Pattern matching for 321-avoiding permutations

Sylvain Guillemot and Stéphane Vialette

LIGM, CNRS UMR 8049, Université Paris-Est,
5 Bd Descartes 77454 Marne-la-Vallée, France
{guillemo,vialette}@univ-mlv.fr

Abstract. Given two permutations π and σ, the NP-complete Permutation Pattern problem is to decide whether π contains σ as a pattern. In case both π and σ are 321-avoiding, we prove the Permutation Pattern problem to be solvable in $O(k^2n^6)$ time, where $k = |\sigma|$ and $n = |\pi|$, and give a $O(kn^{4\sqrt{k}+12})$ time algorithm if only σ is 321-avoiding. Finally, we show W[1]-hardness of a 2-colored version of this latter problem.

1 Introduction

A permutation π is said to contain the pattern (shorter permutation) σ, in symbols $\sigma \preceq \pi$, if there exists a subsequence of entries of π that has the same relative order as σ (alternatively, σ is involved in π). Otherwise, π avoids σ. For example, 3215674 contains the pattern 132 since the subsequence 154 is ordered in the same way as 132. Pattern involvement in permutations has become a very active area of research. For one, pattern containment restrictions are often used to describe classes of permutations that are sortable under various conditions [10]. For another, a great deal of study has been devoted to counting pattern-avoiding permutations [4], culminating in the proof of the Stanley-Wilf conjecture [12].

We consider here the Permutation Pattern problem. Given two permutations σ and π, this problem is to decide whether $\sigma \preceq \pi$ (the problem is ascribed to H. Wilf in [5]). The Permutation Pattern problem is NP-hard [5], but is clearly polynomial-time solvable if σ has bounded size. Indeed, if σ has size k and π has size n, a straightforward brute-force algorithm solves the problem in $O(n^k)$ time. Improvements to this algorithm were presented in [2] and [1], the latter describing a $O(n^{0.47k+o(k)})$ time algorithm. Also, the problem is known to be polynomial-time solvable (in k and n) if σ is *separable*, *i.e.*, σ contains neither the pattern 2413 nor 3142 [5,9]. In case σ is monotone, *i.e.*, $\sigma = 1 \dots k$ or $\sigma = k \dots 1$, a $O(n \log \log n)$ time algorithm is known [8].

We focus in this paper on the Permutation Pattern problem in case σ (possibly σ and π) avoids a pattern of length 3. Recall that Knuth proved in [11] that for all six of the patterns of length 3 it is true that the number of permutations of size n that avoid the pattern is the Catalan number $C_n = \binom{2n}{n}/(n+1)$. First, it is easy to see that the Permutation Pattern problem is polynomial-time solvable if the pattern σ avoids 132, 312, 213 or 231 since σ is clearly separable in this case. Monotone patterns, *i.e.*, 123 and 321, however, deserve separate consideration (we focus here on 321-avoiding permutations but if a permutation avoids 123 then its reverse avoids 321). In case both π and σ are 321-avoiding, we prove the Permutation Pattern problem to be solvable in $O(k^2n^6)$ time, where $k = |\sigma|$ and $n = |\pi|$, and give a $O(kn^{4\sqrt{k}+12})$ time algorithm if only σ is 321-avoiding. Finally, we show $\mathsf{W}[1]$-hardness of a 2-colored version of this latter problem.

This paper is organized as follows. Section 2 briefly reviews the needed material and some basic properties are derived. We consider in Section 3 the Permutation Pattern problem in case both σ and π are 321-avoiding whereas Section 4 is devoted to the Permutation Pattern problem in case only σ is 321-avoiding.

2 Generalities

2.1 Permutation patterns

We will use two different representations of permutations. The usual *array representation* of π is simply the sequence $\pi(1)\ldots\pi(n)$. A second representation describes a permutation π by its graph, *i.e.*, a set of points in the plane where no two two points are aligned horizontally or vertically. We will write $p \in \pi$ to mean that p is a point in the graph of π, and the x- and y-coordinates of p will be denoted by $x(p)$ and $y(p)$, respectively.

We define the notion of pattern containment for permutations in terms of *embeddings*, which are order-preserving mappings between two permutations.

Definition 1 (Embedding). *Given two permutations σ and π, an* embedding *of σ into π is an injective mapping $\phi : \sigma \to \pi$ which is order-preserving for x-coordinates and y-coordinates,* i.e.*, for any two points $p, p' \in \sigma$, it holds that: (i) $x(p) < x(p') \Rightarrow x(\phi(p)) < x(\phi(p'))$, and (ii) $y(p) < y(p') \Rightarrow y(\phi(p')) < y(\phi(p'))$.*

When there exists an embedding of σ into π, we say that σ *occurs* in π, denoted by $\sigma \preceq \pi$. We also say that π *contains* σ; when this does not hold, we say that π *avoids* σ.

