The Longest Common Pattern Problem for two Permutations

Mathilde Bouvel * Dominique Rossin†

March 14, 2007

Abstract

In this paper, we give a polynomial ($\mathcal{O}(n^8)$) algorithm for finding a longest common pattern between two permutations of size n given that one is separable. We also give an algorithm for general permutations whose complexity depends on the length of the longest simple permutation involved in one of our permutations.

1 Introduction and basic concepts

The study of patterns in permutations has blossomed these last years: from a combinatorial point of view with the recent proof of the Stanley-Wilf conjecture by Marcus and Tardös, and from an algorithmic one with the development of algorithms for pattern involvement. Although the general pattern involvement problem is $NP-$hard, some polynomial solutions exist for special kinds of patterns like the separable ones [4, 10]. In this article we study the problem of finding a longest common pattern between two permutations σ_1 and σ_2 i.e. a permutation σ which is involved in both permutations σ_1 and σ_2. This is a generalization of the *pattern involvement problem* since finding if the longest pattern between σ_1 and σ_2 is equal to σ_1 is equivalent to the pattern involvement problem.

First, we give a polynomial algorithm based on the work of [4] for finding the longest common pattern if one permutation is separable. Then we generalize this algorithm for general permutations. The complexity of our algorithm is based on the length of the longest simple permutation involved in our permutations.

1.1 Permutations

A permutation σ of an interval I of $\mathbb{N}$ is a bijective map from I to itself. We denote by σ_i the image of i under σ. The permutation σ could either be seen as

*LIAFA, Université Paris 7 and CNRS, Paris, France, 75251, mbouvel@liafa.jussieu.fr

†LIAFA, Université Paris 7 and CNRS, Paris, France, 75251, rossin@liafa.jussieu.fr, `http://www.liafa.jussieu.fr/~rossin`

a function or a word $\sigma_i\sigma_{i+1}\ldots\sigma_j$, where $I = \{h : i \leq h \leq j\}$. For example the permutation $\sigma = 1\,4\,2\,5\,6\,3$ is the bijective function such that $\sigma(1) = 1$, $\sigma(2) = 4$, $\sigma(3) = 2$, $\sigma(4) = 5\ldots$. In the following, when we consider permutations without giving explicitly the interval I, we mean that $I = \{1, \ldots, n\}$ for some n.

Definition 1.1. A permutation $\pi = \pi_1 \ldots \pi_k$ is called a *pattern* of the permutation $\sigma = \sigma_{i+1} \ldots \sigma_{i+n}$ of $I = \{h : i+1 \leq h \leq i+n\}$, with $k \leq n$, if and only if there exist integers $i+1 \leq i_1 < i_2 < \ldots < i_k \leq i+n$ such that $\sigma_{i_\ell} < \sigma_{i_m}$ whenever $\pi_\ell < \pi_m$. We will also say that π is involved in σ or that σ contains π. The subsequence $\sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_k}$ is called an *occurrence* of π in σ.

A permutation σ that does not contain π as a pattern is said to *avoid* π.

Example 1.2. For example $\sigma = 1\ 4\ 2\ 5\ 6\ 3$ contains the pattern 1 3 4 2, and 1 5 6 3, 1 4 6 3, 2 5 6 3 and 1 4 5 3 are the occurrences of this pattern in σ. But σ avoids the pattern 3 2 1 as no subsequence of length 3 of σ is isomorphic to 3 2 1, i.e. is decreasing.

A number of enumerative results has been proved on classes of pattern avoiding permutations for patterns of length $3, 4$ and multiple patterns. More recently results about the algebricity of the generating function of general classes of permutations have been given [6, 1].

Another field of study of these permutations is from the point of view of pattern involvement. The problem of deciding if a permutation π is a pattern of a permutation σ is $NP-$complete but this problem is proved to be polynomial if the pattern is separable [4, 10].

Definition 1.3. 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.

Definition 1.4. A *binary separating tree* is a binary ordered tree with n leaves such that each internal vertex is labeled by $+$ or $-$.

For each such tree, there is a unique way [4] to decorate its leaves (considering them from left to right) by $\sigma_1, \sigma_2, \ldots, \sigma_n$ such that:

1. $\sigma_1 \ldots \sigma_n$ is a permutation of $\{1 \ldots n\}$.
2. Each node (internal or leaf) is decorated by a permutation of an interval.
3. Each internal node V labeled with $+$ (resp. $-$) is decorated by a permutation of $\{i \ldots j\}$. For some $h \in \{i+1, \ldots, j\}$, V's left child is decorated by a permutation of $\{i \ldots h-1\}$ (resp. $\{h \ldots j\}$) and its right child is decorated by a permutation of $\{h \ldots j\}$ (resp. $\{i \ldots h-1\}$), .

