A Fast Algorithm for Permutation Pattern Matching Based on Alternating Runs

Marie-Louise Bruner[1] and Martin Lackner[2,⋆]

[1] Institute of Discrete Mathematics and Geometry, Vienna University of Technology, Austria
marie-louise.bruner@tuwien.ac.at
[2] Institute of Information Systems, Vienna University of Technology, Austria
lackner@dbai.tuwien.ac.at

Abstract. The NP-complete PERMUTATION PATTERN MATCHING problem asks whether a permutation P can be matched into a permutation T. A matching is an order-preserving embedding of P into T. We present a fixed-parameter algorithm solving this problem with an exponential worst-case runtime of $\mathcal{O}^*(1.79^{\mathsf{run}(T)})$, where $\mathsf{run}(T)$ denotes the number of alternating runs of T. This is the first algorithm that improves upon the $\mathcal{O}^*(2^n)$ runtime required by brute-force search without imposing restrictions on P and T. Furthermore we prove that – under standard complexity theoretic assumptions – such a fixed-parameter tractability result is not possible for $\mathsf{run}(P)$.

1 Introduction

The concept of pattern avoidance (and, closely related, pattern matching) in permutations arose in the late 1960ies. It was in an exercise of his *Fundamental algorithms* [12] that Knuth asked which permutations could be sorted using a single stack. The answer is simple: These are exactly the permutations avoiding the pattern 231 and they are counted by the Catalan numbers. By avoiding (resp. containing) a certain pattern the following is meant: The permutation $\pi = 53142$ (written in one-line representation) contains the pattern 231, since the subsequence 342 of π is order-isomorphic to 231. We call the subsequence 342 a matching of 231 into π. On the other hand, π avoids the pattern 123 since it contains no increasing subsequence of length three. Since 1985, when the first systematic study of *Restricted Permutations* [17] was published by Simion and Schmidt, the area of pattern avoidance in permutations has become a rapidly growing field of discrete mathematics, more specifically of (enumerative) combinatorics [4, 11].

This paper takes the viewpoint of computational complexity. Computational aspects of pattern avoidance, in particular the analysis of the PERMUTATION PATTERN MATCHING (PPM) problem, have received far less attention than enumerative questions until now. The PPM problem is defined as follows:

PERMUTATION PATTERN MATCHING (PPM)	
Instance:	A permutation T (the text) of length n and a permutation P (the pattern) of length $k \leq n$.
Question:	Is there a matching of P into T?

⋆ The second author was supported by the Austrian Science Fund (FWF): P20704-N18.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 261–270, 2012.

In [5] it was shown that PPM is in general NP-complete. From this result follows a trivial brute-force algorithm checking every length k subsequence of T. Its runtime is in $\mathcal{O}^*(2^n)$, i.e. is bounded by $2^n \cdot poly(n)$. To the best of our knowledge, no algorithm with a runtime of $\mathcal{O}^*((2-\epsilon)^n)$ without restrictions on P and T is known yet. If such restrictions are imposed, improvements have been achieved. There are polynomial time algorithms in case of a *separable* pattern [2, 5, 10]. Separable permutations avoid both 3142 and 2413. 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 [16] and in $\mathcal{O}(n \log\log n)$-time for permutations [7, 14]. PPM can be solved in $\mathcal{O}(n \log n)$-time for all patterns of length four [2]. An $\mathcal{O}(k^2 n^6)$-time algorithm is presented in [9] for the case that both the text and the pattern are 321-avoiding.

In this paper we tackle the problem of solving PPM faster than $\mathcal{O}^*(2^n)$ for arbitrary P and T. We achieve this by exploiting the decomposition of permutations into alternating runs. As an example, the permutation $\pi = 53142$ has three alternating runs: 531 (down), 4 (up) and 2 (down). We denote this number of ups and downs in a permutation π by $\mathsf{run}(\pi)$. Alternating runs are a fundamental permutation statistic and had been studied already in the late 19th century by André [3]. An important result was the characterization of the distribution of $\mathsf{run}(\pi)$ in a random permutation: asymptotically, $\mathsf{run}(\pi)$ is normal with mean $\frac{1}{3}(2|\pi|-1)$ [13]. Despite the importance of alternating runs within the study of permutations, the connection to PPM has so far not been explored.

