

The computational landscape of permutation patterns*

Marie-Louise Bruner and Martin Lackner

marie-louise.bruner@tuwien.ac.at
lackner@dbai.tuwien.ac.at

Vienna University of Technology

Abstract

In the last years, different types of patterns in permutations have been studied: vincular, bivincular and mesh patterns, just to name a few. Every type of permutation pattern naturally defines a corresponding computational problem: Given a pattern P and a permutation T (the text), is P contained in T ? In this paper we draw a map of the computational landscape of permutation pattern matching with different types of patterns. We provide a classical complexity analysis and investigate the impact of the pattern length on the computational hardness. Furthermore, we highlight several directions in which the study of computational aspects of permutation patterns could evolve.

Mathematics subject classification: 05A05, 68Q17, 68Q25.

1 Introduction

The systematic study of permutation patterns was started by Simion and Schmidt [29] in the 1980s and has since then become a bustling field of research in combinatorics. The core concept is the following: A permutation P is contained in a permutation T as a pattern if there is an order-preserving embedding of P into T . For example, the pattern 132 is contained in 2143 since the subsequence 243 is order-isomorphic to 132. In recent years other types of permutation patterns have received increased interest, such as vincular [5], bivincular [8], mesh [9], boxed mesh [4] and consecutive patterns [13] (all of which are introduced in Section 3).

Every type of permutation pattern naturally defines a corresponding computational problem. Let \mathcal{C} denote any type of permutation pattern, i.e., let $\mathcal{C} \in \{\text{classical, vincular, bivincular, mesh, boxed mesh, consecutive}\}$.

*Both authors were supported by the Austrian Science Foundation FWF, grant P25337-N23 and grant P25518-N23, respectively.

\mathcal{C} PERMUTATION PATTERN MATCHING (\mathcal{C} PPM)

<i>Instance:</i> A permutation T (the text) and a \mathcal{C} pattern P

<i>Question:</i> Does the \mathcal{C} pattern P occur in T ?
--

In this paper we study the classical, vincular, bivincular, mesh, boxed mesh and consecutive pattern matching problem. Often we abbreviate CLASSICAL PERMUTATION PATTERN MATCHING with PPM and the other problems with \mathcal{C} PPM, where \mathcal{C} is the corresponding pattern type.

While the combinatorial structure of permutation patterns is being extensively studied, the computational perspective has so far received less attention. This paper draws a map of the computational landscape of permutation patterns and thus aims at paving the way for a detailed computational analysis.

The contents of this paper are the following:

- We survey different types of permutation patterns (Section 3). This paper focuses on classical, vincular, bivincular, mesh, boxed mesh and consecutive patterns. The hierarchy of these patterns with the most general one at the top is displayed in Figure 1.
- We study the computational complexity of each corresponding permutation pattern matching problem. It is known that CLASSICAL PERMUTATION PATTERN MATCHING is NP-complete [7] and consequently VINCULAR, BIVINCULAR and MESH PPM as well. We strengthen this result and also show that pattern matching with boxed mesh and consecutive patterns can be performed in polynomial time (Section 4).
- We offer a more fine-grained complexity analysis by employing the framework of parameterized complexity. For most NP-complete problems we provide a more detailed complexity classification by showing W[1]-completeness with respect to the parameter length of P (Section 5). Both the classical as well as the parameterized complexity results are summarized in Table 1 (page 6) and Table 2 (page 7).
- In Section 6 we highlight several possible research directions and mention some aspects of permutation pattern matching that have not been studied in this paper.

2 Preliminaries

Permutations. Let $[n] = \{1, \dots, n\}$ and $[m, n] = \{m, m + 1, \dots, n\}$. A permutation π on the set $[n]$ can be seen as the sequence $\pi(1), \pi(2), \dots, \pi(n)$. Viewing permutations as sequences allows us to speak of *subsequences* of a permutation. We speak of a *contiguous subsequence* of π if the sequence consists of contiguous elements in π .

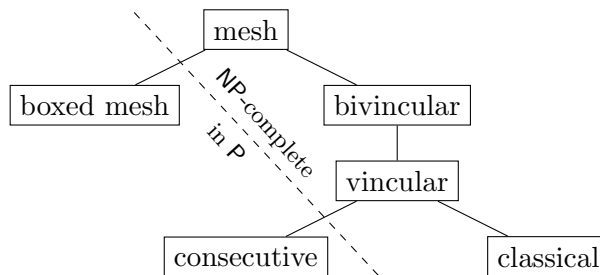


Figure 1: Hierarchy of pattern types.

Graphically, a permutation π on $[n]$ can be represented with the help of a $[0, n + 1] \times [0, n + 1]$ -grid in which elements marked by black circles are placed at the position (i, j) whenever $\pi(i) = j$. This representation thus corresponds to the function graph of π when viewing permutations as bijective maps. For examples, see Tables 1 and 2 where all permutations are represented with the help of grids.

We denote by π^{-1} the inverse of the permutation π , by $\pi^r := \pi(n)\pi(n - 1) \dots \pi(1)$ its reverse and by $\pi^c := (n - \pi(1) + 1)(n - \pi(2) + 1) \dots (n - \pi(n) + 1)$ its complement. From the grid corresponding to π one can obtain π^{-1} 's grid by reflecting across the line $y = x$, π^r 's grid by reflecting across $x = n/2$ and π^c 's grid by reflecting across $y = n/2$.

Classical complexity theory. This paper classifies several permutation pattern matching problems by proving membership in complexity classes. We give a brief reminder of the two fundamental classes P and NP . The class P contains all problems that can be solved in polynomial time on a deterministic Turing machine. It is important to note that polynomial time means polynomial in the size of the input. The class NP contains all problems that can be solved in polynomial time on a *non-deterministic* Turing machine. A problem is NP -hard if every problem in NP can be reduced to it by a polynomial time reduction. Furthermore, a problem is NP -complete if it is contained in NP and NP -hard. For a detailed introduction to complexity theory we refer to the monographs by Papadimitriou [27], Goldreich [16], and Arora and Barak [3].

Parameterized complexity theory. In contrast to classical complexity theory, a parameterized complexity analysis studies the runtime of an algorithm with respect to an additional parameter and not just the input size $|I|$. Therefore every parameterized problem is considered as a subset of $\Sigma^* \times \mathbb{N}$, where Σ is the input alphabet. An instance of a parameterized problem consequently consists of an input string together with a positive integer p , the parameter.

Definition 2.1. A parameterized problem is fixed-parameter tractable (or in FPT) if there is a computable function f and an integer c such that there is an algorithm solving the problem in $\mathcal{O}(|I|^c \cdot f(p))$ time.

The algorithm itself is also called fixed-parameter tractable (fpt). In this paper we want to focus on the exponential runtime of algorithms, i.e., the function f , and therefore use the \mathcal{O}^* notation which neglects polynomial factors.

A central concept in parameterized complexity theory are *fixed-parameter tractable reductions*.