It is easy to prove [4] that to each separable permutation one can associate a binary separating tree (see Figure 1). Note that this tree is not uniquely defined as shown in Figure 1. However, one can associate a unique tree to each separable permutation by taking arbitrary ordered trees instead of binary ordered trees. These trees are the contraction of the binary ones by contracting every edge

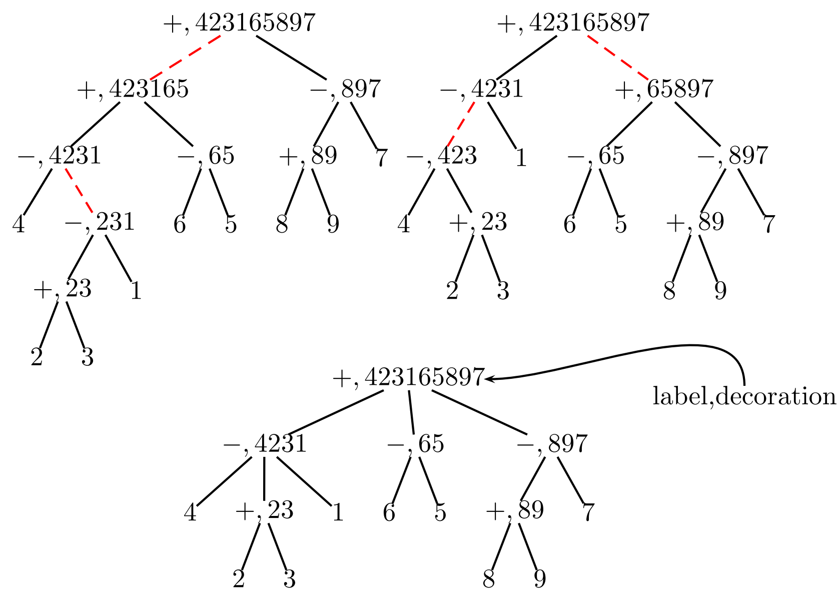


Figure 1: Two decorated binary separating trees and the decorated contracted separating tree of $\sigma = 4\ 2\ 3\ 1\ 6\ 5\ 8\ 9\ 7$. Dashed lines represent edges that should be contracted to transform binary into contracted separating tree.

between two nodes with the same label $+$ or $-$. In these trees, the signs of the internal nodes are thus ranked: if the root of the tree has label $+$ (resp. $-$), then every node at odd depth has label $+$ (resp. $-$) and every node at even depth label $-$ (resp. $+$), so that all labels are determined from the label of the root.

Definition 1.5. The unique *contracted separating tree* associated to a separable permutation σ is obtained from any binary separating tree of σ by contracting every edge between nodes of the same sign.

1.2 Modular decomposition of graphs ; Interval decomposition of permutations

The contracted separating trees we introduced in Definition 1.5 also appear in graph theory. Namely, those trees are a special case of interval decomposition trees (on which we however need to add a labeling). The interval decomposition trees are an equivalent for permutations of the modular decomposition trees for graphs [12, 9]. [1]

[1]Interval decomposition trees are known as common interval decomposition trees in the context of graph decomposition.

Before we come to the pattern matching problem in permutations, we need to introduce the interval decomposition trees, the labeled decomposition trees, and finally the expanded decomposition trees that are the key structure we use in our algorithms.

The *interval decomposition* of a permutation σ of size n is defined as follows. First consider all the *intervals* of σ that is to say all the subsequences $\sigma_j\sigma_{j+1}\ldots\sigma_k$ of consecutive entries of σ such that $\{\sigma_j, \sigma_{j+1}, \ldots, \sigma_k\}$ is an interval of $\mathbb{N}$. Among the intervals, the *strong intervals* are those that do not overlap any other interval[2]. Figure 2 illustrates the notion of intervals and strong intervals.

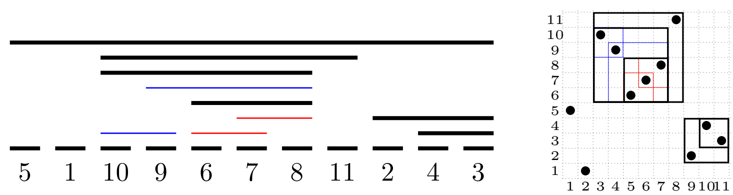


Figure 2: Interval decomposition of $\sigma = 5\ 1\ 10\ 9\ 6\ 7\ 8\ 11\ 2\ 4\ 3$.
In the first figure, intervals are represented by horizontal lines, with strong intervals corresponding to bold lines. In the second one, intervals are represented by squares (bold squares represent strong intervals).

The inclusion ordering yields a tree-like ordering on the set of strong intervals. This ordering is represented by a tree whose leaves are σ_1, σ_2, …, σ_n from left to right in this order, whose root is σ, and such that each internal node is the union of its children.

Note that there are two different types of internal nodes in the tree of strong intervals. For some nodes, say V, with k children $V_1, \ldots, V_k$ from left to right, there do not exist $(i, j) \neq (1, k)$ such that $1 \leq i < j \leq k$ and the union of $V_i, V_{i+1}, \ldots, V_j$ is an interval. These nodes are called *prime* nodes and are of type P.

The other nodes V are such that every union of consecutive children form an interval. Those node are called *linear* nodes and are of type L.

The tree along with the types P and L of the internal nodes (see Figure 3) is called the *interval decomposition tree* of σ. In this tree, the order of the children of a node depends on σ so that we have an ordered tree, unlike the modular decomposition trees for graphs.

Note that nodes of arity 2 verify both linear and prime definitions. We choose to consider them of type L. This choice will be explained later. For further explanation on these trees, like the proof that a node is either linear or prime, refer to [9].

