general cases?

5 References

- [1]. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(1990) 403-410
- [2]. A. Apostolico and C. Guerra. The longest common subsequence problem revisited. *Algorithmica*, 2(1987) 315-336
- [3]. Sergei Bspamyatnikh and Michael Segal. Enumerating longest increasing subsequences and patience sorting. *Information Processing Letters*, 76(2000) 7-11
- [4]. A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(1999) 2369-2376
- [5]. M. L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1975) 29-35
- [6]. Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24(1977) 644-675
- [7]. J. Hunt and T. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(1977) 350-353

[8]. C. Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of Mathematics*, 13(1961) 179-191

[9]. R. Stanley, Longest alternating subsequences of permutations. Preprint, 2005; math. CO / 0511419. <http://www-math.mit.edu/~rstan/pubs/>

[10]. R. Stanley, Increasing and decreasing subsequences and their variants, in *Proceedings of International Congress of Mathematicians 2006 (ICM2006)*.

<http://www-math.mit.edu/~rstan/pubs/>

[11]. R. Stanley, Alternating permutations and symmetric functions, *J. Combinatorial Theory (A)*, 2006, to appear.

<http://www-math.mit.edu/~rstan/pubs/>