Pattern Matching for Separable Permutations

Both Emerite Neou, Romeo Rizzi, Stéphane Vialette

Pattern Matching for Separable Permutations

Both Emerite Neou*[1] Romeo Rizzi[2] and Stéphane Vialette[1]

[1] Université Paris-Est, LIGM (UMR 8049), CNRS, UPEM, ESIEE Paris, ENPC, F-77454, Marne-la-Valle, France
{neou,vialette}@univ-mlv.fr
[2] Department of Computer Science, Universit degli Studi di Verona, Italy
romeo.rizzi@univr.it

Abstract. Given a permutation π (called the *text*) of size n and another permutation σ (called the *pattern*) of size k, the **NP**-complete pattern containment problem asks whether σ is contained in π as an order-isomorphic subsequence. In this paper, we focus on separable permutations (those permutations that avoid both 2413 and 3142, or, equivalently, that admit a *separating tree*). The main contributions presented in this paper are as follows.

- We simplify the algorithm of Ibarra to detect an occurrence of a separable permutation in a permutation and show how to reduce the space complexity from $O(n^3k)$ to $O(n^3 \log k)$.
- In case both the text and the pattern are separable permutations, we give a more practicable alternative $O(n^2k)$ time and $O(nk)$ space algorithm. Furtheremore, we show how to use this approach to decide in $O(nk^3\ell^2)$ time whether a separable permutation is a disjoint union of two given permutations of size k and ℓ
- We give a $O(n^6k)$ time and $O(n^4 \log k)$ space algorithm to compute the longest common pattern of two permutations of size at most n (provided that at least one of these permutations is separable). This improves upon the existing $O(n^8)$ time algorithm.
- Finally, we give a $O(n^6k)$ time and $O(kn^4)$ space algorithm to detect an occurrence of a bivincular separable permutation in a permutation. (Bivincular patterns generalize classical permutations by requiring that positions and values involved in an occurrence may be forced to be adjacent).

1 Introduction

A permutation π is said to contain another permutation σ, in symbols $\sigma \preceq \pi$, if there exists a subsequence of entries of π that has the same relative order as σ, and in this case σ is said to be a *pattern* of π. Otherwise, π is said to *avoid* the permutation σ. During the last decade, the study of the pattern containment on

* On a Co-tutelle Agreement with the Department of Mathematics of the University of Trento

permutations has become a very active area of research and an annual conference (PERMUTATION PATTERN) is devoted to this subject and a database[1] of permutation pattern avoidance is maintained by Bridget Tenner.

We consider here the so-called *pattern containment* problem (also sometimes referred to as the *pattern involvement* problem): Given two permutations σ and π, this problem is to decide whether $\sigma \preceq \pi$ (the problem is ascribed to Wilf in [5]). The permutation containment problem is **NP**-hard [5]. It is, however, polynomial time solvable by brute-force enumeration if σ has bounded size. Improvements to this algorithm were presented in [2] and [1], the latter describing a nice $O(|\pi|^{0.47k+o(|\sigma|)})$ time algorithm. Bruner and Lackner [7] gave a fixed-parameter algorithm solving the pattern containment problem with an exponential worst-case runtime of $O(1.79^{\mathrm{run}(\pi)})$, where $\mathrm{run}(\pi)$ denotes the number of alternating runs of π. Of particular importance, it has been proved in [10] that the pattern containment problem is fixed-parameter tractable for parameter $|\sigma|$.

A few particular cases of the pattern containment problem have been attacked successfully. Of particular interest in our context, the pattern containment problem is solvable in polynomial time for separable patterns. Separable permutations are those permutations that contain neither 2413 nor 3142, and they are enumerated by the Schröder numbers (sequence A006318 in OEIS). The pattern containment problem is solvable in $O(kn^4)$ time and $O(kn^3)$ space for separable patterns [11] (see also [5]), where k is the length of the pattern and n is the length of the target permutation. Notice that there are numerous characterizations of separable permutations. To mention just a few examples, they are the permutations whose permutation graphs are cographs (*i.e.* P_4-free graphs); equivalently, a separable permutation is a permutation that can be obtained from the trivial permutation 1 by *direct sums* and *skew sums* [15]. While the term separable permutation dates only to the work of Bose, Buss, and Lubiw [5], these permutations first arose in Avis and Newborns work on pop stacks [3].

There exist many generalisations of patterns that are worth considering in the context of algorithmic issues in pattern involvement (see [13] for an up-to-date survey). *Vincular* patterns, also called *generalized* patterns, resemble (classical) patterns, with the constraint that some of the letters in an occurrence must be consecutive. Of particular importance in our context, Bruner and Lackner [7] proved that deciding whether a vincular pattern σ of length k occurs in a permutation π of length n is $W[1]$-complete for parameter k. *Bivincular* patterns generalize classical patterns even further than vincular patterns. Indeed, in bivincular patterns, not only positions but also values of elements involved in a occurrence may be forced to be adjacent

The paper is organised as follows. Section 2 is devoted to presenting the needed material. In Section 3, we revisit the polynomial-time algorithm of Ibarra [11] and we propose a simpler dynamic programming approach, and in Section 4 we focus on the case where both the pattern and the target permutation are separable. Section 5 is concerned with presenting an algorithm to test whether a separable permutation is the disjoint union of two given (necessarily separa-

[1] http://math.depaul.edu/bridget/patterns.html

ble) permutations. In Section 6, we revisit the classical problem of computing a longest common separable pattern as introduced by Rossin and Bouvel [14] and propose a slightly faster - yet still not practicable - algorithm. Finally, in Section 7, we prove that the pattern matching problem is polynomial-time solvable for vincular separable patterns. To the best of our knowledge, this is the first time the pattern matching problem is proved to be tractable for a generalization of separable patterns.

2 Definitions

A *permutation* of length n is a one-to-one function from an n-element set to itself. We write permutations as words $\pi = \pi_1\,\pi_2\,\ldots\,\pi_n$, whose letters are distinct and usually consist of the integers $1\,2\,\ldots\,n$. We also designate its i-th element by $\pi[i]$. We let S_n denote the set of all permutations of length n. We shall also represent a permutation π by its *diagram* consisting in the set of points at coordinates $(i, \pi[i])$ drawn in the plane. According to this representation, we say that an element $\pi[i]$ is on the *left* (resp. *right*) of another element $\pi[j]$ if $i < j$ (resp. $i > j$). Furthermore, we say that an element $\pi[i]$ is *above* (resp. *below*) another element $\pi[j]$ if $\pi[j] < \pi[i]$ (resp. $\pi[i] < \pi[j]$).

The *reduced form* of a permutation π on a set $\{j_1, j_2, \ldots, j_k\}$ where $j_1 < j_2 < \ldots < j_k$, is the permutation π' obtained by renaming the letters of π so that j_i is renamed i for all $1 \leq i \leq k$. We let $\mathrm{red}(\pi)$ denote the reduced form of π. For example $\mathrm{red}(5826) = 2413$. If $\mathrm{red}(u) = \mathrm{red}(w)$, we say that u and w are *order-isomorphic*.

If $\pi = \pi_1\pi_2\ldots\pi_n$ is a permutation of S_n, for any $i, j \in [n]$ with $i \leq j$, we let $\pi[i, j]$ stand for the sequence $\pi_i\pi_{i+1}\ldots\pi_j$. A permutation $\sigma \in S_k$ is said to occur within a permutation $\pi \in S_n$, in symbols $\sigma \preceq \pi$, if there is some k-tuple $1 \leq i_1 \leq i_2 \leq \ldots \leq i_k$ such that $\mathrm{red}(\pi_{i_1}\pi_{i_2}\ldots\pi_{i_k}) = \sigma$ (*i.e.*, σ has a subsequence of length k that is order-isomorphic to π). The subsequence $\pi_{i_1}\pi_{i_2}\ldots\pi_{i_k}$ is called an *occurrence* of σ in π. If π does not contain σ, then π is said to avoid σ. For example, the permutation $\pi = 391867452$ contains the pattern $\sigma = 51342$ as can be seen in the highlighted subsequence of $\pi = 3\mathbf{91867}4\mathbf{5}2$, but it avoids 1234 since π contains no increasing subsequence of length four.

For two permutations π_1 of size n_1 and π_2 of size n_2, the *direct sum* of π_1 and π_2 is defined by $\pi_1 \oplus \pi_2 = \pi_1[1]\pi_1[2]\ldots\pi_1[n_1](\pi_2[1]+n_1)(\pi_2[2]+n_1)\ldots(\pi_2[n_2]+n_1)$ [15]. The direct sum operation consists to putting the elements of π_2 on the right and above the elements of π_1. See Figure 2 for an example of a direct sum. Similarly, we define the *skew sum* of π_1 and π_2 by $\pi_1 \ominus \pi_2 = (\pi_1[1]+n_2)(\pi_1[2]+n_2)\ldots(\pi_1[n_1]+n_2)\pi_2[1]\pi_2[2]\ldots\pi_2[n_2]$ [15]. The skew sum operation consists to putting the elements of π_1 on the left and above the elements of π_2. See Figure 3 for an example of a skew sum. It is easy to see whenever a permutation can be decomposed into a direct sum (respt. a skew sum) of two permutation: In the plot of the permutation one can draw a cross that split the permutation into 4 sections, so that the top-left and the bottom-right sections are empty (the top-right and the bottom-left are empty).

Whenever we consider a subsequence of elements of a permutation by selecting the elements by bounding the elements by value and/or by position, we call this subsequence a rectangle. On a plot of a permutation a simple way to see all the elements of a rectangle, is to draw a rectangle where each edge is define by one bound on the plot, every element which is in the rectangle is part of the subsequence. To stay constant with the general definition of a rectangle, we give rectangle as two points (A, B), where A is the bottom left corner and B is the top right corner. Formally we represent a rectangle by two pairs of position/value $((i_1, e_1), (i_2, e_2)$, and the subsequence in π represented by this rectangle is the subsequence formed by the permutation π where we ignore the elements which are on the left of i_1, on the right of i_2, below e_1 and above e_2. For instance the rectangle $((2, 2), (4, 4))$ of the permutation 52143 is 24. The notion of rectangle goes specially well with direct/skew sum, as the left and right permutations of an operation can be define as rectangles and a rectangle (the geometric figure) is a good visualization for the left and right permutations. For this reason we often refer to the operand of a direct/skew sum as a rectangle, specially when we only need to compare the positions of the rectangles. In the same way as we compare two elements, we say that one rectangle is left (resp. right, below and above) of another one if and only if all the elements of the first rectangle are on the left (resp. right, below and above) of all the elements of the second one.

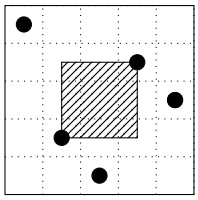


Fig. 1: the rectangle $((2, 2), (4, 4))$ of the permutation 52143.