Definition 2.2. Let $L_1, L_2 \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. An fpt-reduction from L_1 to L_2 is a mapping $R : \Sigma^* \times \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}$ such that

- $(I, k) \in L_1$ iff $R(I, k) \in L_2$.
- R is computable by an fpt-algorithm.
- There is a computable function g such that for $R(I, k) = (I', k')$, $k' \leq g(k)$ holds.

Other important complexity classes in the framework of parameterized complexity are $W[1] \subseteq W[2] \subseteq \dots$, the W-hierarchy. For our purpose, only the class $W[1]$ is relevant.

Definition 2.3. The class $W[1]$ is defined as the class of all problems that are fpt-reducible to the following problem.

SHORT TURING MACHINE ACCEPTANCE

Instance: A nondeterministic Turing machine with its transition table, an input word x and a positive integer k .

Parameter: k

Question: Does the Turing machine accept the input x in at most k steps?

It is conjectured (and widely believed) that $W[1] \neq \text{FPT}$. Therefore showing $W[1]$ -hardness can be considered as evidence that the problem is not fixed-parameter tractable.

Definition 2.4. A parameterized problem is in XP if it can be solved in time $\mathcal{O}(|I|^{f(k)})$ where f is a computable function.

All the aforementioned classes are closed under fpt-reductions. The following relations between these complexity classes are known:

$$\text{FPT} \subseteq W[1] \subseteq W[2] \subseteq \dots \subseteq \text{XP} \quad \text{as well as}$$

$$\text{FPT} \subset \text{XP}.$$

Further details can be found, for example, in the monographs by Downey and Fellows [12], Niedermeier [26] and Flum and Grohe [15].

3 Types of patterns

In this section we give an overview of several different types of permutation patterns that have been introduced in the last years and that will be of interest in this paper. These are classical, vincular, bivincular, mesh, boxed mesh and consecutive patterns. A schematic representation of their hierarchy can be found in Figure 1. For details, we refer to the Chapters 1 and 5-7 in Kitaev's monograph *Patterns in Permutations and Words* [21]. Before we introduce different types of patterns, we precisely define matchings in the context of permutation patterns.

Definition 3.1. *Let $\mathcal{C} \in \{\text{classical, vincular, bivincular, mesh, boxed mesh, consecutive}\}$. A matching of a \mathcal{C} pattern P of length k into a permutation T of length n is an increasing mapping $\mu : [k] \rightarrow [n]$ such that the sequence $\mu(P(1)), \mu(P(2)), \dots, \mu(P(k))$ is an occurrence of the \mathcal{C} pattern P in T .*

Matchings are denoted by μ throughout the paper.

3.1 Classical patterns

Classical permutation patterns, or simply permutation patterns, have implicitly been studied in different contexts for more than a hundred years. The first mentioning of a (classical) permutation pattern is attributed to Knuth and an exercise of his *Fundamental algorithms* [22] in 1968. It was however only in 1985 that Simion and Schmidt performed the first systematic study of patterns in permutations in [29]. For an introduction to classical permutation patterns, see also Bóna's *Combinatorics of permutations* [6].

Definition 3.2. *Let P be a permutation of length k and $T = T(1) \dots T(n)$ a permutation of length n . An occurrence of the classical permutation pattern P is a subsequence $T(i_1)T(i_2) \dots T(i_k)$ of T that is order-isomorphic to P , i.e., a subsequence in which the letters appear in the same relative order as in P . If such a subsequence exists, one says that T contains P or that there is a matching of P into T . If there is no such map one says that T avoids the (classical) pattern P .*

Example 3.3. The classical pattern $P = 132$ is contained several times in the text $T = 164253$ as for instance shown by the subsequence 142. A matching μ is given by $\mu(1) = 1$, $\mu(3) = 4$ and $\mu(2) = 2$. The pattern $P = 1234$ is however not contained in T since no increasing subsequence of length four can be found in T .

Representing permutations with the help of grids allows for a simple interpretation of classical pattern containment respectively avoidance in permutations. Indeed, the pattern P is contained in the permutation T iff the grid corresponding to P can be obtained from the one corresponding to T


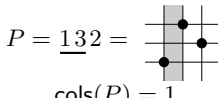
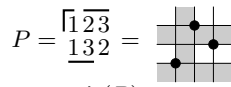


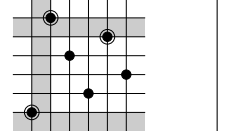
	Classical	Vincular	Bivincular
Pattern	$P = 132 = $ 	$P = \underline{1}32 = $  $\text{cols}(P) = 1$	$P = \overline{123} \underline{132} = $  $\text{cols}(P) = 1$ $\text{rows}(P) = 2$
Text			
Classical complexity	NP-complete [7]	NP-complete Corollary 4.4	NP-complete Corollary 4.4
Parameterized complexity	FPT [17]	W[1]-complete Theorem 5.5	W[1]-complete Theorem 5.6

Table 1: Examples of classical, vincular and bivincular permutation patterns.

by deleting some columns and rows. For the example given above, see the left column in Table 1, in which the elements involved in the matching have been marked by circled elements.

It is easy to see that P can be matched into T iff P^c can be matched into T^c , iff P^r can be matched into T^r and iff P^{-1} can be matched into T^{-1} .

3.2 Vincular patterns

Let $T(i_1)T(i_2)\dots T(i_k)$ be an occurrence of the classical pattern P in the text T . Then there are no requirements on the elements in T lying in between $T(i_j)$ and $T(i_{j+1})$. It is however natural to ask for occurrences of patterns in which certain elements are forced to be adjacent in the text, i.e., $T(i_{j+1}) = T(i_j + 1)$. Vincular patterns are a generalization of classical patterns capturing these requirements on adjacency in the text. They were introduced under the name of *generalized patterns* in 2000 by Babson and Steingrímsson in [5], where it was shown that essentially all Mahonian permutation statistics in the literature can be written as linear combinations of vincular patterns. For a survey of this topic, see [30].

Here we use the name of *vincular patterns* as it was introduced by Kitaev in [21]. We also use the notation introduced there, since it is consistent with the notation for classical patterns.

Definition 3.4. A vincular pattern P is a permutation in which certain consecutive entries may be underlined. An occurrence of P in a permutation T is then an occurrence of the corresponding classical pattern for which underlined elements are matched to adjacent elements. To be more formal: An occurrence of P in T corresponds to a subsequence $T(i_1)T(i_2)\dots T(i_k)$ of

	Mesh	Boxed mesh	Consecutive
Pattern	$P = (\pi, R) = $ $\text{cells}(P) = 5$	$P = \boxed{132} = $	$P = \underline{132} = $
Text			
Classical complexity	NP-complete Corollary 4.4	in P; Theorem 4.5	in P; Theorem 4.6
Parameterized Complexity	W[1]-complete Theorem 5.7	trivially FPT	trivially FPT

Table 2: Examples of mesh, boxed mesh and consecutive permutation patterns.