In detail the contributions of this paper are the following:

- We present a fixed-parameter algorithm for PPM with an exponential runtime of $\mathcal{O}^*(1.79^{\mathsf{run}(T)})$. Since the combinatorial explosion is confined to $\mathsf{run}(T)$, this algorithm performs especially well when T has few alternating runs. Indeed, the runtime depends only polynomially on n, the length of T.
- Since $\mathsf{run}(T) \leq n$, this algorithm also solves PPM in time $\mathcal{O}^*(1.79^n)$. This is a major improvement over the brute-force algorithm.
- Furthermore, we analyze this algorithm with respect to $\mathsf{run}(P)$. We obtain a runtime of $\mathcal{O}^*\big(\big(n^2/2\mathsf{run}(P)\big)^{\mathsf{run}(P)}\big)$. In the framework of parameterized complexity theory this runtime proves XP membership for $\mathsf{run}(P)$.[1]
- Finally, we prove that this XP result cannot be substantially improved. We prove that – under standard complexity theoretic assumptions – no fixed-parameter algorithm exists with respect to $\mathsf{run}(P)$, i.e. no algorithm with runtime $\mathcal{O}^*(c^{\mathsf{run}(P)})$ for some constant c may be hoped for.

Proofs had to be omitted in this version – we refer the reader to the full version of this paper [6]. Runtime calculations can be found there as well.

2 Preliminaries

Permutations. For any $m \in \mathbb{N}$, let $[m]$ denote the set $\{1, \dots, m\}$ and $[0, m]$ denote $\{0, 1, \dots, m\}$. A permutation π on the set $[m]$ can be seen as the sequence $\pi(1), \pi(2),$

[1] XP membership also follows from results in [1] and a lemma shown in [6].

$\ldots, \pi(m)$. 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.

Definition 1. *Let P (the pattern) be a permutation of length k. We say that the permutation T (the text) of length n* contains *P as a pattern or that P* can be matched into *T if we can find a subsequence of T that is order-isomorphic to P. If there is no such subsequence we say that T* avoids the pattern *P. Matching P into T thus consists in finding a monotonically increasing map $\varphi : [k] \to [n]$ so that the sequence $\varphi(P)$, defined as $\big(\varphi(P(i))\big)_{i\in[k]}$, is a subsequence of T.*

Every permutation π on $[m]$ defines a total order $\prec_\pi$ on $[m]$. We write $i \prec_\pi j$ iff $\pi^{-1}(i) < \pi^{-1}(j)$, i.e. the value i stands to the left of the value j in π. When considering the minimum (maximum) of a subset $\mathcal{S} \subseteq [m]$ with respect to $\prec_\pi$, we write $\min_\pi \mathcal{S}$ ($\max_\pi \mathcal{S}$).

We discern two types of local extrema in permutations: valleys and peaks. A *valley* of a permutation π is an element $\pi(i)$ for which it holds that $\pi(i-1) > \pi(i)$ and $\pi(i) < \pi(i+1)$. If $\pi(i-1)$ or $\pi(i+1)$ is not defined, we still speak of valleys. The set $Val(\pi)$ contains all valleys of π. Similarly, a *peak* denotes an element $\pi(i)$ for which it holds that $\pi(i-1) < \pi(i)$ and $\pi(i) > \pi(i+1)$.

Valleys and peaks partition a permutation into contiguous monotone subsequences, so-called *(alternating) runs*. The first run of a given permutation starts with its first element (which is also the first local extremum) and ends with the second local extremum. The second run starts with the following element and ends with the third local extremum. Continuing in this way, every element of the permutation belongs to exactly one alternating run. Observe that every alternating run is either increasing or decreasing. We therefore distinguish between *runs up* and *runs down*. Note that runs up always end with peaks and runs down always end with valleys. The parameter $\mathsf{run}(\pi)$ counts the number of alternating runs in π. Hence $\mathsf{run}(\pi) + 1$ equals the number of local extrema in π. These definitions can be analogously extended to subsequences of permutations.

Example 2. In the permutation 1 8 12 4 7 11 6 3 2 9 5 10 the valleys are 1, 4, 2 and 5 and the peaks are 12, 11, 9 and 10. A decomposition into alternating runs is given by: 1 8 12|4|7 11|6 3 2|9|5|10. A graphical representation can be found in Figure 1 on page 265. ⊣

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 $|I|$. A problem parameterized by a parameter p is *fixed-parameter tractable* (or in FPT) if there is an algorithm solving it in time $\mathcal{O}(f(p) \cdot |I|^c)$, where f is a computable function and c a constant. 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. The classes W[1] $\subseteq$ W[2] $\subseteq \ldots$ build the so-called W-hierarchy. It is conjectured (and widely believed) that W[1] $\neq$ FPT. Therefore showing W[1]-hardness can be considered as evidence that a problem is not fixed-parameter tractable. A problem is in XP with respect to a parameter k if

it can be solved in time $\mathcal{O}(|I|^{f(k)})$ where f is a computable function. It holds that $\mathsf{FPT} \subseteq \mathsf{W[1]} \subseteq \mathsf{W[2]} \subseteq \ldots \subseteq \mathsf{XP}$. For details we refer the reader to [8, 15].