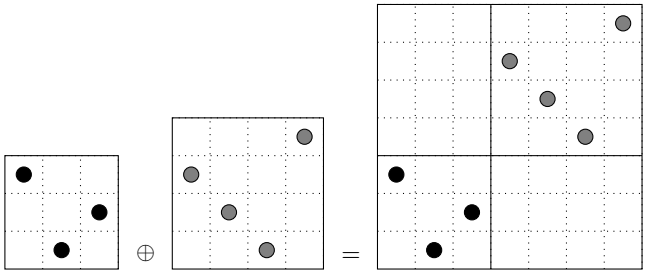


Fig. 2: $312 \oplus 3214 = 3216547$

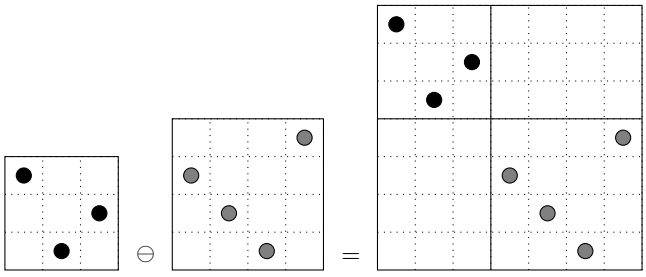
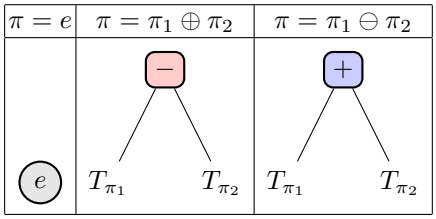


Fig. 3: $312 \ominus 3214 = 7563214$

Separable permutations may be characterized by the forbidden permutation patterns 2413 and 3142. Equivalently, any separable permutations is either a permutation with an unique element or is decomposable into a direct sum or a skew sum of two others separable permutations. Note that the decomposition may not be unique. The preceding definition exhibit a recursive structure. Thanks to that we can represent a separable permutation by a binary tree: The tree representing the separable permutations with an unique element is the tree a with a unique node/leaf which has for value the unique element, any others separable permutations is represented by a binary tree: the root encode what operation (direct sum or skew sum) the permutation is decomposed into, the left child is the tree representing the left permutation of the operation and the right child is the tree representing the right permutation of the operation. As a separable permutation may have more than one decomposition, a separable permutation may be represented by more than one tree. We call such tree a *separating tree* and we denote a separating tree representing π by T_π. This can be summarize as follows:



Formally the definition of separating tree is given by Bose, Buss, and Lubiw in [5]: a rooted binary tree in which the elements of the permutation appear (in permutation order) at the leaves of the tree, and in which the descendants of each tree node form a contiguous subset of these elements. Each interior node of the tree is either a *positive* node in which all descendants of the left child are smaller than all descendants of the right node, or a *negative* node in which all descendants of the left node are greater than all descendants of the right node. See Fig. 4 for an illustration. Each subtree of a separating tree may be interpreted as itself representing a smaller separable permutation, whose element

values are determined by the shape and sign pattern of the subtree. A one-node tree represents the trivial permutation, a tree whose root node is positive represents the direct sum of permutations given by its two child subtrees, and a tree whose root node is negative represents the skew sum of the permutations given by its two child subtrees. In this way, a separating tree is equivalent to a construction of the permutation by direct and skew sums, starting from the trivial permutation.

Let $\sigma \in S_k$ be a separable permutation, and T_σ be the corresponding separating tree. For every node v of T_σ, we let $\sigma(v)$ stand for the sequence of elements of σ stored at the leaves of the subtree rooted at v.

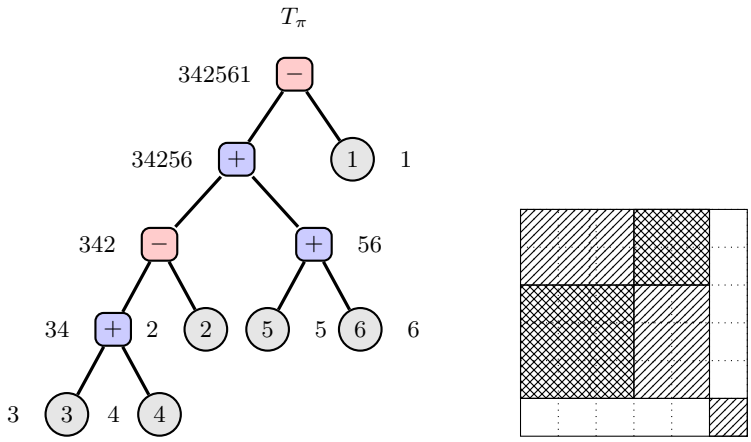


Fig. 4: On the left, a separating tree T_π for the permutation $\pi = 342561$ together with the corresponding $\sigma(v)$ sequences and on the right the decomposition of the root of this tree and of its left child: $342561 = \mathrm{red}(34256) \ominus 1 = (\mathrm{red}(342) \oplus \mathrm{red}(56)) \ominus 1$.

A *bivincular pattern* (abbreviated BVP in the sequel) $\widetilde{\sigma}$ of length k is a permutation in S_k written in two-line notation (that is the top row is $12 \dots k$ and the bottom row is a permutation $\sigma_1\sigma_2 \dots \sigma_k$). We have the following conditions on the top and bottom rows of σ (Definition 1.4.1 in [13]):

- If the bottom line of $\widetilde{\sigma}$ contains $\underline{\sigma_i\sigma_{i+1} \dots \sigma_j}$ then the elements corresponding to $\sigma_i\sigma_{i+1} \dots \sigma_j$ in an occurrence of $\widetilde{\sigma}$ in π must be adjacent, whereas there is no adjacency condition for non-underlined consecutive elements. Moreover if the bottom row of $\widetilde{\sigma}$ begins with ${}_\llcorner\sigma_1$ then any occurrence of $\widetilde{\sigma}$ in a permutation π must begin with the leftmost element of π, and if the bottom row of $\widetilde{\sigma}$ ends with $\sigma_{k\lrcorner}$ then any occurrence of $\widetilde{\sigma}$ in a permutation π must end with the rightmost element of π.
- If the top line of $\widetilde{\sigma}$ contains $\overline{i\, i+1 \dots j}$ then the elements corresponding to $i, i+1, \dots, j$ in an occurrence of $\widetilde{\sigma}$ in π must be adjacent in values, whereas

there is no value adjacency restriction for non-overlined elements. Moreover, if the top row of $\widetilde{\sigma}$ begins with $\ulcorner 1$ then any occurrence of $\widetilde{\sigma}$ in a permutation π must contain the smallest element of π, and if top row of $\widetilde{\sigma}$ ends with $k^{\urcorner}$ then any occurrence of $\widetilde{\sigma}$ in a permutation π must contain the largest element of π.

For example, let $\widetilde{\sigma} = \overline{\underset{\llcorner 213 \lrcorner}{123}}$. If $\pi_i\pi_j\pi_\ell$ is an occurrence of σ in $\pi \in S_n$, then $\pi_i\pi_j\pi_k = (x+1)x(x+2)$ for some $1 \leq x \leq n-2$, $i = 1$ and $\ell = n$. For example 316524 contains one occurrence of $\widetilde{\sigma}$ (the subsequence 324), whereas 42351 contains an occurrence of σ but not $\widetilde{\sigma}$.

3 Improved algorithm to detect a separable pattern

Let $\pi \in S_n$ and $\sigma \in S_k$, and assume that σ is a separable permutation. Ibarra [11] gave a nice $O(kn^4)$ time and $O(kn^3)$ space algorithm to detect an occurrence of σ in π. We revisit the approach of Ibarra and propose a simpler algorithm.

Since σ is a separable permutation, we can assume that we are given in addition a separating tree T_σ for σ (constructing a separating tree of a separable permutation is linear time and space [5]). Let S be a sequence of elements in $[n]$ with no repetitions. A *occurrence* of a node v of T_σ into S is an occurrence of $\mathrm{red}(\sigma(v))$ into $\mathrm{red}(S)$. The *bottom point* $\downarrow(s)$ of an occurrence s of $\sigma(v)$ into S is the minimum value of the sequence s. Similarly, the *upmost point* $\uparrow(s)$ is the maximum value of s. In the following, since all numbers in $[n]$ are positive, we adopt the convention that the maximum value occurring in an empty subset of $[n]$ is 0.

We consider the following family of subproblems that has been first introduced by Ibarra [11]: For every node v of T_σ, every two $i, j \in [n]$ with $i \leq j$, and every upper bound $\mathrm{ub} \in [n]$, we have the subproblem $\hat{\downarrow}_{v,i,j}[\mathrm{ub}]$, where the semantic is the following:

$$\hat{\downarrow}_{v,i,j}[\mathrm{ub}] \stackrel{\Delta}{=} \max\{\downarrow(s) : s \text{ is an occurrence of } \sigma(v) \text{ into } \pi[i,j] \text{ with } \uparrow(s) \leq \mathrm{ub}\}.$$

We first observe that this family of problems is already closed under induction (we do not need to introduce the family H as in [11]). These subproblems can be solved by the following equations:

- **Base**: If v is a leaf of T_σ then

$$\hat{\downarrow}_{v,i,j}[\mathrm{ub}] := \max\{\pi[\iota] : \pi[\iota] \leq \mathrm{ub}, i \leq \iota \leq j\}.$$

- **Step**: Let v_L and v_R be the left and right children of v.
  - If v is a positive node of T_σ (*i.e.*, all elements in the interval associated to v_R are larger than all elements in the interval associated to v_L), then

$$\hat{\downarrow}_{v,i,j}[\mathrm{ub}] := \max\{\hat{\downarrow}_{v_L,i,\iota-1}[\hat{\downarrow}_{v_R,\iota,j}[\mathrm{ub}]] : i < \iota \leq j\}.$$

- If v is a negative node of T_σ (*i.e.*, all elements in the interval associated to v_R are smaller than all elements in the interval associated to v_L), then

$$\hat{\downarrow}_{v,i,j}[\text{ub}] := \max\{\hat{\downarrow}_{v_R,\iota,j}[\hat{\downarrow}_{v_L,i,\iota-1}[ub]] : i < \iota \leq j\}.$$

These relations imply a $O(kn^4)$ time and $O(kn^3)$ space algorithm for detecting an occurrence of a separable permutation of length k in a permutation of length n, as obtained by Ibarra in [11], only simplified.

Proposition 1. *One can reduce the memory consumption of the algorithm above to $O(n^3 \log k)$.*

Proof. Observe first that for computing all the entries $\hat{\downarrow}_{v,\cdot,\cdot}[\cdot]$ for a certain node v with left and right children v_L and v_R, we only need the entries $\hat{\downarrow}_{v_L,\cdot,\cdot}[\cdot]$ and $\hat{\downarrow}_{v_R,\cdot,\cdot}[\cdot]$. The policy for achieving the memory spearing is the following.

- All problems for a same node v are solved together and their solution is maintained in memory until the problems for the parent of v have also been solved. At that point the memory used for node v is released.
- We use a modified DFS traversal on T_σ: for every node v which has two children, we first process its largest child (in terms of the number of nodes in the subtree rooted at that child), then the other child, and finally v itself.

We claim that the above procedure yields a $O(n^3 \log k)$ space algorithm. We first expand our DFS algorithm to what is known as the White-Gray-Black DFS [8]. First, we mark all vertices white. When we call DFS(u), we mark u to be gray. Finally, when DFS(u) returns, we mark u to be black. Provided by this colour scheme, at each step of the modified DFS, we may partition T_σ into a white-gray subtree (all nodes are either white or gray) and a forest of maximal black subtrees (all nodes are black and the parent of the root - if it exists - is either white or gray). Our space complexity claim now reduces to proving that, at any time of the algorithm, the forest contains at most $O(\log k)$ maximal black subtrees. Let h_σ be the height of T_σ, and consider any partition of T_σ into a white-gray subtree and an non-empty forest $\mathcal{T}^{\text{b}}$ of maximal black subtrees. The following property easily follows from the (standard) DFS colour scheme.