A permutation is *k-increasing* if and only if it can be partitioned in k increasing subsequences. It is well-known that a permutation π is k-increasing if and only if its longest decreasing subsequence has length at most k, or equivalently if π avoids $k+1\ldots 1$. We focus here on 2-increasing permutations, which are also characterized as 321-avoiding permutations. These permutations were introduced by [10] as *queue-sortable permutations*, *i.e.*, permutations which can be sorted by a queue (with shortcuts). This characterization yields a linear-time recognition algorithm for this class, which can be turned into a certifying algorithm.

Proposition 1. *Given a permutation π of size n, in $O(n)$ time we can decide if π is 2-increasing, and (i) output a partition in two increasing subsequences (as a positive certificate), and (ii) or output an occurrence of* 321 *into π (as a negative certificate).*

2.2 Stair-decompositions

We introduce here the notion of *stair-decomposition* that will play an important role in our study of 2-increasing permutations. Note that a similar notion was used in [3].

Definition 2 (Stair-decomposition). *A* stair-decomposition *of a permutation π consists of: (i) a partition of the horizontal axis in intervals $I_1, \ldots, I_k$, and (ii) a partition of the vertical axis in intervals $J_1, \ldots, J_k$, such that: if we let $S_{i,j}$ be the square at the intersection of intervals I_i and J_j, then*

- *the graph of π contains points only in the squares $S_{i,i}$ or $S_{i,i+1}$;*
- *the points of π inside a nonempty square form an increasing subsequence.*

A stair-decomposition $\mathcal{D}$ can also be described by its *blocks* $B_1, \ldots, B_k$, where for each i, B_{2i-1} is the points inside $S_{i,i}$, and B_{2i} is the points inside $S_{i,i+1}$. Figure 1 illustrates the definition.

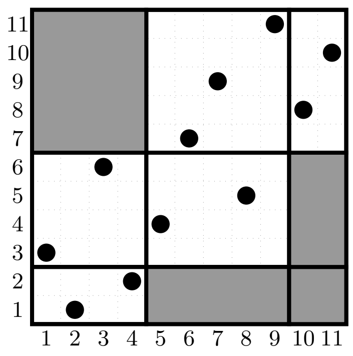

Fig. 1. A stair-decomposition of the permutation $\pi = 3\ 1\ 6\ 2\ 4\ 7\ 9\ 5\ 11\ 8\ 10$. The white blocks on the two diagonals contain increasing subsequences, while the grey blocks contain no point.

It is not difficult to prove that the existence of a stair-decomposition fully characterizes the 2-increasing permutations.

Lemma 1. *π is 2-increasing if and only if π has a stair-decomposition.*

Let $X = \{(i, i+1) : i \in \mathbb{N}^*,\ i \text{ odd}\}$ and $Y = \{(i, i+1) : i \in \mathbb{N}^*,\ i \text{ even}\}$. The following Lemma gives an equivalent characterization of stair-decompositions.

Lemma 2. *Let π be a permutation. Let $\mathcal{D}$ be a partition of π in $B_1, \ldots, B_k$. Then $\mathcal{D}$ is a stair-decomposition if and only if: (i) for each i, B_i forms an increasing subsequence, (ii) for each $p \in B_i, p' \in B_j$, $i < j$ and $(i, j) \notin X$, it holds that $x(p) < x(p')$, and (iii) for each $p \in B_i, p' \in B_j$, $i < j$ and $(i, j) \notin Y$, it holds that $y(p) < y(p')$.*

3 Both σ and π are 2-increasing

3.1 Preliminaries

We first introduce the notions of *ordered preforests* and *ordered forests.*

Definition 3 (Ordered preforest). *An* ordered preforest *consists of (i) a forest F with node set $N(F)$, (ii) a* depth *$d_F(u) \in \mathbb{N}^*$ assigned to each node $u \in N(F)$, (iii) a total order $<_F$ on $N(F)$ satisfying the following conditions. For each i, the* level *L_i of F is the set of nodes of depth i. We then require that:*

1. *for each $v \in N(F)$ with parent u, it holds that $d_F(v) = d_F(u) + 1$;*
2. *$<_F$ ranks nodes by increasing depth, i.e. if $d_F(u) < d_F(v)$ then $u <_F v$;*
3. *for each $u \in N(F)$, the children of u form an interval of $<_F$.*

Condition 3 amounts to say that there is a plane drawing of F such that (i) all nodes of L_i have their y-coordinate equal to i, (ii) in this drawing, $<_F$ orders the nodes of L_i from left to right. The total order $<_F$ can be seen as a breadth-first traversal of F.