3 The Alternating Run Algorithm

We start with an outline of the alternating run algorithm. Its description consists of two parts. In Part 1 we introduce so-called *matching functions*. These functions map runs in P to sequences of adjacent runs in T. The intention behind matching functions is to restrict the search space to certain length k subsequences, namely to those where all elements in a run in P are mapped to elements in the corresponding sequences of runs in T. In Part 2 a dynamic programming algorithm is described. It checks for every matching function whether it is possible to find a compatible matching. This is done by finding a small set of representative elements to which the element 1 can be mapped to, then – for a given choice for 1 – finding representative values for 2, and so on.

Theorem 3. *The alternating run algorithm solves* PPM *in time* $\mathcal{O}^*(1.79^{\mathsf{run}(T)})$. *Therefore* PPM *parameterized by* $\mathsf{run}(T)$ *is in* FPT.

Corollary 4. *The alternating run algorithm solves* PPM *in time* $\mathcal{O}^*(1.79^n)$ *where* n *is the length of the text* T.

Proposition 5. PPM *is in* XP *with respect to the parameter* $\mathsf{run}(P)$ *since the alternating run algorithm solves* PPM *in time* $\mathcal{O}^*\big(\big(\frac{n^2}{2\mathsf{run}(P)}\big)^{\mathsf{run}(P)}\big)$.

Throughout this section the input instance (T_{ex}, P_{ex}) which is given by $T_{ex} = 1\,8\,12\,4\,7\,11\,6\,3\,2\,9\,5\,10$ and $P_{ex} = 2\,3\,1\,4$ serves as a running example.

Part 1: Matching Functions. We introduce the concept of matching functions. These are functions from $[\mathsf{run}(P)]$, i.e. runs in P, to sequences of adjacent runs in T. For a given matching function F the search space in T is restricted to matchings where an element i contained in the j-th run in P is matched to an element in $F(j)$. Two adjacent runs in P are mapped to sequences of runs that overlap with exactly one run. This overlap is necessary since elements in different runs in P may be matched to elements in the same run in T. More precisely, valleys and peaks in P might be matched to the same run in T as their successors (see the following example).

Example 6. In Figure 1 P_{ex} (left-hand side) and T_{ex} (right-hand side) are depicted together with a matching function F. A matching compatible with F is given by $4\,6\,2\,9$. We can see that the elements 6 and 2 lie in the same run in T_{ex} even though 3 (a peak) and 1 (its successor) lie in different runs in P_{ex}. ⊣

Definition 7. *A matching function* F *maps an element of* $[\mathsf{run}(P)]$ *to a subsequence of* T. *It has to satisfy the following properties for all* $i \in [\mathsf{run}(P)]$.

(P1) $F(i)$ *is a contiguous subsequence of* T.
(P2) If the i*-th run in* P *is a run up (down),* $F(i)$ *starts with an element following a valley (peak) or the first element in* T *and ends with a valley (peak) or the last element in* T.

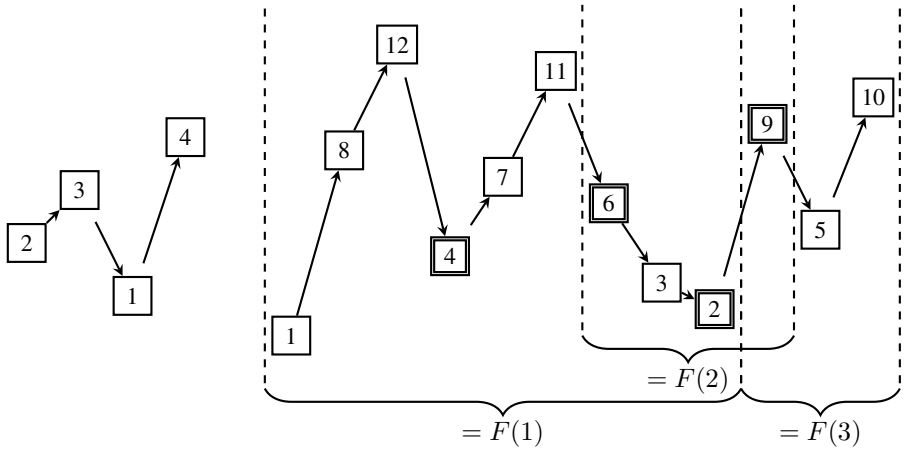


Fig. 1. P_{ex} and T_{ex} together with a matching function F and the compatible matching 4 6 2 9