In [3, 9] they prove the following result:

[2]The definition of overlapping intervals follows the intuition: we say that two intervals I and J are overlapping when $I \setminus J \neq \emptyset$, $J \setminus I \neq \emptyset$ and $I \cap J \neq \emptyset$

Theorem 1.6 ([3, 9])**.** *Computing interval decomposition tree from a permutation is linear.*

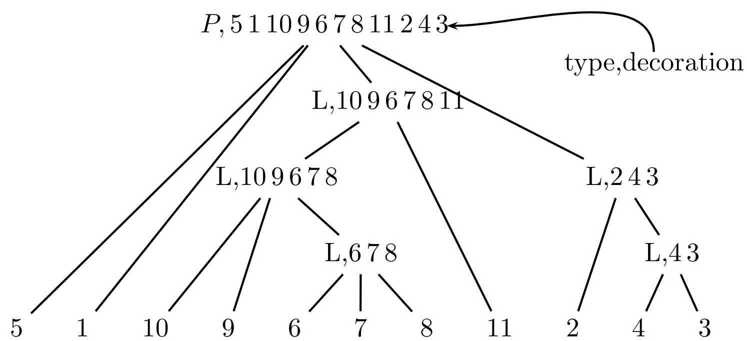


Figure 3: Interval decomposition tree for $\sigma = 5\,1\,10\,9\,6\,7\,8\,11\,2\,4\,3$

For our algorithmic use of interval decomposition trees, the types of the internal nodes are not sufficient and we need to label the internal nodes of the interval decomposition tree of a permutation.

It is easy to see that for each node V of type L, the intervals of values corresponding to the children of V are ordered either by increasing order or by decreasing order, when considering these children from left to right. The linear nodes are subsequently labeled $+$ or $-$ respectively.

The labeling of a prime node with d children consists of a permutation σ of size d that does not have any intervals except $\{1\}, \dots, \{d\}$ and σ. Such permutations are known as *simple permutations* [1, 2, 5, 6, 8]. The permutation σ represents the ordering of the children $V_1, \dots, V_d$ of V between them with respect to the values in the intervals corresponding to the V_i's. Namely, $\sigma_i < \sigma_j$ if and only if the interval corresponding to V_i contains values that are smaller than those contained in the interval corresponding to V_j. For example, the simple permutation labeling the root of the tree on Figure 3 is 3 1 4 2.

This interval decomposition tree along with labels $+$,$-$ and simple permutations can be computed easily in $\mathcal{O}(n^2 \ln n)$ which is sufficient for our purpose. Remind (Theorem 1.6) that the interval decomposition tree can be computed in linear time. Thus it remains to add a label on each internal node V. This can be done by sorting the intervals corresponding to the children of V.

When a interval decomposition tree is labeled, the intervals corresponding to the nodes can be deduced from this labeling, like in the case of binary separating trees (see Definition 1.4). Thus, the intervals can be seen as a *decoration* of the nodes and the same information is contained in the tree without these decorations.

Definition 1.7. The *labeled decomposition tree* of a permutation σ is the interval decomposition tree of σ, where we add the labeling on internal nodes described above, and where we forget the decoration.

Figure 4 gives the labeled decomposition tree of $\sigma = 5\ 1\ 10\ 9\ 6\ 7\ 8\ 11\ 2\ 4\ 3$.

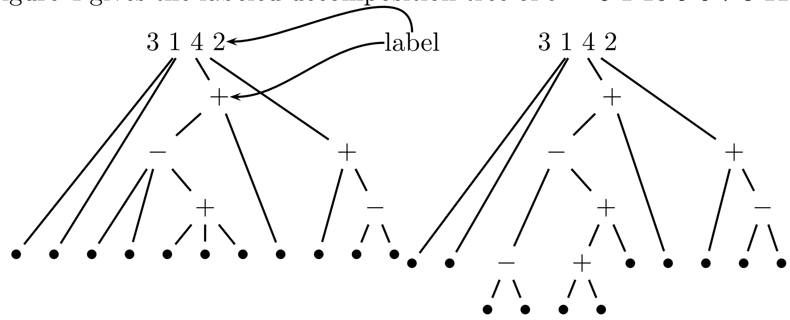


Figure 4: Labeled decomposition tree and expanded decomposition tree for $\sigma = 5\,1\,10\,9\,6\,7\,8\,11\,2\,4\,3$

We have the following nice characterization of separable permutations in terms of labeled decomposition trees:

Proposition 1.1. *The separable permutations are exactly those having a interval decomposition tree with no prime nodes.*
For any separable permutation, its contracted separating tree and its labeled decomposition tree are equal.

The proof of this proposition is straightforward from the definition of separable permutations. Note that it is important to give the type L to binary internal nodes for stating that proposition.

Proposition 1.1 states that labeled decomposition trees are a generalization to all permutations of the contracted separating tree defined only for separable permutations. For binary separating trees, we have an analogous generalization, called *expanded decomposition trees*. In our algorithms, it is easier to work on binary nodes so that the natural representation used later is the expanded decomposition tree.

To transform a labeled decomposition tree into an expanded decomposition tree, take each linear node V with children $V_1, \ldots, V_k$ and note that we can represent it by $(\ldots((V_1, V_2), V_3), \ldots), V_k)$ as shown in Figure 4. Each positive (resp. negative) internal node of arity k gives $k-1$ positive (resp. negative) binary internal nodes.

A consequence of Proposition 1.1 is:

Proposition 1.2. *The expanded decomposition tree of a separable permutation is one of its binary separating trees.*

2 Longest common pattern between two permutations, including one separable

In this section, we describe a polynomial time algorithm for finding a longest common pattern between two permutations σ and τ provided that σ is separable.

