The Longest Common Permutation Problem

Wei-Fu Lu

Department of Bioinformatics, Asia University
Department of Computer Science and Information Engineering, Asia University
weifu@asis.edu.tw

Abstract

A permutation $P = (p_1, p_2, \ldots, p_k)$ of 1 to k is called a pattern of the permutation $T = (t_1, t_2, \ldots, t_n)$ of 1 to n, if and only if there is a length k subsequence T' of T such that the elements of T' are ordered according to the permutation P. The longest common permutation problem is of finding the longest pattern which is involved in two permutations π_1 and π_2. In this paper, we give a $O(n^6)$ time algorithm to find the longest common permutation between two permutations when one permutation is separable. Our results improve the Bouvel's $O(n^8)$ time algorithm in [2].

Key words: pattern, permutation, separating tree.

1. Introduction

The longest common permutation problem can be defined as follows: Given two permutations π_1 and π_2, we want to find the longest pattern which is involved in π_1 and π_2. A permutation $P = (p_1, p_2, \ldots, p_k)$ of 1 to k is called a pattern of the permutation $T = (t_1, t_2, \ldots, t_n)$ of 1 to n, if and only if there is a length k subsequence of T, say $T' = (t_{i_1}, t_{i_2}, \ldots t_{i_k})$, with $i_1 < i_2 < \ldots < i_k$, such that the elements of T' are ordered according to the permutation P, that is, $t_{i_r} < t_{i_s}$ iff $p_r < p_s$. If T does contain such a subsequence, we will say that T contains P, or that P matches into T. The longest common permutation problem is a generalization of the pattern involvement problem that is to decide whether P is a pattern matches into T. The general pattern involvement problem has shown to be NP-complete [1], but some polynomial solutions exist for special kinds of patterns like the separable ones [1, 4]. A permutation is separable, if it contains neither the pattern (2, 4, 1, 3) and (3, 1, 4, 2).

The study of patterns in permutations has become very active in the recent years from both combinatorial and algorithmic research area. Form a combinatorial point of view, there are considerable many interested results in counting the number of permutations T that do not contain a smaller permutation P. Knuth [5] showed that the number of permutations of 1, ... , n without the pattern (3, 1, 2) is equal to the *nth* Catalan number. Lovász [6] considered permutations without the pattern (2, 1, 3). Rotem [7] considered permutations without either of the patterns (2, 3, 1) or (3, 1, 2). Simion and Schmidt [8] go further, counting the number of permutations avoiding all the patterns in any subset of the permutations on 1, 2, 3. West [9] showed that the number of separable n-permutations is the $(n-1)$-*st* Schröder number. From an algorithmic point of view, we want to design algorithms for general pattern involvement and common permutation probelm. Bose [1] gave an algorithm for counting the number of pattern matching in permutations with $O(kn^6)$ time and $O(kn^4)$ space, when P is separable (the early version of this paper is in LNCS 709, page 200-209, 1993). Ibarra [4] improve the Bose's results in 1993 to get an algorithm with $O(kn^4)$ time and $O(kn^3)$ space. Bouvel [2] proposed the longest common permutation problem and give a $O(n^8)$ time algorithm. In this paper, we give a $O(n^6)$ time algorithm to find the longest common permutation between two permutations based on the work of [4], when one permutation is separable.

2. Separable permutation and separating tree

A permutation σ of size n is called separable if it avoids the patterns 3 1 4 2 and 2 4 1 3 or equivalently if it has a binary separating tree. A permutation σ that does not contain π as a pattern is said to avoid π. A separating tree for a permutation $(p_1, \ldots ,p_k)$ of 1, ... , k is an ordered binary tree T with leaves $(p_1, \ldots ,p_k)$ in that order, such that for each node V, if the leaves of the subtree rooted at V are $p_i, p_{i+1},\ldots, p_{i+j}$, then the set of numbers $\{p_i, p_{i+1},\ldots, p_{i+j}\}$ is a subrange of the range 1, ... , k. This subrange is called the range of the node V. Let V_l be the left child and V_r be the right child of node V respectively. Node V is called a positive node we say, if the range of V_l just precedes the range of V_r, and V is called a negative node vice versa. Figure 1 is an example of separating tree for permutation {3, 4, 2, 1, 6, 5, 7, 8}. Note that the separating tree need not be unique. For example, the permutation (4, 5, 3, 1, 2) has the two separating trees (((4, 5), 3), (1,2)) and ((4, 5), (3, (1, 2))).

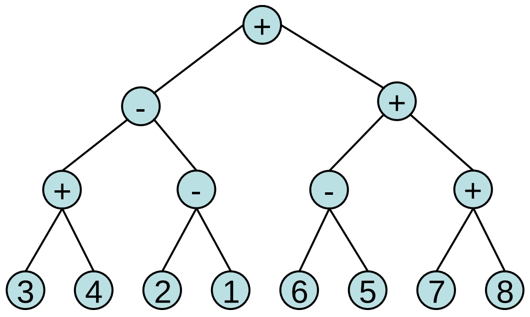

Figure 1 Separating tree for permutation {3, 4, 2, 1, 6, 5, 7, 8}.