Fig. 2. A sketch of a matching function and its M- and W-shaped subsequences

(P3) $F(1)$ *starts with the first and* $F(\mathsf{run}(P))$ *ends with the last element in* T.
(P4) $F(i)$ *and* $F(i+1)$ *have one run in common:* $F(i+1)$ *starts with the leftmost element in the last run in* $F(i)$.

Property (P2) implies that every run up is matched into an M-shaped sequence of runs of the form up-down-up-...-up-down (if the run up is the first or the last run in P the sequence might start or end differently) and every run down is matched into a W-shaped sequence of runs of the form down-up-down-...-down-up (again, if the run down is the first or the last run in P, the sequence might start or end differently). These M- and W-shaped sequences and their overlap are sketched in Figure 2.

The following lemma is essential as it enables us to iterate over all matching functions in fpt time.

Lemma 8. *There are at most* $\sqrt{2}^{\mathsf{run}(T)}$ *functions from* $[\mathsf{run}(P)]$ *to subsequences of* T *that satisfy (P1) to (P4).*

Proof idea. A matching function F can be uniquely determined by fixing the first run up in each $F(i)$. There are at most $\lceil \mathsf{run}(T)/2 \rceil$ runs up in T. □

Part 2: Finding a Matching. When checking whether T contains P as a pattern, it is sufficient to test for all matching functions whether there exists a *compatible* matching. A matching is compatible with a matching function F if an element i contained in the j-th run in P is matched to an element in $F(j)$. This is checked by a dynamic programming algorithm. The algorithm computes the data structure X_κ for each $\kappa \in$

$[k]$. X_κ is a subset of $[0,n]^{\mathsf{run}(P)}$ and contains representative choices for the matching of the largest element in each run in P that is $\leq \kappa$. (X_κ does of course depend on F but we omit this in the notation.)

Let us explain what is meant by representative choices. We search for a compatible matching of P into T by successively determining possible elements for $1, 2, \ldots, k$. Given a choice for $\kappa \in [k]$, possible choices for $\kappa + 1$ are necessarily larger. In addition, it is always preferable to choose elements that are as small as possible. To be more precise: if $\nu \in [n]$ has been chosen for $\kappa \in [k]$, we merely need to consider the valleys of the subsequence of T containing all elements larger than ν. Indeed, if any matching of P into T can be found, it is also possible to find a matching that only involves valleys in the above-mentioned subsequences. Therefore our algorithm will only consider such valleys – we call these elements representative. As an example, consider again Figure 1. Here $4\,6\,3\,10$ is a matching of P_{ex} into T_{ex} where the elements 3 and 10 are not representative. This can be seen since 3 is not a valley and 10 is not a valley in the subsequence consisting of elements larger than 6. However, this matching can be represented by the matching $4\,6\,2\,9$ that only involves representative elements (3 is represented by 2; 10 by 9).

Furthermore, observe that when successively determining possible elements for $1, 2, \ldots, k$, we move from left to right in runs up and from right to left in runs down. Hence the chosen elements do not only have to be larger than the previously chosen element but also have to lie on the correct side of the previously chosen element in the same run. These observations are captured in the following definition.

Definition 9. *For a permutation π on $[n]$ and integers $i, j \leq n$, we define $\pi_{U(i,j)}$ ($\pi_{D(i,j)}$) as the subsequence of π consisting of all elements that are right (left) of j and larger than i. Then $URep(\pi, i, j) := Val(\pi_{U(i,j)})$ (resp. $DRep(\pi, i, j) := Val(\pi_{D(i,j)})$) corresponds to the set of* representative elements *for the case of a run up (resp. down).*

For an example, see Figure 3 where representative elements are shown for the permutation T_{ex}, $i = 3$ and $j = 2$.