Property 1. For every $1 \leq i \leq h_\sigma$, there exist at most two maximal black subtrees in $\mathcal{T}^{\text{b}}$ whose roots are at height i in T_σ. Furthermore, if there are two maximal black subtrees in $\mathcal{T}^{\text{b}}$ whose roots are at height i in T_σ (they must have the same parent), then $\mathcal{T}^{\text{b}}$ contains no maximal black subtree whose root is at height $j > i$ in T_σ.

According to Property 1 and aiming at maximising $|\mathcal{T}^{\text{b}}|$, we may focus on the case $\mathcal{T}^{\text{b}}$ contains one maximal black subtree whose root is at height i, $1 \leq i < h_\sigma$, in T_σ (if $\mathcal{T}^{\text{b}}$ contains one maximal black subtree whose root is at height 0 in T_σ then $|\mathcal{T}^{\text{b}}| = 1$), and $\mathcal{T}^{\text{b}}$ contains two maximal black subtrees whose roots are at height h_σ in T_σ (these two maximal black subtrees reduce to size-1 subtrees). The claimed space complexity for the dynamic programming algorithm (*i.e.*,

$|\mathcal{T}^{\mathrm{b}}| = \log(k)$) now follows from the fact that we are using a modified DFS algorithm where we branch of the largest subtree first after having marked a vertex gray. Indeed, the maximal black subtree whose root is at height 1 in T_σ contains at least half of the nodes of T_σ, and the same argument applies for subsequent maximal black subtrees in the forest $\mathcal{T}^{\mathrm{b}}$. □

4 Both π and σ are separable permutations

4.1 Best algorithm so far.

When both π and σ are separable permutations we can strive for more efficient solutions since we can construct in linear time the two separating trees T_π and T_σ. It turns out, however, that the standard (*i.e.* binary) separating trees are not well-suited to handle this task. We use here the notion of *compact separating tree* (also known as *decomposition tree* [15]). Informally, in compact separating tree, we strive for every node to have as many children as possible. A simple linear time post-processing can be used to produce the decomposition tree out of the binary separating tree: As long as the separating tree contains a positive node whose father is also positive or a negative node whose father is also negative, we simply suppress that node and we let all of its children to be adopted from their grandfather in proper order. We will adopt the convention that a compact separating tree of a separating tree T_π is denoted $\tilde{T}_\pi$.

The compact tree can be understand with direct/skew sums. First note that the direct sum is associative, indeed $\pi_1 \oplus (\pi_2 \oplus \pi_3) = (\pi_1 \oplus \pi_2) \oplus \pi_3 = \pi_1 \oplus \pi_2 \oplus \pi_3$. So we can remove the parenthesis whenever the operator of the parenthesis is also a direct sum. By iterating this operation until no parenthesis with a direct sum is left, we obtain the unique and largest decomposition into direct sum of a permutation. We can do the same for skew sum. The compact tree is equivalent to the largest decomposition into direct/skew sum. The operation of removing a node and give the children of this node to the father, correspond to removing a pair of parenthesis.

One can easily visualize roughly the plot of a permutation from the compact tree by transforming it into the equivalent largest decomposition with direct/skew sum. The operation is as follow: If π has for largest decomposition into direct sum $\pi = \pi_1 \oplus \ldots \oplus \pi_\ell$ then the (unique) compact separating tree of π is the tree with a positive root and with the compact separating tree of π_1 as first child, the compact separating tree of π_i as i^{th} child and the compact separating tree of π_ℓ as ℓ^{th} child. Same goes if π is decomposed into skew sum. See Figure 5 and Figure 6 for an example. Note that when π is decomposed into direct (resp. skew) sums it forms a stair up (resp. down) of rectangles.

Now, recall that the *tree inclusion* problem for ordered and labeled trees is defined as follows: Given two ordered and labeled trees T and T', can T be obtain from T' by deleting nodes? (Deleting a node v entails removing all edges incident to v and, if v has a parent u, replacing the edge from u to v by edges from u to the children of v; see Fig. 7.) This problem has recently been recognized as

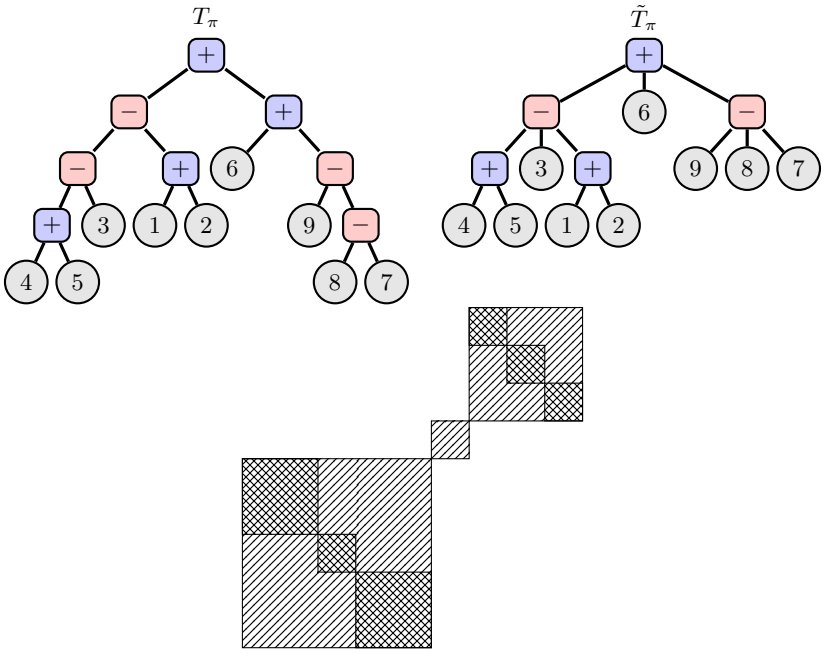


Fig. 5: A separating tree T_π for the permutation $\pi = 453126987$, the corresponding separating tree $\tilde{T}_\pi$ and the decomposition $453126987 = \mathrm{red}(45312) \oplus \mathrm{red}(6) \oplus \mathrm{red}(987) = (\mathrm{red}(45) \ominus \mathrm{red}(3) \ominus \mathrm{red}(12)) \oplus \mathrm{red}(6) \oplus (\mathrm{red}(9) \ominus \mathrm{red}(8) \ominus \mathrm{red}(7))$

an important query primitive in XML databases. The rationale for considering compact separating trees stems from the following property.

Property 2. Let π and σ be two separable permutations. We have $\sigma \preceq \pi$ if and only if the (compact separating) tree $\tilde{T}_\sigma$ is included into the (compact separating) tree $\tilde{T}_\pi$.

Kilpeläinen and Manilla [12] presented the first polynomial time algorithm using quadratic time and space for the tree inclusion problem. Since then several

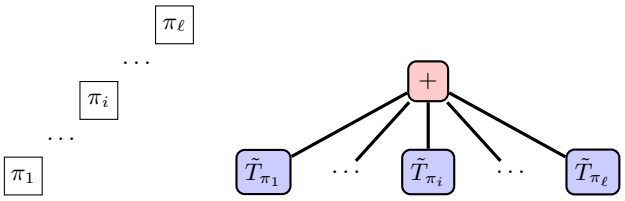


Fig. 6: On the left the permutation a rough plot of $\pi = \pi_1 \oplus \ldots \oplus \pi_i \oplus \ldots \oplus \pi_\ell$ and on the right its corresponding compact separating tree.

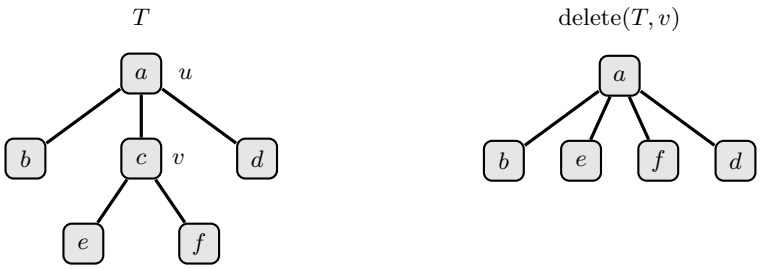


Fig. 7: The effect of removing a node from a tree.

improved results have been obtained for special cases when T and T' have a small number of leaves or small depth. However, in the worst case these algorithms still use quadratic time and space. The best algorithm is by Bille and Gørtz [4] who gave an $O(n_T)$ space and

$$O\left(\min\left\{\begin{array}{l} l_{T'}\, n_T \\ l_{T'}\, l_T \log\log n_T + n_T \\ \frac{n_T\, n_{T'}}{\log n_T} + n_T \log n_T \end{array}\right\}\right)$$

time algorithm, where n_T (resp. $n_{T'}$) denotes the number of node of T (resp. T') and l_T (resp. $l_{T'}$) denotes the number of leaves of T (resp. T').

However, all efficient solutions developed so far for the tree inclusion problem result in very complicated and hard-to-implement algorithms. For example, the main idea in the efficient algorithm presented in [4] is to construct a data structure on T supporting a small number of procedures, called the set procedures, on subsets of nodes of T.

4.2 Our Solution.

We propose here a different solution with a running time largest than the above algorithm.

Proposition 2. *There exits an $O(n^2k)$ time and $O(nk)$ space algorithm to find an occurrence of a separable pattern of length k in a separable permutation of length n.*

We differentiate 4 cases in the algorithm, depending of the decompositions of σ and π. For the 4 cases, we begin from a brute force algorithm which would try every subsequences possible and we reduce the number of subsequences to test. We start by giving propositions which are needed to the understanding and the proof of the correctness of the algorithm.

The next paragraph deals with the case where σ and π are decomposed with the same sum.

Property 3. Given the largest decomposition $\pi = \pi_1 \oplus \ldots \oplus \pi_{\ell_\pi}$ and the largest decomposition $\sigma_1 \oplus \ldots \oplus \sigma_{\ell_\sigma}$, no occurrence of σ_i can be split into two or more rectangles of π.

Proof. If the occurrence of σ_i can be split into two or more rectangles of π then σ_i can be decomposed into direct sum, which is not possible as σ_i is part of a largest decomposition of direct sum.

From this property we know that in an occurrence of σ in π, any rectangle π_i contains no, one or more than one rectangles of σ. Given an occurrence, by taking all the rectangles of π which contain at least one rectangle of σ, we form a subsequence of rectangles $\pi_{P_1}, \ldots, \pi_{P_m}$ of $\pi_1, \ldots, \pi_{\ell_\pi}$, for each of those rectangles we associate the set of rectangles that this rectangle contains in the occurrence, we form a sequence of pair (π_i, P_i) where the permutation represented by the set P_i occurs in π_i. Note that the P_i forms an ordered partitioning of the rectangles of σ. Reversibly, finding sequence is enough to prove that an occurrence of σ exists in π.

Lemma 1. *Given the largest decomposition $\pi = \pi_1 \oplus \ldots \oplus \pi_{\ell_\pi}$ and the largest decomposition $\sigma = \sigma_1 \oplus \ldots \oplus \sigma_{\ell_\sigma}$, if there exists a sequence (π_i, P_i) such that the permutation represented by the set P_i occurs in π_i and all the P_i forms an ordered partitioning of the rectangles of σ then σ occurs in π.*

Proof. Suppose that such sequence does not yield an occurrence, so there exists at least two elements $\sigma[i]$ and $\sigma[j]$ such that their matching do not have the same order as $\sigma[i]$ and $\sigma[j]$. Without lost of generality, we can suppose that $i < j$. Suppose that $\sigma[i]$ and $\sigma[j]$ are contain in a rectangle/two rectangles which is/are contain in P_i, by definition of the sequence, the permutations represented by P_i occurs in π_i, so this case cannot happen. We are left with the case where $\sigma[i]$ and $\sigma[j]$ are contain in rectangles which are contain in P_i and P_j, as the elements are contains in different sets, we deduce that they are in different rectangle moreover σ is decomposed direct sum so P_i is on the left and below P_j thus $\sigma[i] < \sigma[j]$. From the hypothesis their matchings are not in the same order as $\sigma[i]$ and $\sigma[j]$, but their matching are contains in π_i and π_j and π_i is on the left and below π_j as π is decomposed in direct sum, so their matching cannot have different order, which prove that the hypothesis does not hold. So such sequence yield an occurrence. □

We give an algorithm that construct such sequence, which can be used to decide whether σ occurs in π.