This algorithm uses the same technique as the one of Bose, Buss and Lubiw [4] for finding an occurrence of a separable pattern in a general permutation. Namely, it computes a binary separating tree T_σ of σ and uses it as a guide in the search of the longest common pattern with τ. To avoid a complexity blow-up, the key point is to use dynamic programming, the initial problem being decomposed into sub-problems according to the structure of T_σ.

First recall that it can be decided in linear time whether a permutation is separable or not. If it is, its binary separating tree can also be computed in linear time, as described in [4]. Indeed it can be computed by reading the permutation from left to right. This result is actually a special case of a more general one, proved in [3, 9], and stating that the interval decomposition tree of any permutation can be computed in linear time 1.6.

Instead of two permutations σ and τ, our algorithm takes as an input a binary separating tree T_σ for a separable permutation σ of size k, and a permutation τ in the usual representation $\tau_1\tau_2\ldots\tau_n$. Notice that a binary separating tree for σ has $\mathcal{O}(k)$ nodes.

More precisely, the algorithm fills in the array

$$M = \{M(V,i,j,a,b) : V \text{ a node of } T_\sigma, 1 \le i \le j \le n, 1 \le a \le b \le n\}.$$

For any node V in T_σ, let us denote by $\sigma(V)$ the subpermutation of σ corresponding to the subtree of T_σ rooted at V. With the notations of Section 1, $\sigma(V)$ is the permutation decorating the node V. The cell $M(V,i,j,a,b)$ of the array M contains a longest common pattern π between $\sigma(V)$ on one hand, and the subpermutation $\tau_i\ldots\tau_j$ of τ on the other hand, with the additional restriction that the occurrence of π in $\tau_i\ldots\tau_j$ uses only entries of τ whose values are between a and b. The empty pattern, of size 0, will be denoted ϵ.

Example 2.1. If the node V represents the pattern 2 1 (i.e. $\sigma(V) = (i+1)\ i$), and $\tau = 6\ 4\ 2\ 5\ 3\ 1$ then we have for instance $M(V,2,4,3,5) = 1$, $M(V,2,5,3,4) = 2\ 1$ and $M(V,4,5,1,2) = \epsilon$.

The algorithm works as follows. To start the computation, we fill in the subarrays $M(V,\_,\_,\_,\_)$ for all the leaves V of T_σ. Then, we compute $M(V,\_,\_,\_,\_)$ for any internal node V using the subarrays $M(V_L,\_,\_,\_,\_)$ and $M(V_R,\_,\_,\_,\_)$ corresponding to the left child (V_L) and the right child (V_R) of V. In order to combine the patterns found in $M(V_L,\_,\_,\_,\_)$ and $M(V_R,\_,\_,\_,\_)$ with the intention of filling $M(V,\_,\_,\_,\_)$, we need a definition of pattern concatenation.

Definition 2.2. Consider two patterns π and π' of respective lengths k and k'. The positive and negative concatenations of π and π' are defined respectively by: $\pi \oplus \pi' = \pi_1 \cdots \pi_k(\pi'_1+k)\cdots(\pi'_{k'}+k)$ and $\pi \ominus \pi' = (\pi_1+k')\cdots(\pi_k+k')\pi'_1\cdots\pi'_{k'}$

Example 2.3.

$4\ 3\ 5\ 2\ 1 \oplus 3\ 1\ 4\ 2 = 4\ 3\ 5\ 2\ 1 \,\vdots\, 8\ 6\ 9\ 7$

$4\ 3\ 5\ 2\ 1 \ominus 3\ 1\ 4\ 2 = 8\ 7\ 9\ 6\ 5 \,\vdots\, 3\ 1\ 4\ 2$

It is clear from Definition 2.2 that the positive (resp. negative) concatenation of two patterns produces again a pattern.

The detailed dynamic programming algorithm is given in Algorithm 1.

Algorithm 1 Longest common pattern between two permutations, first one separable

---

INPUT: A binary separating tree T_σ of a separable permutation σ of size k and a permutation τ of size n

CREATE AN ARRAY M:
for any node V of T_σ and any integers i, j, a and b between 1 and n **do**
    $M(V, i, j, a, b) \leftarrow \epsilon$
end for

FILL IN M FOR THE LEAVES OF T_σ:
for any leaf V of T_σ **do**
    for any integers i, j, a and b between 1 and n, $i \leq j$, $a \leq b$ **do**
        if there exists some $h \in \{i, i+1, \ldots, j\}$ such that $a \leq \tau_h \leq b$ **then**
            $M(V, i, j, a, b) \leftarrow 1$
        end if
    end for
end for

FILL IN THE REST OF M:
for any internal node V of T_σ, considering the nodes in the postfix ordering **do**
    if V is a positive node **then**
        for any integers i, j, a and b between 1 and n, $i \leq j$, $a \leq b$ **do**
            $M(V, i, j, a, b) \leftarrow Longest(\{M(V_L, i, h-1, a, c-1) \oplus M(V_R, h, j, c, b) : i \leq h \leq j+1, a \leq c \leq b+1\})$
        end for
    else
        /* V is a negative node /*
        for any integers i, j, a and b between 1 and n, $i \leq j$, $a \leq b$ **do**
            $M(V, i, j, a, b) \leftarrow Longest(\{M(V_L, i, h-1, c, b) \ominus M(V_R, h, j, a, c-1) : i \leq h \leq j+1, a \leq c \leq b+1\})$
        end for
    end if
end for

OUTPUT: $M(\text{root of } T_\sigma, 1, n, 1, n)$

In Algorithm 1, for any set S of patterns, *Longest*(S) returns a longest pattern in the set S.