T that is order-isomorphic to P and for which $T(i_{j+1}) = T(i_j + 1)$ whenever P contains $\underline{P(j)P(j+1)}$. Furthermore, if P starts with $\underline{P(1)}$ an occurrence of P in T must start with the first entry in T , i.e., $T(i_1) = T(1)$. Similarly, if P ends with $\underline{P(k)}$ it must hold that $T(i_k) = T(n)$.

When representing permutations by grids, adjacency of positions clearly corresponds to adjacency of columns. In order to represent the underlined elements in vincular patterns in the corresponding grids, one shades the columns which may not contain any elements in a matching. For an example, see the middle column of Table 1. Matching the pattern $\underline{132}$ into the permutation T , means that no elements may lie in the columns between $\mu(1)$ and $\mu(3)$ in T .

In order to specify how many adjacency restrictions are made in the vincular pattern P , we define $\text{cols}(P)$ to be the number of shaded columns in the grid corresponding to P .

Note that the operations *complement* and *reverse* may be performed on vincular patterns, leading to some (other) vincular pattern. Similarly as for classical patterns it then holds that P can be matched into T iff P^c can be matched into T^c and iff P^r can be matched into T^r . The inverse of a vincular pattern is however not clearly defined. This leads to a larger class of patterns which is introduced below.

3.3 Bivincular patterns

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 matching may be forced to be adjacent. When

Bousquet-Mélou, Claesson, Dukes and Kitaev introduced bivincular patterns in 2010 [8], the main motivation was to find a minimal superset of vincular patterns that is closed under the inverse operation. As mentioned in Section 3.2, the inverse of a vincular pattern is not well-defined - it is a bivincular, but not a vincular pattern.

Definition 3.5. A bivincular pattern P is a permutation written in two-line notation, where some elements in the top row may be overlined and the bottom row is a vincular pattern as defined in Definition 3.4. An occurrence $T(i_1)T(i_2)\dots T(i_k)$ of P in a permutation T is an occurrence of the corresponding vincular pattern where additionally the following holds: $T(i_{j+1}) = T(i_j) + 1$ whenever the top row of P contains $\overline{j(j+1)}$. Furthermore, if the top row starts with $\overline{1}$, an occurrence of P in T must start with the smallest entry in T , i.e., $T(i_1) = 1$. Similarly, if the top row ends with \overline{k} , it must hold that $T(i_k) = n$.

This definition gets a lot less cumbersome when representing permutations with the help of grids: As remarked earlier, underlined elements in the bottom row are translated into forbidden columns in which no elements may occur in a matching. Similarly, overlined elements in the top row are translated into forbidden rows. For an example, see the right column in Table 1.

Again, in order to specify how many adjacency restrictions are made in the bivincular pattern P , we define - in addition to $\text{cols}(P)$ - $\text{rows}(P)$ to be the number of shaded rows in the grid corresponding to P .

3.4 Mesh patterns

A further generalization of bivincular patterns was given by Brändén and Claesson who introduced mesh patterns in [9] in 2011. Mesh patterns allow further restrictions on the relative positions of the entries in an occurrence of a pattern. Several permutation statistics can be formulated as the number of occurrences of certain mesh patterns [9].

Definition 3.6. A mesh pattern is a pair $P = (\pi, R)$ where π is a permutation of length k and $R \subset [0, k] \times [0, k]$ is a relation. An occurrence of P in a permutation T is an occurrence of the classical pattern π fulfilling additional restrictions defined by R . That is to say there is a subsequence $T(i_1)T(i_2)\dots T(i_k)$ of T that is order-isomorphic to π and for which holds:

$$(x, y) \in R \implies \nexists i \in [n] : i_x < i < i_{x+1} \wedge T(i_{\pi^{-1}(y)}) < T(i) < T(i_{\pi^{-1}(y+1)}).$$

This definition is again a lot easier to capture when representing permutations as grids. Indeed, the relation R can be translated very easily

into the graphical representation of $P = (\pi, R)$, by shading the unit square with bottom left corner (x, y) for every $(x, y) \in R$. An occurrence of P in a permutation T is then a classical occurrence of π in T such that no elements of T lie in the shaded regions of the grid.

Again, in order to specify how many adjacency restrictions are made in the mesh pattern P , we define $\text{cells}(P)$ to be the number of shaded cells in the corresponding grid. Thus $\text{cells}(\pi, R) := |R|$. For an example with $P = (\pi, R)$ with $\pi = 132$ and $R = \{(1, 0), (1, 2), (2, 3), (3, 0), (3, 1)\}$ see the left column in Table 2.

3.5 Boxed mesh patterns

A special case of mesh patterns, so called boxed mesh patterns, was very recently introduced by Avgustinovich, Kitaev and Valyuzhenich in [4].

Definition 3.7. *A boxed mesh pattern, or simply boxed pattern, is a mesh pattern $P = (\pi, R)$ where π is a permutation of length k and $R = [1, k - 1] \times [1, k - 1]$. P is then denoted by $\boxed{\pi}$.*

In the grid representing a boxed pattern all but the boundary squares are shaded. For an example, see the middle column of Table 2.

It is straightforward to see that the set of boxed patterns is closed under taking complements, reverses and inverses and that these operations are compatible with pattern containment. Interestingly, it was shown [4] that the statement “ π can be matched into T iff $\boxed{\pi}$ can be matched into T ” is only true if π is one of the following permutations: 1, 12, 21, 132, 213, 231, 312.

3.6 Consecutive patterns

Consecutive patterns are a special case of vincular patterns, namely those where *all* entries are underlined. In an occurrence of a consecutive pattern it is thus necessary that all entries are adjacent. Finding an occurrence of a consecutive pattern therefore consists in finding a contiguous subsequence of T that is order-isomorphic to P . For an example, see the right column of Table 2.

Several well-known enumeration problems for permutations can be formulated in terms of forbidden consecutive patterns; Elizalde and Noy [13] provide examples. Chapter 5 in [21] is devoted to and gives an overview of different methods employed in the literature for the study of consecutive patterns.

4 The possibility of polynomial-time algorithms

4.1 NP-completeness

At the 1992 SIAM Discrete Mathematics meeting Herbert Wilf asked whether it is possible to solve the permutation pattern matching problem in polynomial time. The answer is no unless $P=NP$, as shown by the NP-completeness result of Bose, Buss and Lubiw [7]. This result immediately yields NP-hardness for all generalizations of classical permutation pattern matching. In this section we are going to show that NP-hardness holds for these problems even in a more restricted case: with all runs having length at most two.

Definition 4.1. *A run in a permutation is a maximal monotone contiguous subsequence. Let $\text{lrun}(\pi)$ denote the length of the longest run in the permutation π .*

Note that for any permutation π with length at least two it holds that $\text{lrun}(\pi) \geq 2$.