Algorithm 1:

Data: The sequence of rectangles $\pi_1, \ldots, \pi_{\ell_\pi}$ of the largest decomposition of π.
Data: The sequence of rectangles $\sigma_1, \ldots, \sigma_{\ell_\sigma}$ of the largest decomposition of σ.
Result: A sequence as describe above.
start with an empty ordered partition and an empty subsequence ;
Initialize s with $s = \sigma_1, \ldots, \sigma_{\ell_\sigma}$;
for *every rectangle in* $\pi_1, \ldots, \pi_{\ell_\pi}$ **do**
  p = the largest sequence of rectangles of s strating from the first one such that this sequence occurs in the current rectangle;
  if *p is not empty* **then**
    add p to the ordered partition ;
    add the current rectangle to the subsequence ;
    remove p from the s ;
if *s is empty* **then**
  return the solution
else
  return that no solution exists

Note that durinf the "for", the current rectangle is decomposed in skew sum as we are working with the largest decomposition, there are three cases possible, either p is empty, contain a rectangle or contain rectangles. In the second case the rectangle is also decomposed in skew sum, which means that we want to decide whether a largest decomposition in skew sum occurs in another largest decomposition in skew sum, which is solve symmetrically as the current case. For the third case the rectangles are decomposed in largest decomposition direct sum, which means that we want to decide whether a largest decomposition in direct sum occurs in a largest decomposition in skew sum, which we will solve later.

As see above, the algorithm computes a sequence, so if the algorithm give a solution, it means that σ occurs in π. To show that our algorithm find a solution if and only if a σ occurs in π, we still need to prove that if the algorithm does not find a solution then no solution exists. We prove the contraposition. Suppose that a sequence (π_i, P_i) solution exists and that our algorithm does not compute a solution. Note that the rectangles of the solution are all contains in π and as the algorithm over π does not compute sequence, it does not find a correct sequence over the rectangles of the solution, we prove that this assertion is not true. We can always suppose that the algorithm start to compute the same sequence as the solution but start to differ for the rectangle π_i. There is only three differences possible: the algorithm find no rectangle or it find a set which is contain in P_i or it find a set that contain P_i. The first case is not possible because it implies that there is nothing that occurs in π_i but by hypothesis at least P_i occurs in π_i. The second case implies that the algorithm did not compute the largest set of rectangles which is not possible. Suppose that the third case happen, we can

create a new solution which is closer to the sequence that the algorithm compute: as the set $P_{i'}$ computed by the algorithm contains the set P_i, we replace the pair (π_i, P_i) by the pair $(\pi_i, P_{i'})$ in the solution, as $P_{i'}$ contains more rectangles than P_i, we remove from the next sets P_j the rectangles that appears in $P_{i'}$ but not in P_i. We obtain a sequence where there is no difference between the sequence computed by the algorithm or where the difference starts after the pair corresponding to p_i. By iterating this process we can create a solution which have no difference between the sequence computed by the algorithm, so the algorithm compute a solution, which is a contradiction.

The next paragraph deals with the case where σ and π are decomposed with different sum. We focus on the case where π is decomposed in skew sum and σ is decomposed in direct sum. The other case can be dealt with symmetry.

Proposition 3. *Given the largest decomposition $\pi = \pi_1 \ominus \ldots \ominus \pi_{\ell_\pi}$ and the largest decomposition $\sigma_1 \oplus \ldots \oplus \sigma_{\ell_\sigma} = \sigma$, σ occurs in π if and only if there exists π_i such that σ occurs in π_i.*

This proposition claims that σ can only occur in a rectangle of π. This can be easily explained visually, σ forms a "stair up", if it is contain in more than one rectangle of π, as the rectangles of π form a "stair down", the "stair up" of σ would be cut.

Proof. For the backward implication, π_i occurs in π and σ occurs in π_i so σ occurs in π. For the forward implication, suppose that the left most element of σ is matched to an element of π_α and the right most element of σ is a matched to an element of π_β. The left most element of σ is below the right most element of σ as σ is formed by an increasing sequence of rectangles but every element of π_α is above every element of π_β as π is formed by a decreasing sequence of rectangles, So such occurrence is not possible. □

The proposition directly leads to the following algorithm to decide whether σ occurs in π.

Algorithm 2:

Data: The sequence of rectangles $\pi = \pi_1, \ldots, \pi_{\ell_\pi}$ of the largest decomposition of π
Data: σ
Result: Whether σ occurs in π
for *every rectangle in* $\pi_1, \ldots, \pi_{\ell_\pi}$ **do**
  if σ *occurs in the current rectangle* **then**
    return that σ occurs π ;
return that σ does not occur in π ;

Note that to decide whether σ occurs in the current rectangle, this algorithm uses the above algorithm and the above algorithm use this algorithm.

We modify this algorithm for the purpose of computing p require for the first algorithm: instead of computing whether σ occurs in π, the algorithm compute

the rightmost rectangle such that the $\sigma_1, \ldots, \sigma_i$ occurs in π. Remark that if this rectangle is σ_{ℓ_σ} then we can conclude that σ occurs in π.

Algorithm 3:

Data: The sequence of rectangles $\pi = \pi_1, \ldots, \pi_{\ell_\pi}$ of the largest decomposition of π
Data: The sequence of rectangles $\sigma_1, \ldots, \sigma_{\ell_\sigma}$ of the largest decomposition of σ
Result: The rightmost σ_i such that $\sigma_1, \ldots, \sigma_i$ occurs in π
initialize r with nothing ;
for *every rectangle in* $\pi_1, \ldots, \pi_{\ell_\pi}$ **do**
    tmp = The rightmost σ_i such that $\sigma_1, \ldots, \sigma_i$ occurs in the current rectangle ;
    if *tmp is at the right of* r **then**
        $r = tmp$;
return r ;

Note that we also need to modify the first algorithm to compute *tmp*, but the modification is small: instead of returning the solution or that no solution exists, the algorithm return the last rectangle added in p.

We have describe two algorithms which can be used to compute every case possible, so we have an algorithm to decide whether σ occurs in π. We still need to prove the complexity claim. To do so, we introduce a closed set of entries for the algorithms and by using dynamic programming, we can ensure that each algorithm is only called once for each element of the closed set. This gives us a bound to the numbers of time that each algorithm is called. The first algorithm is called to compute *tmp* for the third algorithm or to decide whether σ occurs in π, in both case it only requires two rectangles: σ_i and π_j. The third algorithm is called to compute p for the first algorithm or to decide whether σ occurs in π, for the second case it requires two rectangles σ and π but for the first case it requires a π_i and a sequence of rectangles especially $s = \sigma_i, \ldots, \sigma_{\ell_\sigma}$ but remark that the last rectangle of the sequence is always the rightmost rectangle of the decomposition. We use a good structure to represent the rectangles so that we can represent the sequence with only one rectangle. A good structure to represent the rectangles is the compact separable tree. Indeed, remember that in the tree representation a node represent a rectangle and the decomposition of a rectangle is represented by the children of the node, then is means that the node representing $\sigma_i, \ldots, \sigma_{\ell_\sigma}$ have the same father and thus from the node representing σ_i, the node representing σ_{ℓ_σ} is its rightmost brother. To resume, each algorithm only need to take one node from the compact tree of σ and one node from the compact tree of π. As there is $O(n)$ nodes for the tree of π and $O(k)$ nodes for the tree of σ, each there are $O(n.k)$ entries possible for the algorithms. Finally each algorithm is computed in $O(n)$ times, as they are only doing a iteration over the rectangles of π, then the algorithm run in $O(k.n^2)$ time and $O(k.n)$ space.

5 Deciding the union of a separable permutations

This subsection is devoted to shuffling permutations. Given three permutations π, σ and τ, the problem is to decide whether π is the disjoint union of two patterns that are order-isomorphic to σ and τ, respectively in notation $\pi \in \sigma \sqcup\!\sqcup \tau$. For example 937654812 is the disjoint union of two patterns that are order-isomorphic to 2431 and 53241, as can be seen in the highlighted form **937654812**. This problem is of interest since it is strongly related to two others problems that naturally arise in the context of pattern matching for permutations. The first one is to decide whether the pattern containment problem for parameter $n - k$ is fixed-parameter tractable (FPT). (Recall that the pattern containment problem for parameter k is fixed-parameter tractable [10].) The second one is to decide whether a permutation is a square: Given a permutation π, does there exists a permutation σ such that π is the disjoint union of two patterns that are both order-isomorphic to σ? This problem has recently been proved to be **NP**-complete [9] for general permutations.

Proposition 4. *Given three separable permutations π of length n, σ of length k and τ of length ℓ, there exists a $O(nk^3\ell^2)$ time and $O(nk^2\ell^2)$ space algorithm to decide whether π is the disjoint union of two patterns that are order-isomorphic to σ and τ, respectively.*

Proposition 4 gains in interest if we observe that the complexity of the problem is still open if we do not restrict the input permutations to be separable [9].

First remark that if π is separable, and $\pi \in \sigma \sqcup\!\sqcup \tau$ then σ and τ are also separable. In other words the case where σ or/and τ is/are not separable is always false. Thus we consider the non trivial case where σ and τ are separable.

In the following, given π a separable permutation we denote π_{ℓ_π} as the right most rectangle of its largest decomposition especially $\pi = \pi_1 \oplus \ldots \oplus \pi_{\ell_\pi}$. Moreover we let $\pi(i, j)$ be $\pi_i \oplus \ldots \oplus \pi_j$.

Consider the following family of subproblems : Given a separable permutation π and a sequence of its largest decomposition $\pi(i_\pi, j_\pi)$, a separable permutation σ and a sequence of its largest decomposition $\sigma(i_\sigma, j_\sigma)$ and a separable permutation τ and a sequence of its largest decomposition $\tau(i_\tau, j_\tau)$, we want to know whether $\pi(i_\pi, j_\pi)$ is the shuffle of $\sigma(i_\sigma, j_\sigma)$ and $\tau(i_\tau, j_\tau)$. Which we write in notation

$$\mathrm{S}(\pi(i_\pi, j_\pi), \sigma(i_\sigma, j_\sigma), \tau(i_\tau, j_\tau)) = \begin{cases} True & \text{if } \pi(i_\pi, j_\pi) \in \sigma(i_\sigma, j_\sigma) \sqcup\!\sqcup \tau(i_\tau, j_\tau) \\ False & \text{otherwise.} \end{cases}$$

By definition $\pi \in \sigma \sqcup\!\sqcup \tau$ if and only if $\mathrm{S}(\pi(1, \ell_\pi), \sigma(1, \ell_\sigma), \tau(1, \ell_\tau))$ is true.

Base.

The base cases are when $\sigma = 1$ or (exclusive or) $\tau = 1$. Then if π is also a leaf then the problem is true, otherwise the problem is false, as elements will be left unmatched in π.