Proposition 2.1. *Algorithm 1 is correct: it outputs a longest common pattern between the two permutations σ and τ given in input.*

Proof. The proof is by induction.
We show that the algorithm stores in $M(V, i, j, a, b)$ a longest common pattern between $\sigma(V)$ and $\tau_i \dots \tau_j$ whose occurrence in $\tau_i \dots \tau_j$ uses only values between a and b. We call $\mathcal{P}$ this property.

If V is a leaf, the above statement is clearly true.

If V is an internal node, with two children V_L (its left child) and V_R (its right child), then let i, j, a and b be integers such that $1 \leq i \leq j \leq n$ and $1 \leq a \leq b \leq n$. We assume in the rest of the proof that V is a positive node, the case of a negative node being very similar. To begin with, it is easy to see that $M(V, i, j, a, b)$ contains a common pattern between $\sigma(V)$ and $\tau_i \dots \tau_j$ using only values between a and b in $\tau_i \dots \tau_j$. Indeed, by induction hypothesis, we infer that every pattern in the set $S = \{M(V_L, i, h-1, a, c-1) \oplus M(V_R, h, j, c, b) : i \leq h \leq j+1, a \leq c \leq b+1\}$, and *a fortiori Longest*(S), is a common pattern between $\sigma(V)$ and $\tau_i \dots \tau_j$ using only values between a and b in $\tau_i \dots \tau_j$ (see Example 2.5).

To conclude the inductive step of the proof of Proposition 2.1, it remains to show that:

Lemma 2.4. *Longest*(S) *is of maximal length among all the common patterns between $\sigma(V)$ and $\tau_i \dots \tau_j$ using only values between a and b in $\tau_i \dots \tau_j$.*

Proof. First recall that we are proving Property $\mathcal{P}$ for a node V, and that we can use Property $\mathcal{P}$ on nodes V_L and V_R by induction hypothesis.

Now, let us denote by π a longest common pattern between $\sigma(V)$ and $\tau_i \dots \tau_j$, using only values between a and b in $\tau_i \dots \tau_j$.

As shown on Figure 5, there exist integers $h \in \{i, \dots, j+1\}$ and $c \in \{a, \dots, b+1\}$ such that π is decomposed into $\pi = \pi_1 \oplus \pi_2$, with π_1 a common pattern between $\sigma(V_L)$ and $\tau_i \dots \tau_{h-1}$, using only values between a and $c-1$ in $\tau_i \dots \tau_{h-1}$, and π_2 a common pattern between $\sigma(V_R)$ and $\tau_h \dots \tau_j$, using only values between c and b in $\tau_h \dots \tau_j$.

Notice that in this decomposition π_1 or π_2 might be the empty pattern.

It can be easily seen that π_1 (resp. π_2) is a longest common pattern between $\sigma(V_L)$ (resp. $\sigma(V_R)$) and τ in the given intervals of indices and values. Indeed, if π_1 (resp. π_2) were not a pattern of maximal length for the given intervals of indices and values, then π would not be of maximal length either, contradicting the definition of π. Consequently, by induction hypothesis, $|M(V_L, i, h-1, a, c-1)| = |\pi_1|$ and $|M(V_R, h, j, c, b)| = |\pi_2|$. The pattern stored in $M(V, i, j, a, b)$ by the algorithm is of length at least $|M(V_L, i, h-1, a, c-1) \oplus M(V_R, h, j, c, b)| = |\pi_1 \oplus \pi_2| = |\pi|$. As π is of maximal length by assumption, we conclude that $M(V, i, j, a, b)$ is also of maximal length. □

Finally, $M(V, i, j, a, b)$ contains a longest common pattern between $\sigma(V)$ and $\tau_i \dots \tau_j$ whose occurrence in $\tau_i \dots \tau_j$ uses only values between a and b.

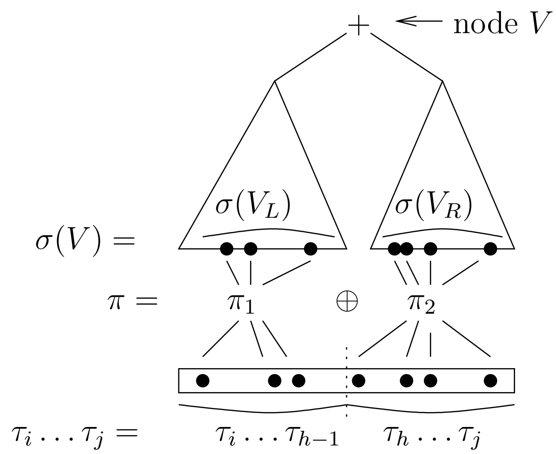
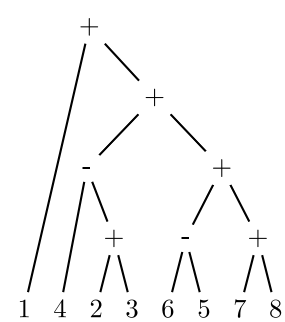