Theorem 4.2. *Every MESH PERMUTATION PATTERN MATCHING instance $(P, T) = ((\pi, R), T)$ can be transformed into an instance (P', T') with $P' = (\pi', R)$ and the following properties: (P', T') is a yes-instance iff (P, T) is yes-instance, $|\pi'| = 2|\pi|$, $|T'| = 2|T|$ and $\text{lrun}(\pi') = \text{lrun}(T') = 2$. This transformation can be done in polynomial time.*

Proof. Let $\pi = p_1 \dots p_k$ and $T = t_1 \dots t_n$. We define

$$\begin{aligned}\pi' &= (k+1) p_1 (k+2) p_2 (k+3) \dots (2k) p_k \\ T' &= (n+1) t_1 (n+2) t_2 (n+3) \dots (2n) t_n.\end{aligned}$$

Clearly, $|\pi'| = 2|\pi|$, $|T'| = 2|T|$ and $\text{lrun}(\pi') = \text{lrun}(T') = 2$. We are now going to show that there is a matching from P into T iff there is a matching from P' into T' . Assume that μ is a matching from P into T , i.e., a map from $[k]$ to $[n]$. We extend this map to a map μ' from $[2k+1]$ to $[2n+1]$ in the following way:

$$\mu'(i) = \begin{cases} \mu(i), & \text{if } i \in [k], \\ T(j), \text{ where } \mu(i-k) = T(j+1) & \text{if } i > k. \end{cases}$$

In other words, μ' maps $(i+k)$ to the element in T left of $\mu(i)$. For example if $\mu(p_3) = t_5$ then $p_3 \in \pi'$ is matched to $t_5 \in T'$ and $(k+3) \in \pi'$ is matched to $n+5 \in T'$ (which is the element in T lying directly to the left of t_5). Observe that the function μ' is a matching from P' into T' .

Now let us assume that μ' is a matching from P' into T' . If we restrict the domain of μ' to $[k]$ then we obtain a matching from P into T . \square

Theorem 4.3. PPM is NP-complete even on permutations P and T with $\text{lrun}(P') = \text{lrun}(T') = 2$.

Proof. We apply the transformation in Theorem 4.2 to show NP-hardness. NP-membership holds for this restricted class of input instances as well. \square

Corollary 4.4. VINCULAR, BIVINCULAR and MESH PPM are NP-complete even if $\text{lrun}(P') = \text{lrun}(T') = 2$.

Proof. NP-hardness follows from the Theorem 4.2 as well as from Theorem 4.3. NP-membership holds since checking whether the additional restrictions imposed by the vincular, bivincular or mesh pattern are fulfilled can clearly be done in polynomial time. \square

4.2 Polynomial time algorithms

We have seen that polynomial time algorithms are unlikely to exist for PPM and its generalizations. However, this is not the case for the special cases of boxed mesh and consecutive pattern matching.

Theorem 4.5. BOXED MESH PERMUTATION PATTERN MATCHING can be solved in $\mathcal{O}(n^3)$ time.

Proof. Let P be a boxed pattern of length k and T a permutation of length n . For every pair (i, j) where $i \in [n]$ and $i + k \leq j \leq n$ check whether there is a matching μ of the boxed pattern P into T where the smallest element in P is matched to i and the largest one to j , i.e., $\mu(1) = i$ and $\mu(k) = j$.

Checking whether such a matching exists can be done in the following way: From the permutation T , construct the permutation \tilde{T} by deleting all elements that are smaller than i and larger than j . Clearly, the matching that we are looking for must be contained in \tilde{T} , it could otherwise not be an occurrence of a boxed pattern. Moreover, it has to consist of k consecutive elements in \tilde{T} . Since the positions of the smallest and the largest element are fixed, the positions for all other elements of P are equally determined. Thus there is only one subsequence of T that could possibly be a matching of P into T with $\mu(1) = i$ and $\mu(k) = j$. Deleting the elements that are too small or too large and checking whether this subsequence actually corresponds to an occurrence of P in T , i.e., whether it is order-isomorphic to P , can be checked in at most n steps. Note that this subsequence might consist of less than k elements in which case it clearly does not correspond to an occurrence.

In total, there are $(n - k + 1) \cdot (n - k + 2)/2 = \mathcal{O}(n^2)$ pairs (i, j) that have to be checked which leads to the runtime bound $\mathcal{O}(n^3)$. \square

Theorem 4.6. CONSECUTIVE PERMUTATION PATTERN MATCHING can be solved in $\mathcal{O}((n - k) \cdot k)$ time.

Proof. Let P be a consecutive pattern of length k and T a permutation of length n . For every $i \in [n - k + 1]$ check whether there is a matching of P into T where the first element of P is mapped to i . Since we are looking for an occurrence of a consecutive pattern, the only possible subsequence of T then consists of the element i and the following $(k - 1)$ elements of T . Whether this sequence is order-isomorphic to P can be checked in k steps which leads to the runtime bound $\mathcal{O}((n - k) \cdot k)$. \square

As was recently shown by Kubica et.al. in [23], this simple result can be improved by an algorithm with runtime $\mathcal{O}(n + k)$.

Even though PPM is NP-complete in the general case, there are special cases of input instances for which the problem can be solved efficiently, i.e., in polynomial time. In the following we list the cases for which it is known that PPM can be solved in polynomial time.

- In case the pattern is a *separable* permutation, i.e., a permutation avoiding both 3142 and 2413, PPM can be solved in $\mathcal{O}(k \cdot n^6)$ [7]. This runtime has been improved to $\mathcal{O}(k \cdot n^4)$ in [20].
- PPM can be solved in $\mathcal{O}(n \log n)$ -time for all patterns of length four [2]. For the pattern 4312, this is even possible in linear time [24].
- In case P is the identity $12 \dots k$, PPM consists of looking for an increasing subsequence of length k in the text – this is a special case of the LONGEST INCREASING SUBSEQUENCE problem. This problem can be solved in $\mathcal{O}(n \log n)$ -time for sequences in general [28] and in $\mathcal{O}(n \log \log n)$ -time for permutations [11, 25].
- A $\mathcal{O}(k^2 n^6)$ -time algorithm is presented in [18] for the case that both the text and the pattern are 321-avoiding.

5 The impact of the pattern length

PPM can be solved in $\mathcal{O}(n^k)$ time by exhaustive search, where k is the length of P . This trivial upper bound has been improved first by Albert et al. to $\mathcal{O}(n^{1+2k/3} \cdot \log n)$ [2] and then to $\mathcal{O}(n^{0.47k+o(k)})$ by Ahal and Rabinovich [1]. In a recent breakthrough result, Guillemot and Marx have shown that PPM can be solved by an FPT algorithm [17]. Its runtime is $2^{\mathcal{O}(k^2 \cdot \log k)} \cdot n$. In this section we are going to show that such a result is likely not to be achievable for VINCULAR, BIVINCULAR and MESH PPM. This is done by showing W[1]-hardness with respect to the parameter k . First, we show that MESH PPM and therefore all other problems studied in this paper are contained in W[1].

All results in this section are summarized in Figure 3 on page 20.

Theorem 5.1. MESH PERMUTATION PATTERN MATCHING is contained in W[1].