Given $u \in N(F)$, we denote by $parent_F(u)$ its parent node, and by $children_F(u)$ its set of child nodes. If all roots of F have depth 1, then F is called an *ordered forest.* If F is an ordered preforest and if $i \in \mathbb{Z}$, $F \uparrow i$ denotes the ordered preforest which contains the nodes of F of depth $\geq i+1$, and where each such node has new depth $d_{F'}(u) = d_F(u) - i$.

We will also need the notion of *split* of an ordered forest. Suppose that F is an ordered forest with n nodes. Let $x = x_1 \dots x_n$ be the depth-first traversal of F (defined in the standard way). A *split* of F is a partition A, B of $N(F)$ s.t. A forms a prefix of x, and B forms a suffix of x. Note that the splits of F are in number $n+1$. From the ordered forest F, we can then define $F[A], F[B]$ which are ordered preforests.

3.2 Representation of 2-increasing permutations

In this section, we describe a way to represent 2-increasing permutations by ordered forests.

Let F be an ordered forest with n nodes. Let $L_1, \dots, L_h$ be the levels of F. The *anchor* of F is the leftmost node of L_1. An *odd-alternating traversal* of F is a traversal of F which proceeds as follows: start with $i = 1$; while $i \leq h$, enumerate the elements of $L_i \cup L_{i+1}$ by a depth-first traversal, and increment i by 2. An *even-alternating traversal* of F is defined in a similar way, except that it starts at $i = 0$ ($L_0 = \emptyset$ by convention).

To the ordered forest F, we associate a permutation π_F of size $n-1$, defined as follows: (i) label the nodes of F with increasing x-coordinates from 0 to $n-1$, by an odd-alternating traversal of F; (ii) label the nodes of F with increasing y-coordinates from 0 to $n-1$, by an even-alternating traversal of F; (iii) ignore the anchor of F, and consider the $n-1$ remaining points as the graph of a permutation π_F. This process is illustrated in Figure 2.

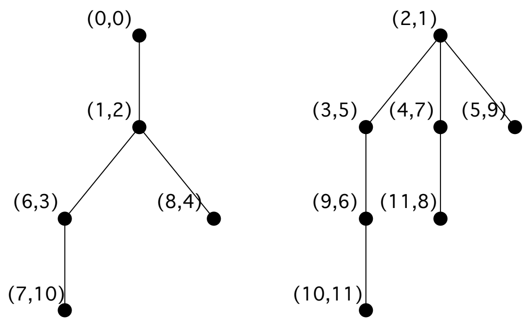

Fig. 2. An ordered forest F; the labeling of its nodes defines the permutation $\pi_F =$ 2 1 5 7 9 3 10 4 6 11 8.

Suppose that each node p of F has been labeled by x- and y-coordinates, denoted by $x(p)$ and $y(p)$. The following Lemma characterizes the relative positioning of two points in π_F, in terms of their relationships in F.

Lemma 3. *Suppose that $p \in L_i, p' \in L_j$, $i \leq j$.*

1. *If $(i,j) \notin X$, then $x(p) < x(p')$ if and only if $p <_F p'$;*
2. *If $(i,j) \notin Y$, then $y(p) < y(p')$ if and only if $p <_F p'$;*
3. *If $(i,j) \in X$, then $x(p) < x(p')$ if and only if $p \leq_F parent_F(p')$;*
4. *If $(i,j) \in Y$, then $y(p) < y(p')$ if and only if $p \leq_F parent_F(p')$.*

The following Lemma shows that a stair-decomposition of π_F can be obtained from the ordered forest F; it implies that the construction always produces a 2-increasing permutation.

Lemma 4. *The sets $L_1, \ldots, L_h$ form a stair-decomposition of π_F.*

As stated above, this result implies that π_F is 2-increasing, in virtue of Lemma 1. The following proposition shows that every 2-increasing permutation can be represented in this way (proof in Appendix).

Proposition 2. *For any 2-increasing permutation π of size n, there exists an ordered forest F with $n+1$ nodes such that $\pi = \pi_F$.*

3.3 Embeddings

In this section, we introduce notions of embedding for ordered forests, and we characterize embeddings of 2-increasing permutations in terms of embeddings of their associated forests (Proposition 3).