- If π and σ are leaves and τ has no element then $\mathrm{S}(\pi(1,1),\sigma(1,1),\emptyset) = True$
- if π and τ are leaves and σ has no element then $\mathrm{S}(\pi(1,1),\emptyset,\tau(1,1)) = True$
- If π is not a leaf, σ is a leaf and τ has no element then $\mathrm{S}(\pi(i_\pi,j_\pi),\sigma(1,1),\emptyset) = False$
- If π is not a leaf, τ is a leaf and σ has no element then $\mathrm{S}(\pi(i_\pi,j_\pi),\emptyset,\tau(1,1)) = False$

Reduction.
Two instances of the problems can represent the same problem. Especially when one of the arguments represents an unique node, we replace this node by all of its children. This happens only when $i_* = j_*$, where $* \in \{\pi,\sigma,\tau\}$.

- If $i_\pi = j_\pi$ then

$$\mathrm{S}(\pi(i_\pi,i_\pi),\sigma(i_\sigma,j_\sigma),\tau(i_\tau,j_\tau)) = \mathrm{S}(\pi_{i_\pi}(1,\ell_{\pi_{i_\pi}}),\sigma(i_\sigma,j_\sigma),\tau(i_\tau,j_\tau))$$

- If $i_\sigma = j_\sigma$ then

$$\mathrm{S}(\pi(i_\pi,j_\pi),\sigma(i_\sigma,i_\sigma),\tau(i_\tau,j_\tau)) = \mathrm{S}(\pi(i_\pi,j_\pi),\sigma_{i_\sigma}(1,\ell_{\sigma_{i_\sigma}}),\tau(i_\tau,j_\tau))$$

- If $i_\tau = j_\tau$ then

$$\mathrm{S}(\pi(i_\pi,j_\pi),\sigma(i_\sigma,j_\sigma),\tau(i_\tau,i_\tau)) = \mathrm{S}(\pi(i_\pi,j_\pi),\sigma(i_\sigma,j_\sigma),\tau_{i_\tau}(1,\ell_{\tau_{i_\tau}}))$$

Recurrence
The idea of the recursion is to splits $\sigma(i_\sigma,j_\sigma)$ and $\tau(i_\tau,j_\tau)$, in every way possible in $\pi(i_\pi,j_\pi)$. For each splitting the obtain two different instances of the problem. We can sort the splitting depending of the pair of instances obtain. The first class can be characterized by the fact that in one of the instance in the pair σ or τ is empty, this is the best case as this correspond to decide whether a separable permutation occurs in another separable permutation without element left. The second class is when the above case does not happen. Note that, in any case, the both instances are smaller in size, especially we reduce the size of the text permutation and the size of either the first or the second pattern. The recursion use the property 3 and the lemma 1 to make "smart" splitting.

- Case where $\pi(i_\pi,j_\pi)$ represents a direct sum decomposition, $\sigma(i_\sigma,j_\sigma)$ represents a direct sum decomposition and $\tau(i_\tau,j_\tau)$ represent a skew sum decomposition: Note that by property 3, if $\tau(i_\tau,j_\tau))$ occurs in $\pi(i_\pi,j_\pi)$ then it occurs in a unique rectangle, especially we can consider the first rectangle of $\pi(i_\pi,j_\pi)$ for the occurrence of $\tau(i_\tau,j_\tau)$ and handle the case accordingly. So π_{i_π} can contain:
  - An occurrence of $\tau(i_\tau,j_\tau))$ and some part of an occurrence of $\sigma(i_\sigma,j_\sigma)$.
  - An occurrence of $\tau(i_\tau,j_\tau))$ and no occurrence of $\sigma(i_\sigma,j_\sigma)$.
  - Nothing of $\tau(i_\tau,j_\tau))$ and some part of an occurrence of $\sigma(i_\sigma,j_\sigma)$.

  The case where π_{i_π} contains nothing cannot happen has we want every element of π to be used in an occurrence. In those three cases, what is left of σ and τ has to occur in $\pi(i_\pi+1,j_\pi)$. Which give us the following solution:

$$\mathrm{S}(\pi(i_\pi,j_\pi),\sigma(i_\sigma,j_\sigma),\tau(i_\tau,j_\tau))$$

$$=$$

$$\bigcup_{j'_\sigma < j_\sigma} (\mathrm{S}(\pi(i_\pi, i_\pi), \sigma(i_\sigma, j'_\sigma), \tau(i_\tau, j_\tau)) \wedge \mathrm{S}(\pi(i_\pi + 1, j_\pi), \sigma(j'_\sigma + 1, j_\sigma), \emptyset))$$

$$OR$$

$$(\mathrm{S}(\pi(i_\pi, i_\pi), \emptyset, \tau(i_\tau, j_\tau)) \wedge \mathrm{S}(\pi(i_\pi + 1, j_\pi), \sigma(i_\sigma, j_\sigma), \emptyset))$$

$$OR$$

$$\bigcup_{j'_\sigma < j_\sigma} (\mathrm{S}(\pi(i_\pi, i_\pi), \sigma(i_\sigma, j'_\sigma), \emptyset) \wedge \mathrm{S}(\pi(i_\pi + 1, j_\pi), \sigma(j'_\sigma + 1, j_\sigma), \tau(i_\tau, j_\tau)))$$

– Case where $\pi(i_\pi, j_\pi)$ represents a direct sum decomposition, $\sigma(i_\sigma, j_\sigma)$ represents a skew sum decomposition and $\tau(i_\tau, j_\tau)$ represent a skew sum decomposition: By property 3, $\sigma(i_\sigma, j_\sigma)$ occurs in a unique child of $\pi(i_\pi, j_\pi)$ $\tau(i_\tau, j_\tau)$ occurs in a unique child of $\pi(i_\pi, j_\pi)$. Moreover, as every element must be used in a occurrence $\pi(i_\pi, j_\pi)$ cannot have more than two rectangles. In other words if $\pi(i_\pi, j_\pi)$ has more than two rectangles than this instance is false and if $\pi(i_\pi, j_\pi)$ has two rectangle $\sigma(i_\sigma, j_\sigma)$ occurs π_{i_π} and $\tau(i_\tau, j_\tau)$ occurs π_{j_π} or $\sigma(i_\sigma, j_\sigma)$ occurs π_{j_π} and $\tau(i_\tau, j_\tau)$ occurs π_{i_π}:

$$\mathrm{S}(\pi(i_\pi, j_\pi), \sigma(i_\sigma, j_\sigma), \tau(i_\tau, j_\tau))$$

$$=$$

$$\mathrm{S}(\pi(i_\pi, i_\pi), \sigma(i_\sigma, j_\sigma), \emptyset) \wedge \mathrm{S}(\pi(j_\pi, j_\pi), \emptyset, \tau(i_\tau, j_\tau))$$

$$OR$$

$$\mathrm{S}(\pi(i_\pi, i_\pi), \emptyset, \tau(i_\tau, j_\tau)) \wedge \mathrm{S}(\pi(j_\pi, j_\pi), \sigma(i_\sigma, j_\sigma), \emptyset)$$

– If $\pi(i_\pi, j_\pi)$, $\sigma(i_\sigma, j_\sigma)$ and $\tau(i_\tau, j_\tau)$ represent direct sum decompositions then by lemma 1, π_{i_π} contains:
  - Some part of the occurrence of $\tau(i_\tau, i_\tau)$ and some part of the occurrence of $\sigma(i_\sigma, j_\sigma)$.
  - Some part of the occurrence of $\tau(i_\tau, i_\tau)$ and nothing of the occurrence of $\sigma(i_\sigma, j_\sigma)$.
  - Nothing of the occurrence of $\tau(i_\tau, i_\tau)$ and some part of the occurrence of $\sigma(i_\sigma, j_\sigma)$.
  - The occurrence of $\tau(i_\tau, i_\tau)$ and some part of the occurrence of $\sigma(i_\sigma, j_\sigma)$.
  - The occurrence of $\tau(i_\tau, i_\tau)$ and nothing of the occurrence of $\sigma(i_\sigma, j_\sigma)$.
  - Some part of the occurrence of $\tau(i_\tau, i_\tau)$ and the occurrence of $\sigma(i_\sigma, j_\sigma)$.
  - Nothing of the occurrence of $\tau(i_\tau, i_\tau)$ and the occurrence of $\sigma(i_\sigma, j_\sigma)$.

  In all those cases, what is left occurs in $\pi(i_\pi + 1, j_\pi)$.

$$\mathrm{S}(\pi(i_\pi, j_\pi), \sigma(i_\sigma, j_\sigma), \tau(i_\tau, j_\tau))$$

$$=$$

$$\bigcup_{\substack{j'_\sigma<j_\sigma \\ j'_\tau<j_\tau}} \mathrm{S}(\pi(i_\pi,i_\pi),\sigma(i_\sigma,j'_\sigma),\tau(i_\tau,j'_\tau))\wedge\mathrm{S}(\pi(i_\pi{+}1,j_\pi),\sigma(j'_\sigma{+}1,j_\sigma),\tau(j'_\tau{+}1,j_\tau))$$

$$OR$$

$$\bigcup_{j'_\sigma<j_\sigma} \mathrm{S}(\pi(i_\pi,i_\pi),\sigma(i_\sigma,j'_\sigma),\emptyset) \wedge \mathrm{S}(\pi(i_\pi+1,j_\pi),\sigma(j'_\sigma+1,j_\sigma),\tau(i_\tau,j_\tau))$$

$$OR$$

$$\bigcup_{j'_\tau<j_\tau} \mathrm{S}(\pi(i_\pi,i_\pi),\emptyset,\tau(i_\tau,j'_\tau)) \wedge \mathrm{S}(\pi(i_\pi+1,j_\pi),\sigma(i_\sigma,j_\sigma),\tau(j'_\tau+1,j_\tau))$$

$$OR$$

$$\bigcup_{j'_\tau<j_\tau} \mathrm{S}(\pi(i_\pi,i_\pi),\sigma(i_\sigma,j_\sigma),\tau(i_\tau,j'_\tau)) \wedge \mathrm{S}(\pi(i_\pi+1,j_\pi),\emptyset,\tau(j'_\tau+1,j_\tau))$$

$$OR$$

$$\mathrm{S}(\pi(i_\pi,i_\pi),\sigma(i_\sigma,j_\sigma),\emptyset) \wedge \mathrm{S}(\pi(i_\pi+1,j_\pi),\emptyset,\tau(i_\tau,j_\tau))$$

$$OR$$

$$\bigcup_{j'_\sigma<j_\sigma} \mathrm{S}(\pi(i_\pi,i_\pi),\sigma(i_\sigma,j'_\sigma),\tau(i_\tau,j_\tau)) \wedge \mathrm{S}(\pi(i_\pi+1,j_\pi),\sigma(j'_\sigma+1,j_\sigma),\emptyset)$$

$$OR$$

$$\mathrm{S}(\pi(i_\pi,i_\pi),\emptyset,\tau(i_\tau,j_\tau)) \wedge \mathrm{S}(\pi(i_\pi+1,j_\pi),\sigma(i_\sigma,j_\sigma),\emptyset)$$

– The others cases can be dealt with symmetry.

We are left with proving the complexity claim. As before we use a dynamic programming strategy so that we only need to compute each instance of the problem once, so we only need to enumerate the number of different entry possible for the problem to find the complexity of the problem. For this paragraph let ℓ be the size of τ. Without lost of generality we can suppose that $\ell < k$. At first it seems that the problem have $n^2.k^2.\ell^2$ cases: for every permutation we associate two rectangles of its decomposition.