Proof. For showing membership we encode MESH PPM as a model checking problem of an existential first order formula. W[1]-membership is then a consequence of the fact that the following problem is W[1]-complete [14].

EXISTENTIAL FIRST-ORDER MODEL CHECKING

Instance: A structure \mathcal{A} and an existential first-order formula φ
Parameter: $|\varphi|$
Question: Is \mathcal{A} a model for φ ?

Let $((P, R), T)$ be a MESH PPM instance. We compute a structure $\mathcal{A} = (A, <, \prec_T, E)$, where the domain set $A = \{1, \dots, n\}$ represents indices in the text. The binary relation \prec_T is defined by $x \prec_T y$ holds iff $T(x) < T(y)$. E is a quaternary relation where $E(w, x, y, z)$ is true iff there are no elements in T that are left of w , right of x , larger than y and smaller than z . Intuitively, w, x, y and z describe a *forbidden* rectangle in the permutation grid of T which may not contain any elements of T . $T_{<}$, E and $<$ can be computed in polynomial time. The formula φ we want to check is

$$\varphi = \exists x_1 \dots \exists x_k \quad x_1 < x_2 \wedge x_2 < x_3 \wedge \dots \wedge x_{k-1} < x_k \wedge$$

$$\underbrace{\bigwedge_{\substack{P(i) < P(j) \\ \text{for } i, j \in [k]}} x_i \prec_T x_j}_{\varphi_1} \wedge \underbrace{\bigwedge_{\substack{P(i) > P(j) \\ \text{for } i, j \in [k]}} \neg(x_i \prec_T x_j)}_{\varphi_2} \wedge \underbrace{\bigwedge_{\substack{i, j \in [k] \text{ and} \\ R(i, j) \text{ is true.}}} E(x_i, x_{i+1}, x_j, x_{j+1})}_{\varphi_3}.$$

Observe that the length of φ is in $\mathcal{O}(k^2)$. The two sub-formulas φ_1 and φ_2 are exactly then true when a subsequence $T(x_1)T(x_2) \dots T(x_k)$ of T can be found such that $T(x_i) < T(x_j)$ iff $P(i) < P(j)$. Thus $\varphi_1 \wedge \varphi_2$ is true iff there is a matching of the classical pattern P into T . The sub-formula φ_3 encodes the relation R and is true iff no elements lie in the forbidden regions of T , as can be seen by recalling Definition 3.6. Thus φ is true iff $((P, R), T)$ is a yes-instance of MESH PPM. \square

We now want to prove W[1]-hardness for vincular, bivincular and mesh permutation matching. For this purpose, we introduce here SEGREGATED PERMUTATION PATTERN MATCHING, a generalization of PPM. All subsequent hardness theorems use reductions from this problem.

SEGREGATED PERMUTATION PATTERN MATCHING (SPPM)

Instance: A permutation T (the text) of length n , a permutation P (the pattern) of length $k \leq n$ and two positive integers $p \in [k]$, $t \in [n]$.
Parameter: k
Question: Is there a matching μ of P into T such that $\mu(i) \leq t$ iff $i \leq p$?

Example 5.2. Consider the pattern $P = 132$ and the text $T = 53142$. As shown by the matching $\mu(2) = 3$, $\mu(1) = 1$ and $\mu(3) = 4$, the instance $(P, T, 2, 3)$ is a yes-instance of the SPPM problem. However, $(P, T, 2, 4)$ is a NO-instance, since no matching of P into T can be found where $\mu(3) > 4$.

Theorem 5.3. SEGREGATED PERMUTATION PATTERN MATCHING is W[1]-hard with respect to the parameter k .

Proof. We show W[1]-hardness by giving an fpt-reduction from the W[1]-complete CLIQUE problem [12] to SPPM:

CLIQUE	
<i>Instance:</i>	A graph $G = (V, E)$ and a positive integer k .
<i>Parameter:</i>	k
<i>Question:</i>	Is there a subset of vertices $S \subseteq V$ of size k such that S forms a clique, i.e., the induced subgraph $G[S]$ is complete?

The reduction has three parts. First, we will show that we are able to reduce a CLIQUE instance to a pair (P', T') , where P' and T' are two permutations on multisets, i.e., permutations in which elements may occur more than once. Applying Definition 3.2 to permutations on multisets means that in a matching repeated elements in the pattern have to be mapped to repeated elements in the text. In addition to repeated elements, P' and T' contain so-called *guard elements*. Their function is explained below. Second, we will show how to get rid of repetitions. The method used in this step has already been used in the NP-completeness proof of PPM provided by Bose, Buss and Lubiw in [7]. Third, we implement the guards by using the segregation property and have thus reduced CLIQUE to SPPM.

Let (G, k) be a CLIQUE instance, where $V = \{v_1, v_2, \dots, v_l\}$ is the set of vertices and $E = \{e_1, e_2, \dots, e_m\}$ the set of edges. Both the pattern and the text consist of a single substring coding vertices (\dot{P} resp. \dot{T}) and substrings coding edges (\bar{P}_i resp. \bar{T}_i for the i -th substring). These substrings are listed one after the other, with *guard elements* placed in between them. These guard elements have the function of separating substrings in a matching: guard elements will have to be mapped to guard elements and substrings embraced by two consecutive guard-elements will also have to be mapped to substrings embraced by two consecutive guard-elements. For the moment, we will simply write brackets to indicate where guard elements are placed. The meaning of these brackets is then the following: a block of elements enclosed by a \langle to the left and a \rangle to the right has to be matched into another block of elements between two such brackets. How the guard-elements are implemented as elements of a permutation is explained at the end of the proof after Claim 2.

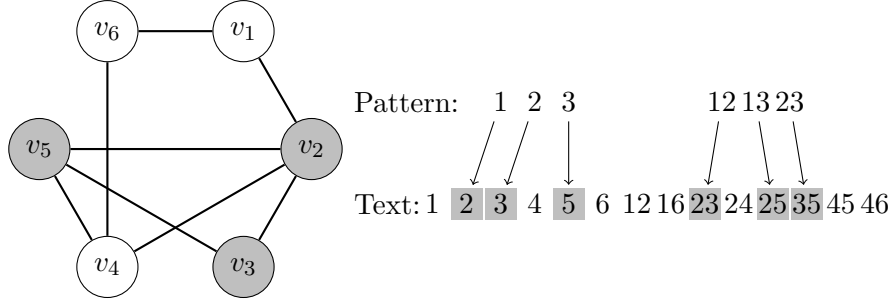


Figure 2: An example for the reduction of an INDEPENDENT SET instance to a PPM instance.

We define the pattern to be

$$\begin{aligned} P' &:= \langle \dot{P} \rangle \langle \bar{P}_1 \rangle \langle \bar{P}_2 \rangle \langle \dots \rangle \langle \bar{P}_{k(k-1)/2} \rangle \\ &= \langle 123 \dots k \rangle \langle 12 \rangle \langle 13 \rangle \langle \dots \rangle \langle 1k \rangle \langle 23 \rangle \langle \dots \rangle \langle 2k \rangle \langle \dots \rangle \langle (k-1)k \rangle. \end{aligned}$$

\dot{P} corresponds to a list of (indices of) k vertices. The \bar{P}_i 's represent all possible edges between the k vertices (in lexicographic order).