Definition 4 (Embedding). *Let F be an ordered preforest with levels $L_1, \ldots, L_h$, and let F' be an ordered preforest with levels $L'_1, \ldots, L'_{h'}$. An* embedding *of F into F' is an injective mapping $\phi : N(F) \to N(F')$ such that for each $1 \leq i \leq h$:*

1. $\phi(L_i) \subseteq L'_i$;
2. *if* $p, p' \in L_i$, *then* $p <_F p' \Leftrightarrow \phi(p) <_{F'} \phi(p')$;
3. *if* $p \in L_i$ *and* $p' \in L_{i+1}$, *then* $p \leq_F parent_F(p') \Leftrightarrow \phi(p) \leq_{F'} parent_{F'}(\phi(p'))$.

In words, the definition requires that ϕ maps points of F to points of F' on the same level (Point 1), preserves the ordering between points on a same level (Point 2), and preserves the parent-child ordering between points on consecutive levels (Point 3). We write $F \preceq F'$ iff there exists an embedding of F into F'.

We now define a second notion of embedding called *twist embedding*.

Definition 5 (Twist embedding). *We say that there exists a* twist embedding *of F into F' (denoted by $F \trianglelefteq F'$) iff there exists A, B split of F and A', B' split of F' s.t. (i) $F[A] \preceq F'[B']$ and (ii) $F[B] \preceq F'[A'] \uparrow 2$.*

Given two permutations π_1, π_2, their sum $\pi_1 \oplus \pi_2$ is the permutation π whose graph can be partitioned in two rectangles R_1, R_2, with R_2 above and at the right of R_1, s.t. $\pi|R_1 = \pi_1, \pi|R_2 = \pi_2$. We say that π is *simple* iff it cannot be written as $\pi_1 \oplus \pi_2$ (where π_1, π_2 are not empty). The following proposition characterizes embeddings of 2-increasing permutations in terms of twist embeddings of their associated forests; this result is illustrated in Figure 3.

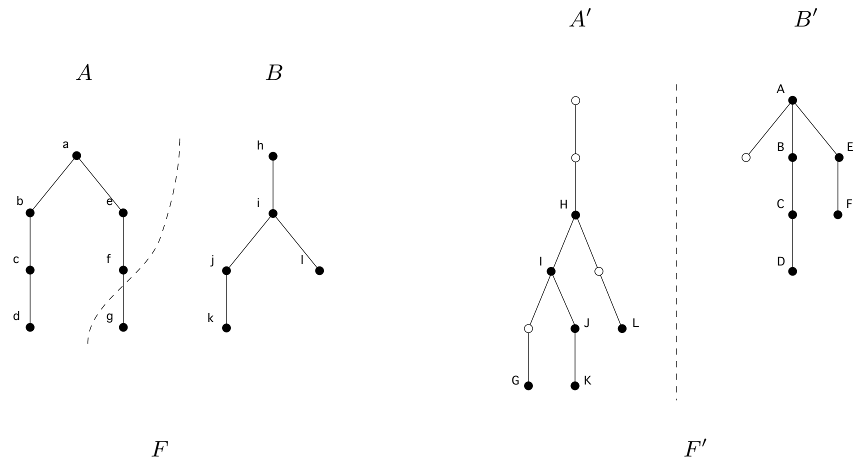

Fig. 3. The ordered forest F, and its bipartition A, B, is represented on the left, with its vertices named by lowercase letters. The ordered forest F', and its bipartition A', B', is represented on the right. The vertices of F' denoted by capital letters are the images of the vertices of F under a twist embedding. Note that $\pi_F = 2\ 4\ 1\ 6\ 3\ 9\ 5\ 10\ 7\ 11\ 8$ and that $\pi_{F'} = 2\ 1\ 4$ **5 7 3 9** 12 **6 14 8** 10 **15 11 16 13**, where the bold positions of $\pi_{F'}$ correspond to the black points of F' and form an occurrence of π_F.

Proposition 3. *Let F, F' be two ordered forests, and let h be the height of F'. Suppose that π_F and $\pi_{F'}$ are simple. Then: $\pi_F \preceq \pi_{F'}$ iff there exists $0 \leq i \leq h$ s.t. $F \trianglelefteq F' \uparrow i$.*

3.4 Finding an embedding

In this section, we describe a polynomial-time algorithm to decide if $F \trianglelefteq F'$ (Proposition 5). By Proposition 3, this will yield a polynomial-time algorithm for the Permutation Pattern problem for 2-increasing permutations (Theorem 1). For the purpose of deciding the embedding relation between forests, we need a result on *labeled dags* (Proposition 4). Before stating this result, we introduce the following definitions.