But in those three pairs, one pair has a redundant information: the size of $\pi(i_\pi, j_\pi)$ must be equal to the size of $\sigma(i_\sigma, j_\sigma)$ plus the size of $\tau(i_\tau, j_\tau)$, so given $\sigma(i_\sigma, j_\sigma)$, $\tau(i_\tau, j_\tau)$, π and i_π, we can deduce j_π. Note that j_π may not exists as the size of $\sigma(i_\sigma, j_\sigma)$ plus the size of $\tau(i_\tau, j_\tau)$ may not exactly equal to the size of a $\pi(i_\pi, j_\pi)$, but in that case we can immediately say that π is not a shuffle. So we have $O(n.k^2.\ell^2)$ different cases to compute. Note that this strategy implies that for every permutation, we have to compute the size of every $\pi(i_\pi, j_\pi)$ possible, which can be pre-computed in $O(n^2)$ time and will take $O(n^2)$ space in the memory.

The worst case scenario to compute a problem is when π, σ and τ are of the same decomposition and some part of the occurrence of $\tau(i_\tau, i_\tau)$) and some part

of the occurrence of $\sigma(i_\sigma, j_\sigma)$. In this case we must iterate every j'_σ, $i_\sigma \leq j'_\sigma \leq j_\sigma$ and every j'_τ, $i_\tau \leq j'_\tau \leq j_\tau$ possible. But remark that (same as above) from $\pi(i_\pi, i_\pi)$, $\tau(i_\tau, j'_\tau)$, σ and i_σ we can deduce j'_σ. So we only need to iterate every j'_τ, $i_\tau \leq j'_\tau \leq j_\tau$ possible. Finally each recursive problem is solved in constant time by dynamic programming. So to compute one case, we take at most $O(\ell)$ time. This gives us an $O(n.k^2.\ell^3)$ time and $O(n.k^2.\ell^2)$ space algorithm.

6 Finding a maximum size separable subpermutation

The longest common pattern problem for permutations is to find the largest permutation that occur in each input permutation. The problem is intended to be the natural counterpart to the classical longest common the subsequence problem. Rossin and Bouvel [14] gave an $O(n^8)$ time algorithm for computing the largest common separable pattern that occurs in two permutations of size (at most) n, one of these two permutation being separable. This problem was further generalised in [6] where it is shown that that the problem of computing the largest separable pattern that occurs in k permutations of size (at most) n is solvable in $O(n^{6k+1})$ time and $O(n^{4k+1})$ space. Notice that this later problem is **NP**-complete for unbounded k, even if all input permutations are actually separable.

The following proposition improves upon the algorithm of Rossin and Bouvel [14].

Proposition 5. *Given a permutation of size n and a separable permutation of size k, one can compute in $O(kn^6)$ time and $O(n^4 \log k)$ space the largest common separable pattern that occurs in two input permutations.*

For clarity of exposition, we begin by considering the problem of computing the largest separable pattern that occurs in a single permutation π. We consider the following family of subproblems: For every two $i, j \in [n]$ with $i \leq j$, and every lower and upper bound $\mathrm{lb}, \mathrm{ub} \in [n]$ with $\mathrm{lb} \leq \mathrm{ub}$, we have the subproblem $P_{i,j,\mathrm{lb},\mathrm{ub}}$, where the semantic is the following:

$$P_{i,j,\mathrm{lb},\mathrm{ub}} \stackrel{\Delta}{=} \max\{|s| : s \text{ is a subsequence of } \pi[i,j], \mathrm{red}(s) \text{ is separable, and every element in } s \text{ is in the interval } [\mathrm{lb}, \mathrm{up}]\}.$$

We show that this family of problems is closed under induction.

- **Base**: We have two base cases.
  - If $i = j$, then
$$P_{i,i,\mathrm{lb},\mathrm{ub}} := \begin{cases} 1 & \text{if } \mathrm{lb} \leq \pi[i] \leq \mathrm{ub}, \\ 0 & \text{otherwise.} \end{cases}$$
  - If $\mathrm{lb} = \mathrm{up}$, then
$$P_{i,j,\mathrm{b},\mathrm{b}} := \begin{cases} 1 & \text{if there exists } \iota \in [i,j] \text{ such that } \pi[\iota] = \mathrm{b}, \\ 0 & \text{otherwise.} \end{cases}$$

– **Step**: Here $i < j$, lb $<$ ub, and we must decide whether the optimum $P_{i,i,\mathrm{lb},\mathrm{ub}}$ is achieved with a positive or negative root node. Thus

$$P_{i,j,\mathrm{lb},\mathrm{ub}} = \max\left\{P^{+}_{i,j,\mathrm{lb},\mathrm{ub}}, P^{-}_{i,j,\mathrm{lb},\mathrm{ub}}\right\},$$

where

- (Hypothesis of a positive node)

$$P^{+}_{i,j,\mathrm{lb},\mathrm{ub}} := \max\{P_{i,\iota-1,\mathrm{lb},\mathrm{b}-1} + P_{\iota,j,\mathrm{b},\mathrm{ub}} : i < \iota \leq j \text{ and } \mathrm{lb} < \mathrm{b} \leq \mathrm{ub}\}.$$

- (Hypothesis of a negative node)

$$P^{-}_{i,j,\mathrm{lb},\mathrm{ub}} := \max\{P_{i,\iota-1,\mathrm{b}-1,\mathrm{ub}} + P_{\iota,j,\mathrm{lb},\mathrm{b}} : i < \iota \leq j \text{ and } \mathrm{lb} < \mathrm{b} \leq \mathrm{ub}\}.$$

This implies an $O(n^6)$ time and $O(n^4)$ space algorithm for finding the largest separable pattern that occurs in a permutation of size n.

The next step is to consider the problem of finding a longest separable pattern that occurs in two given permutations (that may not be separable themselves) [14]. Note that this problem contains as a special case the pattern containment problem for separable patterns. Let π be a permutation of size n. For every $i, j \in [n]$ with $i \leq j$ and every $\mathrm{ub}, \mathrm{lb} \in [n]$ with lb] $\leq$ up, we let $\pi[i, j, \mathrm{lb}, \mathrm{up}]$ stand for the subsequence obtained from $\pi[i, j]$ by trimming away all elements above lb or below up. Now, let π_1 and π_2 be two permutations of S_n. We consider the following family of subproblems: For every $i_1, j_1, \mathrm{lb}_1, \mathrm{ub}_1 \in [n]$ with $i_1 \leq j_1$ and $\mathrm{lb}_1 \leq \mathrm{ub}_1$, and every $i_2, j_2, \mathrm{lb}_2, \mathrm{ub}_2 \in [n]$ with $i_2 \leq j_2$ and $\mathrm{lb}_2 \leq \mathrm{ub}_2$, we have the subproblem $P_{i_1,j_1,\mathrm{lb}_1,\mathrm{ub}_1,i_2,j_2,\mathrm{lb}_2,\mathrm{ub}_2}$ where the semantic is as follows:

$$P_{i_1,j_1,\mathrm{lb}_1,\mathrm{ub}_1,i_2,j_2,\mathrm{lb}_2,\mathrm{ub}_2} \stackrel{\Delta}{=} \max\{|s| : s \text{ is a pattern occuring both in } \pi_1[i_1, j_1, \mathrm{lb}_1, \mathrm{ub}_1] \text{ and in } \pi_2[i_2, j_2, \mathrm{lb}_2, \mathrm{ub}_2]\}$$

It is easy to see that this family of subproblems is closed under induction and yields a $O(n^{12})$ time and $O(n^8)$ space algorithm for finding the size of largest separable pattern that occurs in two permutation of size (at most) n. This is a slight improvement compared to [6] where an $O(n^{13})$ time and $O(n^9)$ space algorithm is proposed.

The outlined approach can be extended to a polynomial-time algorithm for a fixed number of input permutations (as shown in [6]). However, in practice the complexity of this solution is already prohibitive for just two sequences. Therefore, rather than further extending this approach we focus on underlining how it encompasses other natural problems. Indeed, following Bouvel and Rossin [14], we consider the problem of computing a longest separable pattern that occurs in two input permutations of length at most n, one of these two permutation being separable. For this precise problem Bouvel and Rossin gave an $O(n^8)$ algorithm. We consider the following family of subproblems: For every node v of T_σ, every two $i, j \in [n]$ with $i \leq j$, and every lower and upper bounds $\mathrm{lb}, \mathrm{ub} \in [n]$ with $\mathrm{lb} \leq \mathrm{up}$, we have the subproblem $P_{v,i,j,\mathrm{lb},\mathrm{ub}}$, where the semantic is the following.

$$P_{v,i,j,\mathrm{lb},\mathrm{ub}} \stackrel{\Delta}{=} \max\{|s| : s \text{ is a common subsequence of } v \text{ and } \pi[i, j] \text{ with all values in the interval } [\mathrm{lb}, \mathrm{ub}].$$

We show that this family of problems is closed under induction.

– **Bases**: We have three base cases:
  • If $i = j]$ then
$$P_{v,i,i,\mathrm{lb},\mathrm{ub}} := \begin{cases} 1 & \text{if } lb \leq \pi[i] \leq \mathrm{ub}, \\ 0 & \text{otherwise.} \end{cases}$$
  • If $\mathrm{lb} = \mathrm{ub}]$ then
$$P_{v,i,j,\mathrm{b},\mathrm{b}} := \begin{cases} 1 & \text{if there exists } \iota \in [i,j] \text{ such that } \pi[\iota] = b, \\ 0 & \text{otherwise.} \end{cases}$$
  • If v is a leaf then
$$P_{v,i,j,\mathrm{lb},\mathrm{ub}} := \begin{cases} 1 & \text{if there exists } \iota \in [i,j] \text{ such that } \mathrm{lb} \leq \pi[\iota] \leq \mathrm{ub}, \\ 0 & \text{otherwise.} \end{cases}$$

– **Step**: Here $i < j$, $\mathrm{lb} < \mathrm{ub}$, and we let v_L and v_R stand for the left and right children of v, respectively.
  • If v is a positive node then
$$P_{v,i,j,lb,ub} := \max_{i<\iota\leq j} \max_{\mathrm{lb}<\mathrm{b}\leq\mathrm{ub}} P_{v_L,i,\iota-1,\mathrm{lb},\mathrm{b}-1} + P_{v_R,\iota,j,\mathrm{b},\mathrm{ub}}.$$
  • If v is a negative node then
$$P_{v,i,j,lb,ub} := \max_{i<\iota\leq j} \max_{\mathrm{lb}<\mathrm{b}\leq\mathrm{ub}} P_{v_L,i,\iota-1,\mathrm{b}-1,\mathrm{ub}} + P_{v_R,\iota,j,\mathrm{lb},\mathrm{b}}.$$

The above description implies an $O(kn^6)$ time $O(kn^4)$ space algorithm for computing the largest common separable pattern that occurs in two permutations of size (at most) n, one of these two permutation being separable, thereby improving on Rossin and Bouvel [14]. The memory can be reduced to $O(n^4 \log k)$ with the approach detailed Section 3.

7 Vincular and bivincular separable patterns

This subsection is devoted to vincular and bivincular separable patterns. The problem is $W[1]$-complete for parameter the size of the pattern. As for regular pattern adding constraints can reduce the complexity of the problem. We prove that detecting a vincular or a bivincular separable pattern in a permutation is polynomial time solvable. To the best of our knowledge, this is the first time the pattern matching problem is proved to be tractable for a generalization of separable patterns. Since a vincular pattern is a bivincular pattern, we focus on bivincular patterns.