For the text

$$T' := \langle \dot{T} \rangle \langle \bar{T}_1 \rangle \langle \bar{T}_2 \rangle \langle \dots \rangle \langle \bar{T}_m \rangle$$

we proceed similarly. \dot{T} is a list of the (indices of the) l vertices of G . The \bar{T}_i 's represent all edges in G (again in lexicographic order). Let us give an example:

Example 5.4. Let $l = 6$ and $k = 3$. Then the pattern permutation is given by

$$P' = \langle 123 \rangle \langle 12 \rangle \langle 13 \rangle \langle 23 \rangle.$$

Consider for instance the graph G with six vertices v_1, \dots, v_6 and edge-set

$$\{\{1, 2\}, \{1, 6\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 5\}, \{4, 5\}, \{4, 6\}\}.$$

represented in Figure 2 (we write $\{i, j\}$ instead of $\{v_i, v_j\}$).

Then the text permutation is given by:

$$T' = \langle 123456 \rangle \langle 12 \rangle \langle 16 \rangle \langle 23 \rangle \langle 24 \rangle \langle 25 \rangle \langle 35 \rangle \langle 45 \rangle \langle 46 \rangle.$$

Claim 1. A clique of size k can be found in G iff there is a simultaneous matching of \dot{P} into \dot{T} and of every \bar{P}_i into some \bar{T}_j .

Example 5.4 (continuation). In our example $\{v_2, v_3, v_5\}$ is a clique of size three. Indeed, the pattern P' can be matched into T' as can be seen by matching the elements 1, 2 and 3 onto 2, 3 and 5 respectively. See again Figure 2 where the involved vertices respectively elements of the text permutation have been marked in gray.

Proof of Claim 1. A matching of \dot{P} into \dot{T} corresponds to a selection of k vertices amongst the l vertices of G . If it is possible to additionally match every one of the \bar{P} 's into a \bar{T} this means that all possible edges between the selected vertices appear in G . This is because T' only contains pairs of indices that correspond to edges appearing in the graph. The selected k vertices thus form a clique in G . Conversely, if for every possible matching of \dot{P} into \dot{T} defined by a monotone map $\mu : [k] \rightarrow [l]$ some $\bar{P}_i = xy$ cannot be matched into T' , this means that $\{\mu(x), \mu(y)\}$ does not appear as an edge in G . Thus, for every selection of k vertices there will always be at least one pair of vertices that are not connected by an edge and therefore there is no clique of size k in G . \square

In order to get rid of repeated elements, we identify every variable with a real interval: 1 corresponds to the interval $[1, 1.9]$, 2 to $[2, 2.9]$ and so on until finally k corresponds to $[k, k+0.9]$ (resp. l to $[l, l+0.9]$). In \dot{P} and \dot{T} we shall therefore replace every element j by the pair of elements $(j + 0.9, j)$ (in this order). The occurrences of j in the \bar{P}_i 's (resp. \bar{T}_i 's) shall then successively be replaced by real numbers in the interval $[j, j + 0.9]$. For every j , these values are chosen one after the other (from left to right), always picking a real number that is larger than all the previously chosen ones in the interval $[j, j + 0.9]$.

Observe the following: The obtained sequence is not a permutation in the classical sense since it consists of real numbers. However, by replacing the smallest number by 1, the second smallest by 2 and so on, we do obtain an ordinary permutation. This defines P and T (except for the guard elements).

Example 5.4 (continuation). Getting rid of repetitions in the pattern of the above example could for instance be done in the following way:

$$P = \langle 1.9 \ 1 \ 2.9 \ 2 \ 3.9 \ 3 \rangle \langle 1.1 \ 2.1 \rangle \langle 1.2 \ 3.1 \rangle \langle 2.2 \ 3.2 \rangle$$

This permutation of real numbers is order-isomorphic to the following ordinary permutation:

$$P = \langle 4 \ 1 \ 8 \ 5 \ 12 \ 9 \rangle \langle 2 \ 6 \rangle \langle 3 \ 10 \rangle \langle 7 \ 11 \rangle.$$

Claim 2. P can be matched into T iff P' can be matched into T' .

Proof of Claim 2. Suppose that P' can be matched into T' . When matching P into T , we have to make sure that elements in P that were copies of some

repeated element in P' may still be mapped to elements in T that were copies themselves in T' . Indeed this is possible since we have chosen the real numbers replacing repeated elements in increasing order. If i in P' was matched to j in T' , then the pair $(i + 0.9, i)$ in P may be matched to the pair $(j + 0.9, j)$ in T and the increasing sequence of elements in the interval $[i, i + 0.9]$ may be matched into the increasing sequence of elements in the interval $[j, j + 0.9]$.

Now suppose that P can be matched into T . In order to prove that this implies that P' can be matched into T' , we merely need to show that elements in P that were copies of some repeated element in P' have to be mapped to elements in T that were copies themselves in T' . Then returning to repeated elements clearly preserves the matching. Firstly, it is clear that a pair of consecutive elements $i + 0.9$ and i in \dot{P} has to be matched to some pair of consecutive elements $j + 0.9$ and j in \dot{T} , since j is the only element smaller than $j + 0.9$ and appearing to its right. Thus intervals are matched to intervals. Secondly, an element x in P for which it holds that $i < x < i + 0.9$ must be matched to an element y in T for which it holds that $j < y < j + 0.9$. Thus copies of an element are still matched to copies of some other element.

Finally, replacing real numbers by integers does not change the permutations in any relevant way. \square

It remains to implement the guards in order to ensure that substrings are matched to corresponding substrings. Let P_{\max} and T_{\max} denote the largest integer that is contained in P respectively T at this point. We now replace all guards with integers larger than P_{\max} respectively T_{\max} and will choose the *segregating elements* p and t such that guards and “original” pattern/text elements are separated. We insert the guard elements in the designated positions (previously marked by \langle and \rangle) in the following order: $P_{\max} + 2$ (instead of the first \langle), $P_{\max} + 1$ (instead of the first \rangle), $P_{\max} + 4$ (instead of the second \langle), $P_{\max} + 3$ (instead of the second \rangle), \dots , $P_{\max} + 2i$ (instead of the i -th \langle), $P_{\max} + 2i - 1$ (instead of the i -th \rangle), \dots , and so on until we reach the last guard-position. The guard elements are inserted in this specific order to ensure that two neighboring guard elements \langle and \rangle in P have to be mapped to two neighboring guard elements \langle and \rangle in T . We proceed analogously in T . To ensure that guards in P are matched to guards in T and pattern elements of P are matched to text elements in T , we set p to P_{\max} and t to T_{\max} .