Figure 5: Proof of Lemma 2.4

When V is a negative node, we decompose π into $\pi_1 \ominus \pi_2$, with π_1 a common pattern between $\sigma(V_L)$ and $\tau_i \dots \tau_{h-1}$, using only values between c and b in $\tau_i \dots \tau_{h-1}$, and π_2 a common pattern between $\sigma(V_R)$ and $\tau_h \dots \tau_j$, using only values between a and $c-1$ in $\tau_h \dots \tau_j$, and the proof follows the same steps. ☐

Example 2.5. Consider the permutations $\sigma = 1\ 4\ 2\ 3\ 6\ 5\ 7\ 8$ and $\tau = 4\ 1\ 3\ 2\ 5\ 6\ 8\ 9\ 7$. A separating tree T_σ of σ is represented on the right. For example, we can choose V to be the right child of the root of T_σ. Then we have $\sigma(V) = 4\ 2\ 3\ 6\ 5\ 7\ 8$, $\sigma(V_L) = 4\ 2\ 3$ and $\sigma(V_R) = 6\ 5\ 7\ 8$. Now choose $i = 2, j = 7, a = 2$ and $b = 8$. We want to show that for any $h \in \{i, \dots j+1\}$ and $c \in \{a, \dots b+1\}$, $M(V_L, i, h-1, a, c-1) \oplus M(V_R, h, j, c, b)$ is a common pattern between $\sigma(V)$ and $\tau_i \dots \tau_j$ using only values between a and b in $\tau_i \dots \tau_j$. Take for example $h = 5$ and $c = 4$.



By induction hypothesis, $M(V_L, i, h-1, a, c-1)$ contains a longest common pattern between $\sigma(V_L)$ and $\tau_i \dots \tau_{h-1}$ using only values between a and $c-1$ in $\tau_i \dots \tau_{h-1}$. Here, $M(V_L, i, h-1, a, c-1) = 2\ 1$, an occurrence of $2\ 1$ in $\tau_i \dots \tau_{h-1} = 1\ 3\ 2$ using values between 2 and 3 being 1 **3 2**, and an occurrence of 2 1 in $\sigma(V_L) = 4\ 2\ 3$ being **4 2** 3. Similarly, we have $M(V_R, h, j, c, b) = M(V_R, 5, 7, 4, 8) = 1\ 2\ 3$, as shown by the occurrences **5 6 8** in $\tau_h \dots \tau_j = 5\ 6\ 8$ and **6** 5 **7 8** in $\sigma(V_R) = 6\ 5\ 7\ 8$. The occurrence of $M(V_L, i, h-1, a, c-1) \oplus M(V_R, h, j, c, b) = 2\ 1\ 3\ 4\ 5$ in $\tau_i \dots \tau_j$ using values between a and b is thus obtained by considering simultaneously the two occurrences in τ selected before. Namely, an occurrence of 2 1 3 4 5 in $\tau_i \dots \tau_j = 1\ 3\ 2\ 5\ 6\ 8$ using values between 2 and 8 is 1 **3 2 5 6 8**. Notice also that the occurrence **4 2** 3 **6** 5 **7 8** of

2 1 3 4 5 in $\sigma(V) = 4\ 2\ 3\ 6\ 5\ 7\ 8$ is again obtained considering simultaneously the occurrences of 2 1 and 1 2 3 in $\sigma(V_L)$ and $\sigma(V_R)$ respectively.

Proposition 2.2. *Algorithm 1 has a time complexity in $\mathcal{O}(min(k, n)kn^6)$.*

Proof. Algorithm 1 handles an array M of size $\mathcal{O}(kn^4)$, where each cell contains a pattern of length at most $min(n, k)$, so that the total space complexity is $\mathcal{O}(min(n, k)kn^4)$. For filling in the subarrays $M(V, \_, \_, \_, \_)$ for all the leaves V of T_σ (line 6 to 13 of Algorithm 1), the total time complexity is $\mathcal{O}(kn^5)$. And for any internal node V, filling in one entry of the subarray $M(V, \_, \_, \_, \_)$ costs $\mathcal{O}(min(k, n)n^2)$, since at line 18 (or 23) of Algorithm 1, we search for an element of maximal length among $\mathcal{O}(n^2)$ elements, each of size $\mathcal{O}(min(k, n))$. Consequently, completely filling in this subarray $M(V, \_, \_, \_, \_)$ requires a time complexity $\mathcal{O}(min(k, n)n^6)$. Since there are $\mathcal{O}(k)$ internal nodes in T_σ, we have the announced result. □

This complexity can be improved to $\mathcal{O}(kn^6)$, by storing an integer, a label ($\oplus$ or $\ominus$), and two pointers in $M(V, i, j, a, b)$ (when V is an internal node) instead of a pattern. Namely, if Algorithm 1 fills in $M(V, i, j, a, b)$ with the pattern $\rho = M(V_L, i, h-1, a, c-1) \oplus M(V_R, h, j, c, b)$, it is sufficient to store in $M(V, i, j, a, b)$ the length of ρ, the label $\oplus$, and two pointers pointing to the entries $M(V_L, i, h-1, a, c-1)$ and $M(V_R, h, j, c, b)$ of the array M. At the end of the algorithm, this system of pointers gives a binary separating tree of a longest common pattern π between σ and τ. From this tree, π can be computed in linear time [4].