Recall that bivincular patterns generalize classical pattern even further than vincular patterns. Indeed, in bivincular pattern, not only positions but also values of elements involved in an occurrence may be forced to be adjacent (see

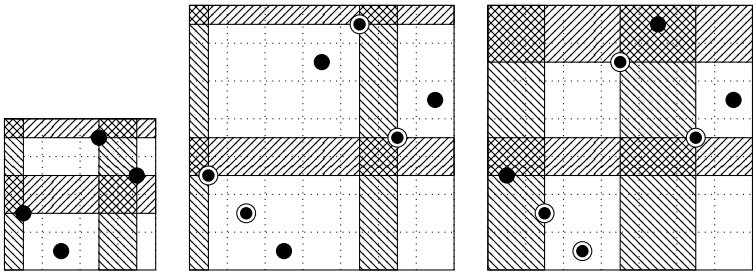


Fig. 8: From left to right, the bivincular pattern $\sigma = {}^{1\overline{234}\urcorner}_{\llcorner 21\underline{43}}$, An occurrence of σ in 3216745, An occurrence of 2143 in 3216745 but not an occurrence of σ in 3216745 because the point $(1, 3)$ and $(5, 7)$ are in the forbidden area.

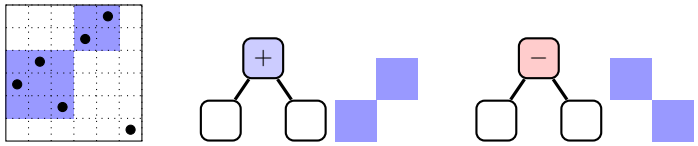


Fig. 9: From left to right the permutation 342561 (see fig 4) and the minimal rectangles of the nodes 342 and 56, the rectangles of the children of a negative node, the rectangles of the children of a positive node.

Section 2). Let $\widetilde{\sigma} = (\sigma, X, Y)$ be a separable bivincular pattern[2] of length k and $\pi \in S_n$. We represent bivincular patterns (as well as occurrences of bivincular patterns in permutations) by diagrams. Such a diagram consists in the set of points at coordinates $(i, \sigma[i])$ drawn in the plane together with forbidden regions denoting adjacency constraints. A vertical forbidden region between two points denotes the fact that the occurrences of these two points must be consecutive in positions. In a similar approach, an horizontal forbidden region between two points denotes the fact that the occurrences of these two points must be consecutive in value. Now given a permutation π and a bivincular pattern σ, The rational for introducing these augmented diagrams stems from the following fact: π contains an occurrence of a bivincular pattern $\widetilde{\sigma}$ if there exists a set of points in the diagram of π that is order-isomorphic to σ and if the forbidden region do not contain any point (see Figure 8).

The algorithm of section 3 cannot be used to find an occurrence of a bivincular pattern as we do not have any control on the position and the value of matching element.

Proposition 6. *Given a permutation π of length n and a bivincular separable pattern σ of length k, there exists an $O(kn^6)$ time and $O(kn^4)$ space algorithm to decide whether σ occurs in π.*

[2] This is a shortcut for σ being separable and (σ, X, Y) being bivincular.

Given a bivincular separable permutation pattern $\widetilde{\sigma}$, and a subsequence σ', we define the bivincular permutation $\widetilde{\sigma'}$ has the binvincular permutation with the element of σ', with top line the top line of $\widetilde{\sigma}$ where we remove the elements not in σ' and with bottom line the elements of the node where m and $m+1$ are underlined if and only if m and $m+1$ are element of σ' and m and $m+1$ are underlined in $\widetilde{\sigma}$, moreover:

- if $\sigma[i]$ is the bottommost element of σ' and $\underline{\sigma[i-1]\sigma[i]}$ or $\llcorner\sigma[i]$ then we add the bottom left corner.
- if $\sigma[i]$ is the toptommost element of σ' and $\underline{\sigma[i]\sigma[i+1]}$ or $\sigma[i]\lrcorner$ then we add the bottom right corner.
- if $\sigma[i]$ is the leftmost element of σ' and $\overline{\sigma[i-1]\sigma[i]}$ or $\ulcorner\sigma[i]$ then we add the top left corner.
- if $\sigma[i]$ is the leftmost element of σ' and $\overline{\sigma[i-1]\sigma[i]}$ or $\sigma[i]\urcorner$ then we add the top right corner.

Especially (remember that $\sigma(v)$ represent the subsequence embedded in the node v) we represent by $\widetilde{\sigma}(v)$ the bivincular pattern formed by the subsequence $\sigma(v)$.

The idea of the algorithm is to use the separable tree of σ to find an occurrence and a second trick: to find an occurrence of $\widetilde{\sigma}$, we need to find an occurrence for the (permutation embedded in) its left child and the (permutation embedded in) its right child, such that those two permutations are compatible which each other. More precisely, the occurrences must be contain in rectangles such that the rectangle of the occurrence of left child is on the left of the rectangle of the occurrence of the right child, and one has to be above the other depending whether σ is decomposed in direct or skew sum. Note that this is enough to find a occurrence for σ, but not $\widetilde{\sigma}$. Indeed the constraints on values and positions are not respected.

This is where the second trick comes handy: remark that given that $\widetilde{\sigma} = \widetilde{\sigma_\alpha} \oplus \widetilde{\sigma_\beta}$, if the topmost, bottommost, leftmost and rightmost elements of $\widetilde{\sigma_\alpha}$ (resp. $\widetilde{\sigma_\beta}$) are respectively in the top, bottom, left and right edges of the rectangle of the occurrence of $\widetilde{\sigma_\alpha}$ (resp. $\widetilde{\sigma_\beta}$) and if the two rectangle are consecutive in the x-coordinate (the first minimal rectangle end at x and the second minimal rectangle start at $x+1$), then the matching of the right most element of $\widetilde{\sigma_\alpha}$ and the matching of the left most element of $\widetilde{\sigma_\beta}$ are consecutive in index. In the same fashion, if the rectangles are consecutive in the y-coordinate (the first minimal rectangle end at y and the second minimal rectangle start at $y+1$), then the matching of the top most element of $\widetilde{\sigma_\alpha}$ and the matching of the bottom most element of $\widetilde{\sigma_\beta}$ are consecutive in value.

The above remark allow us to take care of the constraint value between the bottommost element of $\widetilde{\sigma_\beta}$ and the topmost element of $\widetilde{\sigma_\alpha}$ and the constraint position between the leftmost element of $\widetilde{\sigma_\beta}$ and the rightmost element of $\widetilde{\sigma_\alpha}$.

Moreover remark that an occurrence of $\widetilde{\sigma_\alpha}$ (resp. $\widetilde{\sigma_\beta}$) take care of all constraint on value and position of the elements in σ_α (resp.σ_β) except for it topmost and rightmost (resp. bottommost and leftmost) elements. So the only constraint left are on those elements, be we can take care of them by positioning the rectangle correctly.

In the following, if u is a leaf such that $\sigma[i] = \sigma(u)$ then we write $\sigma(u+1)$ to represent $\sigma[i+1]$ and $\sigma(u-1)$ to represent $\sigma[i-1]$.

The pattern matching problem with bivincular permutation can be solved by the following family of subproblems : For every node v of T_σ, for every two $i, j \in [n]$ with $i \leq j$, for every lower and upper bound $\mathrm{lb}, \mathrm{ub} \in [n]$ with $\mathrm{lb} \leq \mathrm{ub}$, which form a rectangle with left bottom corner (i, lb) and with right top corner (j, ub), we want to decide whether the rectangle contains an occurrence of $\widetilde{\sigma}(v)$, which give us the notation

$$\mathrm{PMB}_{v,i,j,\mathrm{lb},\mathrm{ub}} = \begin{cases} True & \text{if there exists an occurence of the bivincular pattern} \\ & \widetilde{\sigma}(v) \text{ in } \pi \text{ where every element are contains} \\ & \text{in the rectangle } ((i,\mathrm{lb}),(j,\mathrm{ub})). \\ False & \text{otherwise} \end{cases}$$

By abuse of language, we say that $\widetilde{\sigma}(v)$ occurs in $((i,\mathrm{lb}),(j,\mathrm{ub}))$. This problem can be solve by the following induction:

Base: If v is a leaf then :

$$\mathrm{PMB}_{v,i,j,\mathrm{lb},\mathrm{ub}} = \begin{cases} True & \text{If } \exists \iota \in [i,j] \text{ and } \pi[\iota] \in [lb, ub] \\ & \text{and if } \overline{\sigma(v)(\sigma(v)+1)} \text{ then } \pi[\iota] = ub \\ & \text{and if } \sigma(v)^{\urcorner} \text{ then } \pi[\iota] = ub = n \\ & \text{and if } \overline{(\sigma(v)-1)\sigma(v)} \text{ then } \pi[\iota] = lb \\ & \text{and if } {}^{\ulcorner}\sigma(v) \text{ then } \pi[\iota] = lb = 1 \\ & \text{and if } \underline{\sigma(v)\sigma(v+1)} \text{ then } \iota = j \\ & \text{and if } \sigma(v)_{\lrcorner} \text{ then } \iota = j = n \\ & \text{and if } \underline{\sigma(v-1)\sigma(v)} \text{ then } \iota = i \\ & \text{and if } {}_{\llcorner}\sigma(v) \text{ then } \iota = i = 1 \\ False & \text{otherwise} \end{cases}$$

A leaf occurs in $((i,\mathrm{lb}),(j,\mathrm{ub}))$ if and only if the rectangle is not empty. This is what the first condition is testing. The fourth next conditions assure that the matched element is on an edge of $((i,\mathrm{lb}),(j,\mathrm{ub}))$. For example if $\underline{\sigma(v)\sigma(v+1)}$ then the matched element must be on the right edge, and intuitively $\overline{\sigma(v+1)}$ will be on the left edge of the "next" rectangle.

Step. Here $i < j$ and $\mathrm{lb} < \mathrm{ub}$ and we let v_L and v_R stands for the left and right child of v respectively.

Suppose that $\sigma(v)$ occurs in $((i,\mathrm{lb}),(j,\mathrm{ub}))$, and that v is a positive node. So $\sigma(v) = \sigma(v_L) \oplus \sigma(v_R)$, in other words $\sigma(v)$ forms a stair up of two rectangles and so must the occurrence of $\sigma(v)$. So an occurrence of $\sigma(v)$ is composed by a left rectangle that contains the occurrence of $\sigma(v_L)$ and a right rectangle that contains the occurrence of $\sigma(v_R)$.