This finally yields that (G, k) is a yes-instance of CLIQUE iff (P, T) is a yes-instance of SPPM. It can easily be verified that this reduction can be done in fpt-time. \square

As can easily be seen, the reduction performed in the proof of Theorem 5.3 can be done in polynomial time. Thus this proof immediately yields

NP-hardness for SPPM.

Now, that we have obtained this result, we are able to show $W[1]$ -hardness for PPM with vincular, bivincular and mesh patterns. As before, the parameter is the length of the pattern.

Theorem 5.5. *VINCULAR PERMUTATION PATTERN MATCHING is $W[1]$ -complete with respect to k . This holds even when restricting the problem to instances (P, T) with $\text{cols}(P) = 1$.*

Proof. We reduce from SEGREGATED PPM. Let (P, T, p, t) be an SPPM instance. The VINCULAR PPM instance (P', T') constructed from (P, T) will have an additional element in P' and an additional element in T' . The new element in P' , denoted by p' , is $p + 0.5$, i.e., p' is larger than p but smaller than $p + 1$. Analogously, $t' = t + 0.5$ is the new element in T' . We define $P' = \lfloor p' P$ and $T' = t' T$. In order to obtain a permutation P on $[k + 1]$ and T on $[n + 1]$, we simply need to relabel the respective elements order-isomorphically. In every matching of P' into T' the element p' has to be mapped to t' . Consequently, all elements larger than p' in P' have to be mapped to elements larger than t' in T' and all elements smaller than p' have to be mapped to elements smaller than t' . This implies that (P, T, p, t) is a SEGREGATED PPM yes-instance iff (P', T') is a VINCULAR PPM yes-instance. This reduction is done in linear time which proves $W[1]$ -hardness of VINCULAR PPM. Membership follows from Theorem 5.1. \square

Theorem 5.6. *BIVINCULAR PERMUTATION PATTERN MATCHING is $W[1]$ -complete with respect to k . This holds even when restricting the problem to instances (P, T) with $\text{rows}(P) = 1$.*

Proof. As in the previous proof we reduce from SEGREGATED PPM. Let (P, T, p, t) be an SPPM instance. Identically to the previous proof, we define $p' = p + 0.5$ and $t' = t + 0.5$. The BIVINCULAR PPM instance consists of a permutation P' with elements in $[k + 1] \cup \{p'\}$ and T' , a permutation on $[n + 1] \cup \{t'\}$. We define

$$P' = \begin{array}{ccccccc} 1 & 2 & 3 & \dots & p' & \dots & \overline{(k+1)} \\ p' & (k+1) & P(1) & & \dots & & P(k) \end{array}$$

and $T' = t'(n + 1)T$. In order to obtain permutations on $[k + 2]$ respectively $[n + 2]$ we again relabel the elements order-isomorphically.

In any matching of P' into T' the element $(k + 1)$ has to be mapped to $(n + 1)$ and therefore p' has to be mapped to t' . Thus all elements larger than p' in P' have to be mapped to elements larger than t' in T' and all elements smaller than p' have to be mapped to elements smaller than t' . This implies that (P, T, p, t) is a SEGREGATED PPM yes-instance iff (P', T') is a BIVINCULAR PPM yes-instance. Since this reduction can again be done in linear time, BIVINCULAR PPM is $W[1]$ -hard. Membership follows again from Theorem 5.1. \square

Theorem 5.7. MESH PERMUTATION PATTERN MATCHING is $W[1]$ -complete with respect to k . This holds even if $\text{cells}(P) = 1$.

Proof. Let (P, T, p, t) be a SEGREGATED PPM instance. As before, we define $p' = p + 0.5$ and $t' = t + 0.5$. The MESH PPM instance consists of a permutation P' with elements in $[k] \cup \{p'\}$ and T' , a permutation on $[n + 1] \cup \{p'\}$. Again, permutations on $[k + 1]$ respectively $[n + 2]$ can be obtained by relabelling the elements order-isomorphically. We define $P' = p' P$ and $T' = t' (n + 1) T$. Furthermore, let $R = \{(0, (k + 1))\}$. This means that for every matching μ of P' into T' the following must hold: to the left of $\mu(p')$ in T' , there are no elements larger than $\mu(k)$. However, it surely holds that $\mu(k) \leq (n + 1)$. Consequently, p' has to be mapped to t' . This implies that (P, T, p, t) is a SEGREGATED PPM yes-instance iff (P', T') is a MESH PPM yes-instance. Since this reduction can again be done in linear time, MESH PPM is $W[1]$ -hard. Membership follows from Theorem 5.1. \square

These hardness results show that we cannot hope for a fixed-parameter tractable algorithm for VINCULAR/BIVINCULAR/MESH PERMUTATION PATTERN MATCHING.

6 Further directions

In this paper, we have strengthened the previously known NP-hardness result for PPM and proved NP-completeness for its generalizations. We have also found polynomial time algorithms for boxed mesh and consecutive PPM. Furthermore, we have performed a parameterized complexity analysis, which shows that for vincular, bivincular and mesh PPM a fixed-parameter tractable algorithm is unlikely to exist. Refer to Figure 3 for an overview of these parameterized results. In this section we highlight several possible research directions and mention some aspects of permutation pattern matching that have not been studied in this paper.

Polynomial time algorithms. In Section 4.2 we have listed several special cases for which PPM is in P. This list, however, is certainly far from being complete. In particular, polynomial time fragments of vincular, bivincular and mesh permutation pattern matching are not known at all.

Other parameters than $k = |P|$. In Section 5 we studied the influence of the length of the pattern on the complexity of the different types of PERMUTATION PATTERN MATCHING problems. For the cases where $W[1]$ -hardness was shown as well as for PPM which is solvable by an FPT algorithm, it is of interest to find out whether other parameters of the input instances lead to fixed parameter tractability results. The authors [10] provided a

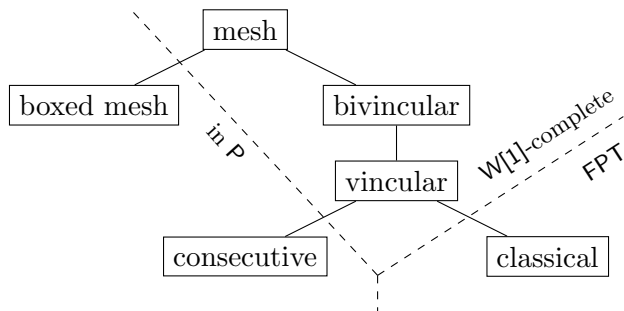


Figure 3: The influence of the pattern length on the computational hardness: parameterized complexity of permutation pattern matching.

first result in this vein by designing an algorithm that solves PPM with a worst-case runtime of $\mathcal{O}(1.79^{\text{run}(T)} \cdot n \cdot k)$, where $\text{run}(T)$ denotes the number of alternating runs of T . For future work any permutation statistic (see for instance the list in Appendix A.1 of [21]) could be taken into account for a parameterized complexity analysis of all versions of PPM. An analysis of PPM with respect to several different parameters would then allow to draw a more detailed picture of the computational landscape of permutation pattern matching.