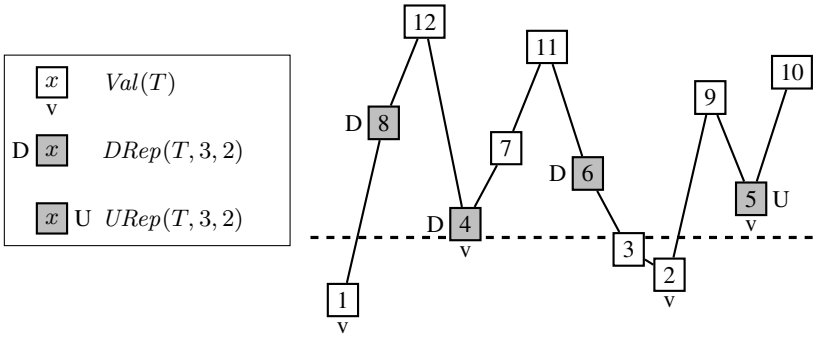


Fig. 3. Illustrating Definition 9

We now describe how the algorithm checks whether there is a matching from P into T compatible with the matching function F. The data structure X_κ consists of $\mathsf{run}(P)$-tuples with entries in $[0, n]$. The i-th component of a tuple in X_κ is a representative choice for the largest element $\leq \kappa$ that lies in the i-th run. Limiting X_κ to representative elements allows to bound its size by $1.262^{\mathsf{run}(T)}$. In order to achieve this upper bound, we impose the following conditions (C1) and (C2) and remove unnecessary elements by applying the rules (R1) and (R2). In order to state these conditions and rules, we write $r(\kappa) = i$ iff κ is contained in the i-th run in P. For notational convenience we define $r(0) := 1$.

First, we set $X_0 := \big\{(0, 0, \ldots, 0)\big\}$. The set X_κ is then constructed from $X_{\kappa-1}$ as follows. Let $\mathbf{x} = \big(x_1, \ldots, x_{\mathsf{run}(P)}\big) \in X_{\kappa-1}$. We now define $N_{\kappa,\mathbf{x}}$ to be the set of all $\nu \in [n]$ that satisfy (C1) and (C2). This set contains representative elements to which κ may be mapped to for the given $\mathbf{x}$.

(C1) It has to hold that $\nu \in \mathit{URep}(F(r(\kappa)), x_{r(\kappa-1)}, x_{r(\kappa)})$ in case κ lies in a run up and analogously $\nu \in \mathit{DRep}(F(r(\kappa)), x_{r(\kappa-1)}, x_{r(\kappa)})$ in case κ lies in a run down.

This condition ensures that ν is larger than the previously chosen element for $\kappa - 1$, i.e. larger than $x_{r(\kappa-1)}$. Furthermore, it enforces ν to lie on the correct side of $x_{r(\kappa)}$, the previously chosen element in this run. Instead of considering all such elements in $F(r(\kappa))$ we only take into account representative elements.

(C2) If κ is *not* the largest element in its run in P, there has to exist $\xi \in F(r(\kappa))$ with $\nu < \xi$ and $\nu \prec_T \xi$ for κ appearing in a run up ($\xi \prec_T \nu$ for κ appearing in a run down).

This condition excludes a choice for κ that cannot lead to a matching. A non-maximal element in a run up (down) in P has to be mapped to an element having larger elements to its right (left). We therefore exclude elements in the rightmost (leftmost) run of $F(r(\kappa))$ if this is a run down (up). Condition (C2) is necessary to obtain the runtime bounds for the dynamic programming algorithm.

As an intermediate step let $X'_\kappa := \{\mathbf{x}(\nu) \mid \mathbf{x} \in X_{\kappa-1} \text{ and } \nu \in N_{\kappa,\mathbf{x}}\}$, where $\mathbf{x}(\nu) := \big(x_1, \ldots, x_{r(\kappa)-1}, \nu, x_{r(\kappa)+1}, \ldots, x_{\mathsf{run}(P)}\big)$. The tuple $\mathbf{x}(\nu)$ thus differs from $\mathbf{x}$ only at the $r(\kappa)$-th position. Note that two different elements $\mathbf{x}$ and $\mathbf{x}'$ in $X_{\kappa-1}$ may lead to the same element $\mathbf{x}(\nu) = \mathbf{x}'(\nu)$ in X_κ if they only differ in $x_{r(\kappa)}$. Rule (R1) describes how to compute X_κ from X'_κ. Stating it requires the following definition.