To find whether $\sigma(v)$ occurs in $((i,\mathrm{lb}),(j,\mathrm{ub}))$ we just have to find whether such occurrence of $\sigma(v_L)$ and $\sigma(v_R)$ exists. We can do so by trying every pair of such rectangle, but to reduce the number of pair to test and to control the

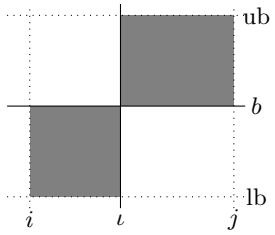


Fig. 10: What the pair of rectangle looks like.

position and value of the elements of the occurrence we require that the left rectangle share the same bottom edge and the same left edge as the rectangle $((i, \mathrm{lb}), (j, \mathrm{ub}))$ ie the left rectangle is $((i, \mathrm{lb}), (*, *))$, the right rectangle share the same top edge and the same right edge as rectangle $((i, \mathrm{lb}), (j, \mathrm{ub}))$ ie the right rectangle is $((*, *), (j, \mathrm{ub}))$ and that the left rectangle is consecutive in x and y coordinate to the right rectangle. In other words, we will try to find an occurrence of $\sigma(v_L)$ and $\sigma(v_R)$ in every pair of rectangle $((i, \iota - 1), (\mathrm{lb}, b - 1)$ and $((\iota, j), (b, \mathrm{ub}))$, see Figure 10. In notation this gives us:

$$\mathrm{PMB}_{v,i,j,\mathrm{lb},\mathrm{ub}} = \bigvee_{\substack{\iota \in (i,j] \\ \mathrm{b} \in (\mathrm{lb},\mathrm{ub}]}} \mathrm{PMB}_{v_L,i,\iota-1,\mathrm{lb},\mathrm{b}-1} \wedge \mathrm{PMB}_{v_R,\iota,j,\mathrm{b},\mathrm{ub}}$$

The case where v is a negative node can be dealt with symmetry:

$$\mathrm{PMB}_{v,i,j,\mathrm{lb},\mathrm{ub}} = \bigvee_{\substack{\iota \in (i,j] \\ \mathrm{b} \in (\mathrm{lb},\mathrm{ub}]}} \mathrm{PMB}_{v_L,i,\iota-1,\mathrm{b}-1,\mathrm{ub}} \wedge \mathrm{PMB}_{v_R,\iota,j,\mathrm{lb},\mathrm{b}}$$

If there is no condition on position nor on value, this algorithm solves the pattern matching problem: If $\sigma = \sigma_\alpha \oplus \sigma_\beta$, then every element of σ_α is left and below every element of σ_β. Moreover if there is an occurrence of σ_α and an occurrence of σ_β such that the elements of the occurrence of σ_α are left below the elements of the occurrence of σ_β then there exists an occurrence of σ in π. More formally we have to check every case possible, and to whether or not the element really occur on the edges.

For the constraints on position and value, intuitively, Whenever we have $\underline{\sigma(v)\sigma(v')}$,and w as the deepest ancestor of v and v', $\sigma(v)$ will be matched to the right edge of the left rectangle and $\sigma(v')$ will be matched to the left edge of the right rectangle so that $\sigma(v)$ and $\sigma(v')$ are consecutive in index. Likewise if $\underline{(\sigma(v))\sigma(v')}$, $\sigma(v)$ will be matched to the top edge of the left rectangle and $\sigma(v')$ will be matched to the bottom edge of the right rectangle so that $\sigma(v)$ and $\sigma(v')$ are consecutive in value. See Figure 12). More

Position Constraint. There are 3 types of position constraint added by bivincular permutation.

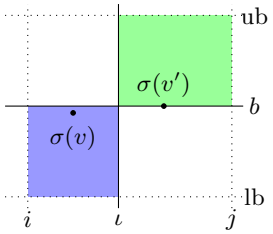


Fig. 11: The matching elements of $\sigma(v)$ and $\sigma(v')$ are consecutive in value if and only if $\sigma(v)$ is matched to $b-1$ and $\sigma(v')$ is matched to b.

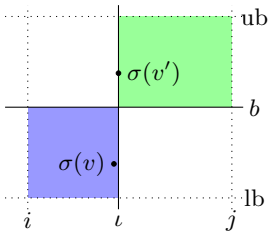


Fig. 12: The matching element of $\sigma(v)$ and $\sigma(v')$ are consecutive in position if and only if $\sigma(v)$ is matched to $\pi[\iota-1]$ and $\sigma(v')$ is matched to $\pi[\iota]$.

- If $\llcorner\sigma[1]$ then the leftmost element of σ must be matched to the leftmost element of π ($\sigma[1]$ occurs in $\pi[1]$ in an occurrence of σ in π). Remark that the leaf v such that $\sigma(v) = \sigma[1]$ is the leftmost ancestor of r_σ. Note that the rectangle of a left child shares the same left edge as the rectangle of his father (by induction), plus this is solved in the base case and the base case asks to the matched element to be on its left edge, finally note that the main problem has $x = 1$ for its left edge.
- The condition $\sigma[n_\sigma]\lrcorner$ follows the same idea as the condition $\llcorner\sigma[1]$.
- If $\underline{\sigma(v)\sigma(v')}$ (and thus $\sigma(v+1) = \sigma(v')$ and $\sigma(v) = \sigma(v'-1)$) then the index of the occurrence of $\sigma(v)$ and $\sigma(v')$ must be consecutive. In other words if $\sigma(v)$ occurs in $\pi[j]$ then $\sigma(v')$ must occur in $\pi[j+1]$. Let w be the first common ancestor of v and v' and w_L be the left child of w and w_R be the right child of w. First note that v is the rightmost ancestor of w_L and thus the rectangle of v shares the same right edge as the rectangle of w_L, plus v' is the leftmost ancestor w_R and thus the rectangle of v' shares the same left edge as the rectangle of w_R. Moreover those cases are solved as base cases and the base case asks to the element matching v to be on its right edge, and to the element matching v' to be on its left edge. Finally remark that the pair of rectangles of two brothers is consecutive in the x-coodinate (by induction).

Before diving into the value constraints, we highlight a property.

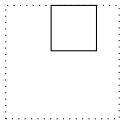


Fig. 13: if v occurs in the dotted rectangle then u occurs in the bold rectangle

Property 4. Given a node v and a leaf u, such that $\sigma(u)$ is the maximal (respectively minimal) element of $\sigma(v)$ then if v occurs in $((i, \mathrm{lb}), (j, \mathrm{ub}))$ then there exists $i \leq \iota \leq \iota' \leq j$, and $\mathrm{lb} \leq b \leq \mathrm{ub}$ and u occurs in $((\iota, b), (\iota', \mathrm{ub}))$ (resp. $((\iota, \mathrm{lb}), (\iota', b))$).

In other words if v occurs in a rectangle R and u is the leaf with the maximal element of $\sigma(v)$ then there exists an occurrence of u included in a rectangle that shares the same top edge as R (see fig 13). Especially if $\mathrm{PMB}_{v,i,j,\mathrm{lb},\mathrm{ub}}$ is true then $\mathrm{PMB}_{u,\iota,\iota',b,\mathrm{ub}}$ is true.

Proof. We focus on proving the assertion in the case where $\sigma(u)$ is the maximal element of $\sigma(v)$, the other case can be dealt by symmetry. Suppose that the assertion is false. If the rectangle of u can not have the same top edge as the rectangle of v it means that there exists a rectangle in between, in other words there exists a value bigger than $\sigma(u)$ which is not possible.

Value Constraint. There are 3 types of value constraint added by bivincular permutation.

- If $\ulcorner\sigma(v)$ (and thus $\sigma(v) = 1$) then the minimal value of σ must occur in the minimal value of π. By property 4, the rectangle of $\sigma(v)$ has the same bottom edge as the rectangle of $\sigma(r_\sigma)$, plus this is solved in the base case and the base case asks to the matching element to be on the bottom edge, finally the rectangle of r_σ has $y = 1$ as its bottom edge .
- The $\sigma(v)\urcorner$ follows the same idea as $\ulcorner\sigma(v)$.
- If $\overline{\sigma(v)\sigma(v')}$ (and thus $\sigma(v') = \sigma(v) + 1$) then the occurrence of $\sigma(v)$ and $\sigma(v')$ must be consecutive. In other words if $\sigma(v)$ occurs in $\pi[j]$ then $\sigma(v')$ must occur in $\pi[j] + 1$. Let w be the first common ancestor of v and v'. Let w_L be the left child of w and w_R be the right child of w. Suppose that w is positive then v is a child of w_L and v' is a child of w_R. First remark that $\sigma(v)$ is the the maximal element of $\sigma(w_L)$, otherwise $\sigma(v)$ and $\sigma(v')$ would not be consecutive. Thus by property 4, the rectangle of $\sigma(v)$ has the same top edge as the rectangle of $\sigma(w_L)$. Plus $\sigma(v')$ is the the minimal element of $\sigma(w_R)$, otherwise $\sigma(v)$ and $\sigma(v')$ would not be consecutive. Thus by property 4, the rectangle of $\sigma(v)$ has the same bottom edge as the rectangle of $\sigma(w_R)$. Moreover this is solved in the base case and the base case asks the element matching $\sigma(v)$ to be on its top edge and the element matching $\sigma(v')$ to be on its bottom edge. Finally remark that the pair of rectangles of two brothers are consecutive in the y-coordinate (by induction). We can prove the same by symmetry if w is negative.

So at the end of the algorithm we can decide whether $\widetilde{\sigma}$ occurs in π. Finally, the algorithm has kn^4 different cases, and each case try every pair of rectangle in constant time (by dynamic programming), so each case takes $O(n^2)$ time to solve. This gives us an $O(kn^6)$ time and $O(kn^4)$ space algorithm.

Acknowledgments

We thank the anonymous reviewers whose comments and suggestions helped improve and clarify this manuscript.

References

1. S. Ahal and Y. Rabinovich, *On Complexity of the Subpattern Problem*, SIAM Journal on Discrete Mathematics **22** (2008), no. 2, 629–649.
2. M.H. Albert, R.E.L. Aldred, M.D. Atkinson, and D.A. Holton, *Algorithms for pattern involvement in permutations*, Proc. International Symposium on Algorithms and Computation (ISAAC), Lecture Notes in Computer Science, vol. 2223, 2001, pp. 355–366.
3. D. Avis and M. Newborn, *On pop-stacks in series*, Utilitas Mathematica **19** (1981), 129–140.
4. P. Bille and I.L. Gørtz, *The tree inclusion problem: In linear space and faster*, ACM Trans. Algorithms **7** (2011), no. 3, 38.
5. P. Bose, J.F.Buss, and A. Lubiw, *Pattern matching for permutations*, Information Processing Letters **65** (1998), no. 5, 277–283.
6. M. Bouvel, D. Rossin, and S. Vialette, *Longest common separable pattern between permutations*, Proc. Symposium on Combinatorial Pattern Matching (CPM), London, Ontario, Canada (B. Ma and K. Zhang, eds.), Lecture Notes in Computer Science, vol. 4580, 2007, pp. 316–327.
7. M.-L. Bruner and M.Lackner, *A fast algorithm for permutation pattern matching based on alternating runs*, 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), Helsinki, Finland (F.V. Fomin and P. Kaski, eds.), Springer, 2012, pp. 261–270.
8. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to algorithms*, third ed., MIT Press, Cambridge, 2009.
9. S. Giraudo and S. Vialette, *Unshuffling permutations*, 12th Latin American Theoretical Informatics Symposium (LATIN), Lecture Notes in Computer Science, no. 9644, Springer, 2016, pp. 509–521.
10. S. Guillemot and D. Marx, *Finding small patterns in permutations in linear time*, Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Portland, Oregon, USA (C. Chekuri, ed.), SIAM, 2014, pp. 82–101.
11. L. Ibarra, *Finding pattern matchings for permutations*, Information Processing Letters **61** (1997), no. 6, 293–295.
12. P. Kilpeläinen and H. Manilla, *Ordered and unordered tree inclusion*, SIAM J. on Comput. **24** (1995), no. 2, 340–356.
13. S. Kitaev, *Patterns in permutations and words*, Springer-Verlag, 2013.
14. D. Rossin and M. Bouvel, *The longest common pattern problem for two permutations*, Pure Mathematics and Applications **17** (2006), 55–69.

15. V. Vatter, *Permutation classes*, Handbook of Enumerative Combinatorics (M. Bóna, ed.), Chapman and Hall/CRC, 2015, pp. 753–818.