Consider a well-known result from graph theory. For any permutation P of 1, ..., n, a graph can be defined with vertices 1, ..., n and with an edge (i, j) for $i < j$ iff i appears after j in the permutation. The graphs formed this way are permutation graphs. It is easy to see that a permutation containing the pattern (3, 1, 4, 2) or (2, 4, 1, 3) is equivalent to the graph containing an induced P_4. In general, graphs that do not contain an induced path on 4 vertices are called P_4-free graphs. Thus, the recognition of separable permutation is equivalent to the recognition of P_4-free graphs, which can be accomplished in linear time [3].

3. The Algorithm

In this section, we describe a polynomial time algorithm for finding a longest common permutation between two permutations. The input of our algorithm is a permutation τ, and a separating tree T_σ of a separable permutation σ. Our algorithm uses the dynamic programming strategy, and defines the following array should be filled. For every node V of T_σ, $1 \le i \le j \le n$, and $1 \le x \le n$, let $S_{ML}(V, i, j, x)$ be the set of longest common permutations in the match of V into $(\tau_i, \ldots, \tau_j)$ with every matched element of $(\tau_i, \ldots, \tau_j) \le x$, and $S_{MR}(V, i, j, x)$ be the set of longest common permutations in the match of V into $(\tau_i, \ldots, \tau_j)$ with every matched element of $(\tau_i, \ldots, \tau_j) \ge x$. Define $L(V, i, j, x) = \max \{0\} \cup \{y$: y is the smallest matched element of m in $S_{ML}\}$, and $M_L(V, i, j, x)$ = the longest common permutations in S_{ML} with the smallest matched element is $L(V, i, j, x)$. Similarly, define $R(V, i, j, x) = \min \{n+1\} \cup \{y$: y is the largest matched element of m in $S_{MR}\}$, and $M_R(V, i, j, x)$ = the longest common permutations in S_{MR} with the largest matched element is $R(V, i, j, x)$. Note that $L(V, i, j, x) = 0$, if $M_L(V, i, j, x)$ is empty, and $H(V, i, j, x) = n+1$, if $M_H(V, i, j, x)$ is empty.

We now describe the algorithm to compute a node's L, M_L. The computation of H, M_H is similar. Initially, all values of L are set to 0, all values of H are set to n+1, and all values of both M_L and M_H are set to be empty. For any leaf node V, do the following:

```
$L(V, i, j, x) = \max_{i \le l \le j} \{t_l \mid t_l \le x\}$
**if** $L(V, i, j, x) > 0$ **then** $M_L(V, i, j, x) = L(V, i, j, x)$
```

Let V be a positive node with left child V_l, and right child V_r, the following routines compute the L and M_L values for V from the L and M_L for V_l, and H and M_H for V_r. Note that $x \circ y$ is the string concatenation of x and y.

```
/* part 1*/
for $k = i+1$ to $j$
for $x = 2$ to $n$
  $low = L(V_l, i, k-1, x-1)$
  $high = H(V_r, k, j, x)$
      if $low > 0$ or high $< n+1$ then
         if $low = 0$ then $low = x$
         if $high = n+1$ then $high = x-1$
         if $|M_L(V_l, i, k-1, x-1)| + |M_H(V_r, k, j, x)| > |M_L(V, i, j, \text{high})|$ and $low > L(V, i, j, high)$
         then
             $L(V, i, j, high) = low$
             $M_L(V, i, j, high) = M_L(V_l, i, k-1, x-1) \circ M_H(V_r, k, j, x)$
/*part 2*/
$best = L(V, i, j, 1)$
for $x = 2$ to $n$
   if $L(V, i, j, x) > best$
      $best = L(V, i, j, x)$
   else
      $L(V, i, j, x) = best$
      $M_L(V, i, j, x) = M_L(V, i, j, x-1)$
```

4. The correctness and time complexity of the algorithm

In this section, we prove the correctness of our algorithms, and analysis its time complexity.

Lemma 3.1 *When the algorithm halts, $M_L(V, i, j, x)$ is the longest common permutation in the match of V into $(\tau_i, \ldots, \tau_j)$ with every matched element of $(\tau_i, \ldots, \tau_j) \leq x$ and the smallest matched element in this match is* $L(V, i, j, x)$.