A consequence of Properties 2.1 and 2.2 is:

Theorem 2.6. *The problem of finding a longest common pattern between two permutations, one being separable, is in P.*

3 Longest common pattern between two permutations

The result of Theorem 2.6 can be easily extended to classes of permutations that are less restricted than separable permutations. Using the interval decomposition tree introduced in Section 1.2, we will see that a longest common pattern between two permutations σ and τ can be computed in polynomial time as soon as the arity of any prime node in the interval decomposition tree of σ is bounded by a constant d chosen independently.

Proposition 1.2 states that expanded decomposition trees are a generalization to all permutations of binary separating trees. From this remark, it becomes natural to try and use expanded decomposition trees in an algorithm for finding a longest common pattern between two general permutations. In the following, we describe such an algorithm and analyze its complexity: it is not in general a polynomial time algorithm, but the complexity analysis reveals classes of permutations for which the algorithm runs in polynomial time.

First, we notice that the expanded decomposition tree of a permutation can be computed in polynomial time, and even in linear time if we have an independent bound on the arity of the prime nodes. Theorem 1.6 provides a $\mathcal{O}(n)$-time algorithm for computing the interval decomposition tree T_σ of any permutation σ of size n. This tree can easily be labeled in time $\mathcal{O}(nd \log d)$ where d is a bound on the arity of prime nodes: for every internal node V of T_σ – of which there are $\mathcal{O}(n)$ – it is enough to sort its children to find the label of V. Finally, the vertical expansion necessary to obtain the expanded decomposition tree of σ requires again a linear time.

Algorithm 2 takes as an input an expanded decomposition tree T_σ of a permutation σ and a permutation τ. It outputs a longest common pattern between σ and τ. It works just like Algorithm 1, except for the case of prime nodes in T_σ. The procedure in this additional case is described in Algorithm 2.

Algorithm 2 Longest common pattern between two permutations

INPUT : An expanded decomposition tree T_σ of a permutation σ of size k and a permutation τ of size n

CREATE AN ARRAY M: proceed as in Algorithm 1

FILL IN M FOR THE LEAVES OF T_σ: proceed as in Algorithm 1

FILL IN THE REST OF M:

for any internal node V of T_σ, considering the nodes in the postfix ordering **do**
  if V is a positive or a negative node **then**
    proceed as in Algorithm 1
  else
    /* V is a prime node /*
    Let ρ be the simple permutation labeling V
    Let d be the arity of V, and $V_1, \dots, V_d$ the children of V, from left to right
    for any integers i, j, a and b between 1 and n, $i \le j$, $a \le b$ **do**
      $M(V, i, j, a, b) \leftarrow \mathit{Longest}(S)$ where
      $$S = \Big\{ \odot_\rho \left(\left(M(V_k, h_{k-1}, h_k - 1, c_{\rho_k - 1}, c_{\rho_k} - 1) \right)_{1 \le k \le d} \right) : i = h_0 \le h_1 \le \ldots \le h_d = j+1, a = c_0 \le c_1 \le \ldots \le c_d = b+1 \Big\}$$
    end for
  end if
end for

OUTPUT : $M(\text{root of } T_\sigma, 1, n, 1, n)$

Algorithm 2 uses a more general kind of pattern concatenation than just $\oplus$ and $\ominus$. The ρ-concatenation, or pattern concatenation according to ρ, is defined as follows:

Definition 3.1. Given $\rho = \rho_1 \dots \rho_n$ a permutation of size n, and n patterns $\pi^1, \dots, \pi^n$ of respective size $k_1, \dots, k_n$, the ρ-concatenation of the $(\pi^i)_{1 \le i \le n}$ is

$$\odot_\rho(\pi^1, \dots, \pi^n) = \text{shift}(\pi^1, \rho_1) \dots \text{shift}(\pi^n, \rho_n) \text{ where}$$

$$\text{shift}(\pi^i, \rho_i) = \text{shift}(\pi^i, \rho_i)(1) \dots \text{shift}(\pi^i, \rho_i)(k_i) \text{ and}$$

$$\text{shift}(\pi^i, \rho_i)(x) = (\pi^i(x) + k_{\rho^{-1}(1)} + \dots + k_{\rho^{-1}(i-1)}) \text{ for all } x \text{ between 1 and } k_i$$

Example 3.2.

$\odot_{25314}(21, 312, 4321, 12, 231) = 4\ 3 \vdots 14\ 12\ 13 \vdots 8\ 7\ 6\ 5 \vdots 1\ 2 \vdots 10\ 11\ 9.$

We can also notice that $\odot_{12} = \oplus$ and $\odot_{21} = \ominus$.

The idea behind Algorithm 2 is quite simple. When filling in $M(V, i, j, a, b)$ for a prime node V labeled by ρ and having d children $V_1 \dots V_d$, we "slice" the intervals $\{i, \dots, j\}$ and $\{a, \dots, b\}$ into $I_1 \dots I_d$ and $A_1 \dots A_d$ respectively, such that $I_p \prec I_k$ and $A_p \prec A_k$ whenever $p < k$ (By $A \prec B$, we mean that $\forall a \in A, \forall b \in B, a < b$.). Then we ρ-concatenate longest common patterns between the $\sigma(V_k)$ and τ in the intervals I_k of indices and A_{ρ_k} of values. With the notation of Algorithm 2, $I_k = \{h_{k-1}, \dots, h_k - 1\}$ and $A_k = \{c_{k-1}, \dots, c_k - 1\}$.