Definition 6 (Labeled dag). *Let Σ be an alphabet, and let D be a symmetric reflexive relation over Σ. A Σ, D-labeled dag consists in a dag $G = (V, A)$, together with a labeling $\lambda : V \to \Sigma$, having the following property: for each $u, v \in V$ distinct, u, v are adjacent in G iff $(\lambda(u), \lambda(v)) \in D$.*

Definition 7 (Morphism). *Let G, G' be two Σ, D-labeled dags. A* morphism *of G into G' is an injective mapping $\phi : V(G) \to V(G')$ such that: (i) ϕ is arc-preserving: for each $u \in V(G)$, $(u, v) \in A(G) \Leftrightarrow (\phi(u), \phi(v)) \in A(G')$; (ii) ϕ is label-preserving: for each $u \in V(G)$, u and $\phi(u)$ have the same label.*

We denote $G \preceq G'$ iff there exists a morphism of G into G'. It turns out that this property can be efficiently decided, thanks to an algorithm based on topological sorting.

Proposition 4. *Given two Σ, D-labeled dags G and G', we can decide in polynomial time if $G \preceq G'$.*

Proof (Sketch). We first perform a topological sorting of G', which gives us a linear extension of G'. Let $y = y_1 \dots y_n$ be the labeling of this linear extension.

We now perform a topological sort of G as follows. We maintain Min the set of minimal elements in G, and j a position in y; initially $j = 0$. At each step of the topological sort, we need to pick the next element in Min. We proceed as follows: (i) for each $x \in Min$ labeled by a, we compute m_x as the first position $i > j$ s.t. $y_j = a$ (if it exists), or ∞ otherwise; (ii) if all values m_x are ∞, we stop and we answer negatively. Otherwise, we choose the element $x \in Min$ with minimum m_x, and we update $j \leftarrow m_x$. Note that there is a unique choice for x at each step, since no two elements of Min can have the same label. When the algorithm completes the topological sort, it answers positively.

The algorithm clearly runs in polynomial-time, and its correctness proof is deferred to the Appendix. □

In fact, a more involved algorithm can solve the problem in linear $O(|G|+|G'|)$ time, we will present it into the journal version.

The following Proposition shows how these results allow us to find embeddings (Point 1) and twist-embeddings (Point 2) between two forests.

Proposition 5. *Let F and F' be two ordered forests, with $|F| = k$ and $|F'| = n$. Then:*

1. *We can decide in $O(n)$ time if $F \preceq F'$.*
2. *We can decide in $O(kn^2)$ time if $F \trianglelefteq F'$.*

Proof. Point 1. Suppose that F has levels $L_1, \ldots, L_h$ and that F' has levels $L'_1, \ldots, L'_h$. Let $\Sigma = \{1, \ldots, h\}$ and let D be the set of pairs $(i,i), (i, i+1)$. We represent F by a labeled dag G_F as follows: (i) G_F has vertex set $N(F)$, (ii) each vertex of G_F is labeled by its depth in F, (iii) for each $u, v \in L_i$, G_F contains an arc (u,v) iff $u <_F v$, (iv) for each $u \in L_i, v \in L_{i+1}$, G_F contains an arc (u,v) iff $u \leq_F parent_F(v)$. It is readily seen from Definitions 4 and 7 that embeddings of F into F' correspond to morphisms of G_F into $G_{F'}$. It follows that $F \preceq F'$ iff $G_F \preceq G_{F'}$, which can be decided in $O(|G_F| + |G_{F'}|)$ time by the above algorithm. Note that G_F may have size $O(k^2)$ and that $G_{F'}$ may have size $O(n^2)$. However, we can apply the above algorithms to their *transitive reductions*, whose size is now $O(k)$ and $O(n)$, and which can be constructed in linear time from F and F'. This is correct since G_F and its transitive reduction has the same linear extensions. We therefore obtain a $O(k + n) = O(n)$ time algorithm to decide if $F \preceq F'$.

Point 2. It follows from Definition 5 that to decide if $F \trianglelefteq F'$, we need to examine every split $S = A, B$ of F, every split $S' = A', B'$ of F', and in each case to test if $F[A] \preceq F'[B']$ and $F[B] \preceq F'[A'] \uparrow 2$. For a given S, S', the two tests are carried out in $O(n)$ time by Point 1. Now, the possible S are in number $k+1$ and the possible S' are in number $n+1$, leading to the claimed $O(kn^2)$ running time. □

Propositions 3 and 5 yield a polynomial-time algorithm for the PERMUTATION PATTERN problem for 2-increasing permutations (proof in Appendix).

Theorem 1. *Let σ, π be two 2-increasing permutations, with $|\sigma| = k$ and $|\pi| = n$. We can decide in $O(k^2 n^6)$ time if $\sigma \preceq \pi$.*

4 Only σ is 2-increasing

In this section, we consider the restriction of the PERMUTATION PATTERN problem to the case when σ is 2-increasing but π is arbitrary. For convenience, we define the problem so that a stair-decomposition of σ is also given as input.