Proof. This lemma is proved by induction. If V is a leaf, the above statement is clearly true. Let V be an internal node with left child V_l, and right child V_r. Without loss of generality, assume that V is positive. The case of negative node is symmetric. Assume that M_L and M_R are generated correctly for V_l and V_r with respect to all value of i, j, and x. There two cases should be discussed. Case 1 $M_L(V, i, j, x)$ is not empty when the part 1 of the algorithm is finished. Consider a longest common permutation π in the match of V into $(\tau_i, \ldots, \tau_j)$ with every matched element $\leq x$. It is easy to see that, there exist two numbers k and x', such that $\pi=\pi_l \circ \pi_r$, where π_l is the longest common permutation in $S_{ML}(V, i, k\text{-}1, x'\text{-}1)$ with the smallest matched element is $L(V, i, k\text{-}1, x'\text{-}1)$, and π_r is the longest common permutation in $S_{MR}(V, k, j, x')$ with the largest matched element is $R(V, k, j, x')$. Note that in this decomposition, π_l or π_r might be empty. If π_l (respectively π_r) is not the maximum pattern for the given intervals of indices and values, then π would not be of maximum value, contradiction to the definition of π. By induction hypothesis, $M_L(V_l, i, k\text{-}1, x'\text{-}1) = \pi_l$, $M_H(V_r, k, j, x') = \pi_r$ and $H(V_r, k, j, x') = x$. The pattern π will be set to $M_L(V, i, j, x)$ if $L(V_l, i, k\text{-}1, x'\text{-}1)$ is larger than the $L(V, i, j, x)$, such that the permutation stored in $M_L(V, i, j, x)$ is the longest common permutation in $S_{ML}(V, i, j, x)$ with the smallest matched element in this match is $L(V, i, j, x)$. Case 2, $M_L(V, i, j, x)$ is empty when the part 1 of the algorithm is finished. It is easy to see that, there is a number $x' < x$ such that the longest common permutation in $S_{ML}(V, i, j, x')$ is also in $S_{ML}(V, i, j, x)$. Note that *Best* stores the smallest matched element in this match. When the part 2 is finished, the pattern stored in $M_L(V, i, j, x)$ is the longest common permutation in $S_{ML}(V, i, j, x')$ with the smallest matched element in this match is $L(V, i, j, x')$. Thus the proof of this lemma is accomplished. ■

The prove of $M_R(V, i, j, x)$ could be generate correctly using our algorithm is very similar to lemma 1. The correctness of our algorithm is proved by the following theorem.

Theorem 3.2 *The algorithm described above outputs the longest common permutation correctly.*

Proof. From Lemma 3.1, we know that $M_L(V, i, j, x)$ and $M_R(V, i, j, x)$ could be generated correctly under the above algorithm. Thus, M_L (*root*, 1, *n*, *n*) or M_H (*root*, 1,*n*,1) will be the longest common permutation between permutation τ and of a separable permutation σ, where *root* is the root of separating tree T_σ of permutation σ. ■

Finally, the time and space complexity of our algorithm is analyzed in Theorem 3.3.

Theorem 3.3 *The algorithm solves the longest common permutation problem in $O(n^6)$ time using $O(n^5)$ space.*

Proof. The algorithm handles arrays M_L and M_R of size $O(n^5)$. Each cell contains pattern of length at most *n*. So that the total space complexity of this algorithms is $O(n^5)$. The total time need to fill M_L and M_R for all leaves are $O(n^4)$. Since there are *n* leaves, and computing the M_L and M_R values for each leaf requires $O(n)$ time for each *i*, *j* pair. Since there are $O(n)$ internal nodes, and computing M_L and M_R values for an internal node requires $O(n^3)$ for each *i*, *j* pair, then computing all M_L and M_R for all internal nodes need $O(n^6)$ time. Thus, the algorithm solves the longest common permutation problem in $O(n^6)$ time using $O(n^5)$ space. ■

5. Conclusion and further research

Bouvel *et al.* proposed the longest common permutation problem and give a $O(n^8)$ time algorithm to solve it in [2]. They ask whether there exists a $O(n^6)$ time algorithm for this problem as an open problem. In this paper, we give a $O(n^6)$ time algorithm to solve the longest common permutation problem, which improves the Bouvel's $O(n^8)$ time algorithm and answers to Bouvel's open problem. In the future, we will consider the longest common permutation in the classes of permutations that are less restricted than separable permutations, and generalize the problem of two permutations into the one of *k* permutations. On the other hand, the NP-complete result of the longest common permutation problem in permutations without restriction indicates the invention of approximation algorithm for this issue is needed. This problem is still open.

Reference

[1] P. Bose, J.F. Buss, and A. Lubiw, Pattern matching for permutations, Information Processing Letters 65 (1998), no. 5, 277–283.

[2] M. Bouvel and D. Rossin. The longest common pattern problem for two permutations. Pure Mathematics and Applications, 2007. to be published, arXiv:math.CO/0611679.

[3] D.G. Corneil, H. Lerchs, L. Stewart Burlingham, Complement-reducible graphs, Discrete Applied Math. 3, 163-174, 1981.

[4] L. Ibarra, Finding pattern matchings

for permutations, Information Processing Letters 61 (1997), 293–295.

[5] D.E. Knuth, The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 2nd edition, Addison-Wesley, 1973.

[6] L. Lovász, Combinatorial Problems and Exercises, North-Holland, 1979.

[7]D. Rotem, Stack-sortable permutations, Discrete Math, 33, 185-196, 1981.

[8] R. Simion and F.W. Schmidt, Restricted permutations, European J. Combinatorics 6, 383-405, 1985.

[9] J. West, Generating trees and the Catalan and Schröder numbers, Discrete Math.,146(1-3):247-262,1995.