Barred patterns. Another type of patterns was introduced by West [33]: *barred patterns*. A barred pattern $\bar{\pi}$ is a permutation where some letters are barred. One denotes by π the underlying permutation without any bars and by π' the permutation obtained by removing all barred letters. A permutation τ then avoids the barred pattern $\bar{\pi}$ if every occurrence of π' in τ is part of an occurrence of π in τ . For details and results on barred patterns, see Chapter 7 in [21]. Brändén and Claesson [9] showed that mesh patterns can easily be used to write any barred pattern with only one barred letter.

Marked mesh and decorated patterns. There exist further generalizations of mesh patterns: so-called *marked mesh* and *decorated patterns*. These two types of patterns were introduced by Úlfarsson in [31] and [32], respectively. They allow a finer control over whether certain regions in a permutation may contain elements. Mesh patterns permit to specify regions in a permutation that may not contain elements in a matching. Marked mesh patterns allow more, namely to specify how many elements may be contained in certain regions (exactly, at most or at least). Decorated patterns go even further: they allow to detail in which order these elements may lie by describing forbidden patterns. These forbidden patterns may again be decorated patterns. Since both marked mesh and decorated patterns are

generalizations of mesh patterns, the $W[1]$ -hardness result given in Theorem 5.7 can be extended to these two types of patterns.

Patterns in words. In this paper, we have considered patterns in *permutations*. However, the concept of pattern avoidance respectively containment can easily be extended to patterns in words over ordered alphabets (or permutations on multisets). In a matching of a word W into another word V , copies of the same letter have to be mapped to copies of some letter in the text. The topic of patterns in words has received quite some attention in the last years, see e.g. Heubach and Mansour's monograph *Combinatorics of compositions and words* [19]. The corresponding pattern matching problems have not yet been studied.

This list contains only some of the open problems for permutation pattern matching and shows that the computational landscape of permutation patterns remains to a large extent terra incognita.

Acknowledgements

We wish to thank Henning Úlfarsson for drawing our attention to marked mesh, decorated and other types of patterns, Sergey Kitaev for helping us with notational questions and the anonymous referees for valuable comments and suggestions.

References

- [1] S. Ahal and Y. Rabinovich. On complexity of the subpattern problem. *SIAM Journal of Discrete Mathematics*, 22(2):629–649, 2008.
- [2] M. Albert, R. Aldred, M. Atkinson, and D. Holton. Algorithms for pattern involvement in permutations. In P. Eades and T. Takaoka, editors, *Algorithms and Computation*, volume 2223 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2001.
- [3] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
- [4] S. Avgustinovich, S. Kitaev, and A. Valyuzhenich. Avoidance of boxed mesh patterns on permutations. *Discrete Applied Mathematics*, 161(1-2):43–51, 2013.
- [5] E. Babson and E. Steingrímsson. Generalized permutation patterns and a classification of the mahonian statistics. *Sém. Lothar. Combin*, 44:117, 2000.

- [6] M. Bóna. *Combinatorics of permutations*. Discrete Mathematics and Its Applications. Chapman & Hall/CRC, 2004.
- [7] P. Bose, J. F. Buss, and A. Lubiw. Pattern matching for permutations. *Information Processing Letters*, 65(5):277 – 283, 1998.
- [8] M. Bousquet-Mélou, A. Claesson, M. Dukes, and S. Kitaev. $(2+2)$ -free posets, ascent sequences and pattern avoiding permutations. *Journal of Combinatorial Theory Series A*, 117(7):884–909, 2010.
- [9] P. Brändén and A. Claesson. Mesh patterns and the expansion of permutation statistics as sums of permutation patterns. *The Electronic Journal of Combinatorics*, 18(2):P5, 2011.
- [10] M.-L. Bruner and M. Lackner. A fast algorithm for permutation pattern matching based on alternating runs. In F. V. Fomin and P. Kaski, editors, *SWAT*, volume 7357 of *Lecture Notes in Computer Science*, pages 261–270. Springer, 2012.
- [11] M.-S. Chang and F.-H. Wang. Efficient algorithms for the maximum weight clique and maximum weight independent set problems on permutation graphs. *Information Processing Letters*, 43(6):293–295, 1992.
- [12] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
- [13] S. Elizalde and M. Noy. Consecutive patterns in permutations. *Advances in Applied Mathematics*, 30(1):110–125, 2003.
- [14] J. Flum and M. Grohe. Model-checking problems as a basis for parameterized intractability. *Logical Methods in Computer Science*, 1(1), 2005.
- [15] J. Flum and M. Grohe. *Parameterized complexity theory*. Springer Berlin / Heidelberg, 2006.
- [16] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.
- [17] S. Guillemot and D. Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 82–101.
- [18] S. Guillemot and S. Vialette. Pattern matching for 321-avoiding permutations. In Y. Dong, D.-Z. Du, and O. Ibarra, editors, *Algorithms and Computation*, volume 5878 of *Lecture Notes in Computer Science*, pages 1064–1073. Springer Berlin / Heidelberg, 2009.

- [19] S. Heubach and T. Mansour. *Combinatorics of compositions and words*. Chapman & Hall/CRC, 2009.
- [20] L. Ibarra. Finding pattern matchings for permutations. *Information Processing Letters*, 61(6):293–295, 1997.
- [21] S. Kitaev. *Patterns in Permutations and Words*. Springer Berlin / Heidelberg, 2011.
- [22] D. E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms*. Addison-Wesley, 1968.
- [23] M. Kubica, T. Kulczyński, J. Radoszewski, W. Rytter, and T. Wale. A linear time algorithm for consecutive permutation pattern matching. *Information Processing Letters*, 113(12):430 – 433, 2013.
- [24] H. Magnússon and H. Úlfarsson. Algorithms for discovering and proving theorems about permutation patterns. *ArXiv e-prints*, Nov. 2012.
- [25] E. Mäkinen. On the longest upsequence problem for permutations. *International journal of computer mathematics*, 77(1):45–53, 2001.
- [26] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics And Its Applications. Oxford University Press, 2006.
- [27] C. H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.
- [28] C. Schensted. Longest increasing and decreasing subsequences. *Classic Papers in Combinatorics*, pages 299–311, 1987.
- [29] R. Simion and F. W. Schmidt. Restricted permutations. *European Journal of Combinatorics*, 6:383–406, 1985.
- [30] E. Steingrímsson. Generalized permutation patterns - a short survey. *LMS Lecture Note Series*, 376:137–152, 2010.
- [31] H. Úlfarsson. A unification of permutation patterns related to Schubert varieties. *ArXiv e-prints*, Feb. 2010.
- [32] H. Úlfarsson. Describing west-3-stack-sortable permutations with permutation patterns. *Séminaire Lotharingien de Combinatoire*, 67:20, 2012.
- [33] J. West. *Permutations with forbidden subsequences, and, stack-sortable permutations*. PhD thesis, Massachusetts Institute of Technology, 1990.