Name: 2-INCREASING PERMUTATION PATTERN (2IPP)
Input: a 2-increasing permutation σ of size k, a stair-decomposition $\mathcal{D}$ of σ with r blocks, each of size at most s, and a permutation π of size n.
Question: Decide if $\sigma \preceq \pi$.

4.1 Algorithms

In this section, we show that the 2IPP problem can be solved in $n^{O(\sqrt{k})}$ time. The algorithm is by combining two different strategies for solving the problem.

We will describe a first strategy which solves the problem in $n^{O(s)}$ time, and a second strategy which solves the problem in $n^{O(r)}$ time. Recall that s is the maximum size of the blocks of $\mathcal{D}$, and that r is the number of blocks of $\mathcal{D}$. A combination of both strategies will yield an algorithm with the claimed $n^{O(\sqrt{k})}$ running time.

The first strategy uses a simple dynamic-programming approach. It stems from the observation that to find an embedding of σ into π, it suffices to find embeddings ϕ_i of B_i into π (for every i), and to check the consistency between pairs of consecutive embeddings ϕ_i, ϕ_{i+1}. To formalize this idea, we need the following definitions. Given a block B_i, a *partial solution* for B_i is a pair $S = (R, \psi)$, where $R = (x_1, y_1, x_2, y_2)$ is a rectangle in the graph of π, and ψ is an embedding of B_i into π whose image is included into R (*i.e.*, for each $p \in B_i$, it holds that $x_1 \leq x(\psi(p)) < x_2$ and $y_1 \leq y(\psi(p)) < y_2$). Intuitively, a partial solution specifies the restriction of an embedding to B_i, as well as the region to which the points of B_j $(j \geq i)$ are mapped. Let $\mathcal{S}_i$ denote the set of partial solutions for B_i.

Definition 8. *Given a partial solution $S = (R, \psi) \in \mathcal{S}_i$ with $R = (x_1, y_1, x_2, y_2)$, we define the predicate $\Pi(i, S)$ to hold if and only if there exists an embedding ϕ of $B_i \cup \ldots \cup B_r$ into π such that: (i) $\phi|B_i = \psi$; (ii) if $p \in B_j$ with $j > i$ then:*

- *if $j > i+1$: then $x(\phi(p)) \geq x_2$ and $y(\phi(p)) \geq y_2$;*
- *if $j = i+1$ and i odd: then $x(\phi(p)) \geq x_1$ and $y(\phi(p)) \geq y_2$;*
- *if $j = i+1$ and i even: then $x(\phi(p)) \geq x_2$ and $y(\phi(p)) \geq y_1$.*

The predicates $\Pi(i, S)$ can be computed by dynamic programming thanks to the following Lemma. Given partial solutions $S = (R, \psi) \in \mathcal{S}_i$ and $S' = (R', \psi') \in \mathcal{S}_{i+1}$, with $R = (x_1, y_1, x_2, y_2)$, and $R' = (x'_1, y'_1, x'_2, y'_2)$, say that S and S' are *compatible* if and only if:

- either B_i, B_{i+1} are on the same column (i odd), and it holds that: $x'_1 = x_1$, $x'_2 = x_2$, $y'_1 = y_2$, and for any $p \in B_i$, $p' \in B_{i+1}$, $x(p) < x(p') \Leftrightarrow x(\psi(p)) < x(\psi'(p'))$, or
- B_i, B_{i+1} are on the same row (i even), and it holds that: $y'_1 = y_1$, $y'_2 = y_2$, $x'_1 = x_2$, and for any $p \in B_i$, $p' \in B_{i+1}$, $y(p) < y(p') \Leftrightarrow y(\psi(p)) < y(\psi'(p'))$.

Lemma 5. *Suppose that $i < r$, and let $S \in \mathcal{S}_i$. Then: $\Pi(i, S)$ holds if and only if there exists $S' \in \mathcal{S}_{i+1}$ compatible with S such that $\Pi(i+1, S')$ holds.*

The above lemma yields a polynomial-time algorithm for the 2IPP problem when s is bounded.