Proposition 3.1. *Algorithm 2 is correct: it outputs a longest common pattern between the two permutations σ and τ given in input.*

Proof. Similar to the proof of Proposition 2.1.

With the notations of the proof of Proposition 2.1, in the case of a prime node V labeled by ρ, with children $V_1 \dots V_d$, there exist integers $i = h_0 \le h_1 \le \dots \le h_d = j + 1$ and $a = c_0 \le c_1 \le \dots \le c_d = b + 1$, such that we can decompose π into $\pi = \odot_\rho(\pi^1, \dots, \pi^d)$, with π^k a common pattern between $\sigma(V_k)$ and $\tau_{h_{k-1}} \dots \tau_{h_k - 1}$, using only values between $c_{\rho_k - 1}$ and $c_{\rho_k} - 1$ in $\tau_{h_{k-1}} \dots \tau_{h_k - 1}$. Using this decomposition of π, we can use the induction hypothesis on the nodes $(V_k)_{1 \le k \le d}$ and finish the proof as before. ☐

In this proof, the trick relies on the fact that a common pattern between $\sigma(V)$ and $\tau_i \dots \tau_j$ is always a concatenation of common patterns between the children of V and "slices" of $\tau_i \dots \tau_j$. This stability when going from parents to children in the expanded decomposition tree also appears in a paper of Albert and Atkinson [1], for example in their Lemma 15.

The main difference between Algorithms 1 and 2 lies in the complexity analysis. Those two algorithms deal with dynamic programming arrays of the same size, but the cost for computing one entry can be much greater in Algorithm 2 than in Algorithm 1. Indeed, for any internal node V, in order to fill in one entry of $M(V, \_, \_, \_, \_)$, Algorithm 1 computes a longest pattern in a set containing $\mathcal{O}(n^2)$ elements, whereas in Algorithm 2, the set from which we have to extract a longest pattern contains $\mathcal{O}(n^{2d-2})$ elements, if d is the arity of V (see line 13 of Algorithm 2).

With no hypothesis on a permutation σ of size k, the only bound we can give on the maximal arity d of a prime node in the expanded decomposition tree of σ is $d \leq k$. This bound is optimal since the equality $d = k$ is achieved when σ is a simple permutation. The total time complexity of Algorithm 2 is consequently $\mathcal{O}(min(n,k)kn^{2k+2})$, and it is not polynomial. However, if we consider classes of permutations such that the arity of any prime node in their expanded decomposition tree is bounded by a constant d, then Algorithm 2 has a polynomial time complexity $\mathcal{O}(min(n,k)kn^{2d+2})$. For example, any class containing finitely many simple permutations satisfy this condition.

This can be summarized in the following theorem:

Theorem 3.3. *Let d be a integer. Consider the class R of permutations having an expanded decomposition tree with all prime nodes of arity smaller than d. Then the problem of finding a longest common pattern between a permutation in R and another unrestricted permutation is in P.*

4 Conclusion and open problems

We generalize the algorithm given in [4] for the *longest common pattern* problem. Yet our algorithm seems far from optimal. For example, for separable permutations, our work is based on [4]'s $\mathcal{O}(n^7)$ pattern involvement algorithm, but Ibarra [10] give a faster $\mathcal{O}(n^5)$ algorithm for the pattern involvement problem. Could this algorithm be adapted to the *longest common pattern* problem? Yet a lower bound is given by the edit distance problem [13] as the edit distance problem between two trees is a special case of the longest common pattern problem as shown in [11].

Acknowledgment

The authors thank the referee for his useful comments and improvements.

References

[1] M.H. Albert, M.D. Atkinson, Simple Permutations and Pattern restricted Permutations, Discrete Mathematics 300 (2005) 1-15

[2] M.H. Albert, M.D. Atkinson, M. Klazar, The enumeration of simple permutations, Journal of Integer Sequences, Vol. 6 (2003)

[3] A. Bergeron, C. Chauve, F. de Montgolfier, M. Raffinot, Computing Common Intervals of K Permutations, with Applications to Modular Decomposition of Graphs, ESA 2005, 779-790

[4] P. Bose, J. F. Buss, A. Lubiw, Pattern matching for permutations, Information Processing Letters 65 (1998), 277-283

[5] R. Brignall, S. Huczynska, V. Vatter, Decomposing Simple Permutations with Enumerative Consequences, arXiv:math.CO/0606186

[6] R. Brignall, S. Huczynska, V. Vatter, Simple permutations and algebraic generating functions, arXiv:math:CO/0608391

[7] R. Brignall, N. Ruškuc, V.Vatter, Simple permutations: decidability and unavoidable substructures, arXiv:math.CO/0609211

[8] S. Corteel, G. Louchard, R. Pemantle, Common intervals of permutations, Discrete Math. Theor. Comput. Sci. 8, 1 (2006), 189-214

[9] B.-M. Bui Xuan, M. Habib, C. Paul, Revisiting T. Uno and M. Yagiura's Algorithm, ISAAC 2005, 146-155

[10] L. Ibarra, Finding pattern matchings for permutations, Information Processing Letters 61 (1997) 293-295

[11] A. Micheli, D. Rossin, Average edit distance between ordered trees, RAIRO TIA, (2006)

[12] F. de Montgolfier, Décomposition modulaire des graphes : théories extensions et algorithmes, PhD Thesis, (2003)

[13] K. Zhang, D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, SIAM Journ. of Comput. 18 (6) 1245, (1989)