Proposition 6. 2IPP *is solvable in $O(rn^{2s+5})$ time.*

Proof. By dynamic-programming, we compute the predicates $\Pi(i, S)$ for each $1 \leq i \leq r$ and each $S \in \mathcal{S}_i$, using Lemma 5. An element of $\mathcal{S}_i$ consists of a rectangle R and of an embedding ψ; there are $O(n^4)$ choices for R and $O(n^s)$ choices for ψ (since $|B_i| \leq s$). It follows that each $\mathcal{S}_i$ has size $O(n^{s+4})$. Consider now the running time needed to compute a value $\Pi(i, S)$ assuming that the values $\Pi(i+1, S')$ are available. By Lemma 5, we need to examine every $S' = (R', \psi') \in \mathcal{S}_{i+1}$ compatible with S. There are $O(n)$ choices for R' (since three of the four coordinates are determined by R), and $O(n^s)$ choices for ψ'. It follows that computing any $\Pi(i, S)$ is $O(n^{s+1})$ time. Therefore, the algorithm, as a whole, runs in $O(rn^{s+4}n^{s+1}) = O(rn^{2s+5})$ time. □

The second strategy solves the problem in $n^{O(r)}$ time, where r is the number of blocks in the stair-decomposition $\mathcal{D}$. For the sake of clarity, we formulate our approach as a non-deterministic algorithm.

This is an extension of the following idea. If σ is increasing, then we can find an embedding of σ into π by the following simple non-deterministic algorithm: (i) choose a point p_1 in π; (ii) for $i = 2$ to $|\sigma|$, choose a point p_i in π such that $x(p_i) > x(p_{i-1})$ and $y(p_i) > y(p_{i-1})$. The key idea in this algorithm is that we only need to memorize the image of the last point of σ examined so far.

We will extend this idea to the case where σ has a stair-decomposition with r blocks: in this case, in each block B_i the points will be examined from left to right, and we will memorize the image of the *first* and *last* point in each row and each column of $\mathcal{D}$. At the time of considering a new point in B_i, we will choose an image for this point in π, and check that this image is well-positioned with respect to the last points of the current row and column, and with respect to the first points of the next row and column.

We need to specify the order in which the points of σ will be examined. This leads us to the following definition.

Definition 9. *An* insertion order *for σ with respect to $\mathcal{D}$ is an enumeration of the points of σ such that: (i) on each row, the points are ordered by increasing y-coordinate, and (ii) on each column, the points are ordered by increasing x-coordinate.*

Lemma 6. *There exists an insertion order for σ with respect to $\mathcal{D}$.*

Let us show how this notion of insertion order allows us to solve the problem. Suppose that $s = p_1 \dots p_k$ is an insertion order for σ with respect to $\mathcal{D}$. Define a *configuration* to be a tuple C consisting of: (i) two values firstr_i and lastr_i (which may be equal to $\bot$) for each row i, and (ii) two values $\mathrm{firstc}_i, \mathrm{lastc}_i$ for each column i. We then find an embedding of σ into π using (non-deterministic) Algorithm 1. This yields a polynomial-time algorithm for the 2IPP problem when the number of blocks r is bounded as shown in the following proposition.

Proposition 7. 2IPP *is solvable in $O(kn^{2r+3})$ time.*

Proof. We perform a deterministic simulation of the non-deterministic Algorithm 1. To this aim, for each configuration C and each $0 \leq i \leq k$, we compute

Algorithm 1 A non-deterministic algorithm for the 2IPP problem

```
1: choose values $\mathrm{firstr}_i, \mathrm{firstc}_i$ in $\{1,\dots,n\}$ such that for each $i$,
2:     $\mathrm{firstr}_i \leq \mathrm{firstr}_{i+1}$, $\mathrm{firstc}_i \leq \mathrm{firstc}_{i+1}$;
3: initialize the values $\mathrm{lastr}_i$ and $\mathrm{lastc}_i$ to $\perp$;
4: for $h = 1$ to $k$ do
5:     if $p_h$ belongs to row $i$, column $j$ of $\mathcal{D}$, then:
6:     choose $p$ point of $\pi$;
7:     check that $x(p) < \mathrm{firstc}_{j+1}$ and $y(p) < \mathrm{firstr}_{i+1}$;
8:     if $\mathrm{lastr}_i = \perp$, check that $y(p) = \mathrm{firstr}_i$; if $\mathrm{lastr}_i \neq \perp$, check that $y(p) > \mathrm{lastr}_i$;
9:     if $\mathrm{lastc}_j = \perp$, check that $x(p) = \mathrm{firstc}_j$; if $\mathrm{lastc}_j \neq \perp$, check that $x(p) > \mathrm{lastc}_j$;
10:    update $\mathrm{lastc}_j \leftarrow x(p)$, $\mathrm{lastr}_i \leftarrow y(p)$.
11: end for
12: accept at the end of the algorithm.
```

a predicate $P(i, C)$ which holds if and only if there exists of an execution of the algorithm which reaches configuration C at the beginning of loop $h = i+1$. The predicates $P(i, C)$ are then computed by induction on i; (i) obtaining $P(0, C)$ takes $O(r)$ time; (ii) for $0 \leq i < k$, if the values $P(i, C)$ have been previously computed then we can deduce the values $P(i+1, C')$, where each value $P(i, C)$ gives rise to n values $P(i+1, C')$ (since we need to examine every possible choice of p in Line 6). Finally, we accept if and only if $P(k, C)$ holds for at least one C.

The claimed running time follows by observing that, if $\mathcal{D}$ has p rows and q columns, then the number of configurations is bounded by $n^{2(p+q)}$ and that $p + q \leq r + 1$. Hence, the predicates $P(i, C)$ are computed in $O(knn^{2(r+1)}) = O(kn^{2r+3})$ time. □

We have thus presented two strategies for solving the 2IPP problem, one in $n^{O(s)}$ time and the other in $n^{O(r)}$ time. By combining these two strategies in a nontrivial way, we achieve a $n^{O(\sqrt{k})}$ running time for solving 2IPP, as stated in the following Theorem (whose proof is given in Appendix).

Theorem 2. 2IPP *can be solved in* $O(kn^{4\sqrt{k}+12})$ *time.*

4.2 Hardness results

While we are still unable to settle the parameterized complexity of the 2IPP problem, we can show hardness results for a slight generalization of the problem. We consider the following c-colored version of the 2IPP problem: given σ, $\mathcal{D}$ and π where σ and π are c-colored permutations (*i.e.*, a color in $[c]$ is associated to each point of the permutation), find a color-preserving embedding of σ into π. We call this problem c-Colored 2IPP. It is not difficult to see that the algorithmic results of the previous section can be adapted to this colored version. When $c \geq 2$, we are able to prove the asymptotic optimality of these algorithms, conditioned by the Exponential-Time Hypothesis (ETH). Recall that ETH is the assumption that 3-Sat cannot be solved in $2^{o(n)}$ time (where n is the number of variables). It was shown in [6] that Clique cannot be solved in $n^{o(k)}$ time assuming ETH.

Theorem 3. *(i)* 2-Colored 2IPP *parameterized by k is* W[1]*-hard and cannot be solved in $n^{o(\sqrt{k})}$ time assuming ETH. (ii)* 2-Colored 2IPP *parameterized by s is* WNL*-hard and cannot be solved in $n^{o(s)}$ time assuming ETH.*

The parameterized class WNL was introduced in [7] to capture the parameterized complexity of problems solvable by k-dimensional dynamic programming.

5 Conclusion

We mention here some directions of interest for future works. The computational complexity of the Permutation Pattern problem in case only σ is 321-avoiding is still unknown (we have proved the problem to be polynomial-time solvable if both σ and π are 321-avoiding). We conjecture this problem to be NP-hard. Also, improving the time complexity of Theorem 2 is of particular interest. At a more general level, it remains a challenging problem to determine the parameterized complexity of the Permutation Pattern problem for parameter $|\sigma|$.

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), LNCS, vol. 2223, Springer-Verlag, 2001, pp. 355–366.
3. M.D. Atkinson, M.M. Murphy, and N. Ruskuc, *Pattern Avoidance Classes and Subpermutations*, Electronic Journal of Combinatorics **12** (2005), no. 1.
4. M. Bona, *Combinatorics of permutations*, Discrete Mathematics and Its Applications, Chapman-Hall and CRC Press, 2004.
5. P. Bose, J.F. Buss, and A. Lubiw, *Pattern Matching for Permutations*, Information Processing Letters **65** (1998), no. 5, 277–283.
6. J. Chen, B. Chor, M. Fellows, X. Huang, D. Juedes, I. Kanj, and G. Xia, *Tight lower bounds for certain parameterized NP-hard problems*, Proceedings of 19th Annual IEEE Conference on Computational Complexity (CCC), 2004, pp. 150–160.
7. S. Guillemot, *Parameterized complexity and approximability of the SLCS problem*, Proc. International Workshop on Parameterized and Exact Computation (IWPEC), LNCS, vol. 5018, Springer-Verlag, 2008, pp. 115–128.
8. J. Hunt and T. Szymanski, *A fast algorithm for computing longest common subsequences*, Communications of the ACM **20** (1977), 350–353.
9. L. Ibarra, *Finding pattern matchings for permutations*, Information Processing Letters **61** (1997), no. 6, 293–295.
10. D.E. Knuth, *Fundamental Algorithms, The Art of Computer Programming Vol. 1 (Second Edition)*, Addison-Wesley, 1973.
11. ———, *Sorting and Searching, The Art of Computer Programming Vol. 3 (Second Edition)*, Addison-Wesley, 1998.
12. A. Marcus and G. Tardos, *Excluded permutation matrices and the Stanley-Wilf conjecture*, Journal of Combinatorial Series A **107** (2004), no. 1, 153–160.