Definition 10. *Let π be a permutation of length n. A subsequence of π consisting of a consecutive run down and run up (formed like a V) is called a* vale. *If π starts with a run up, this run is also considered as a vale and analogously if π ends with a run down. For two elements $\nu_1, \nu_2 \in [n]$, $\nu_1 \sim \nu_2$ if both lie in the same vale*[2]*. For two k-tuples $\mathbf{x}, \mathbf{y} \in [n]^k$, $\mathbf{x} \sim \mathbf{y}$ if for every $i \in [k]$ it holds that $x_i \sim y_i$. For a fixed set of k-tuples S and $\mathbf{x} \in S$, the equivalence class $[\mathbf{x}]_\sim$ is defined as all $\mathbf{y} \in S$ with $\mathbf{x} \sim \mathbf{y}$.*

(R1) We set $X_\kappa := \big\{\min_{(r(\kappa))}([\mathbf{x}]_\sim) \mid \mathbf{x} \in X'_\kappa\big\}$, where $\min_{(i)}(S)$ is the function picking the tuple in S with the smallest value at the i-th position. If this minimum is not unique, it arbitrarily picks one candidate.

[2] Note that every element in a permutation is contained in exactly one vale.

This rule is the key to prove the $1.262^{\mathsf{run}(T)}$ upper bound on $|X_\kappa|$. It is based on the observation that it is enough to keep a single tuple for each $[\mathbf{x}]_\sim$. This means that for a set of tuples with coinciding vales it is enough to consider one of them. We provide an intuition about the rule and its correctness in the following example.

Example 11. Consider the text permutation schematically represented in Figure 4. We are searching for representative choices for κ, an element lying in a run down. For κ', the previous element lying in the same run as κ, two representative elements are μ_1 (circle) and μ_2 (square). They lead to one representative element for $\kappa - 1$ each: if μ_1 has been chosen ν_1 is a representative element (circle) and if μ_2 has been chosen ν_2 is one. Following condition (C1), we find three representative elements for κ in $F(r(\kappa))$: ξ_1 (if ν_1 has been chosen), ξ_2 and ξ_3 (if ν_2 has been chosen).

We can now observe that it is not necessary to store all three representative elements for κ. Indeed, in the vale containing ξ_1 and ξ_2 we only need to keep track of ξ_1 since this is always a better choice than ξ_2. This can be seen in the following way: In general, elements that lie further to the right (left) in a run down (up) might be preferable since they leave more possibilities for future elements that are to be matched. Within a vale however, the horizontal position does not make any difference, it is only the vertical position that matters. Here, the elements left of ξ_2 and right of ξ_1 are not available for following choices even if we choose ξ_2 since they are smaller than ξ_2. However, the elements left of ξ_1 that are smaller than ξ_2 are only available if we choose ξ_1. ⊣

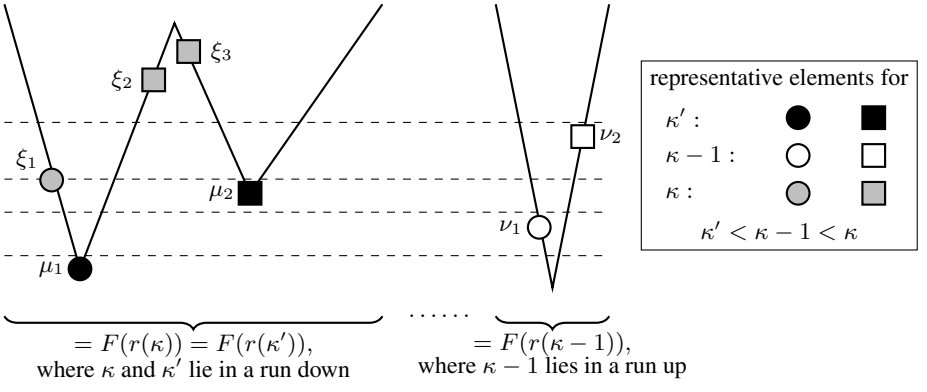


Fig. 4. Illustrating Rule (R1)

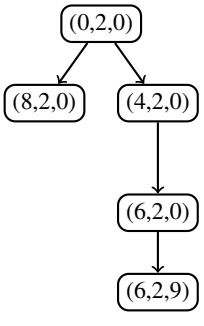
In the case that κ is the largest element in its run, it is enough to consider a single representative element in $F(r(\kappa))$. This is because the position of the element ν is no longer relevant since no further elements have to be chosen in this run. Hence the following data reduction is performed on X_κ.

(R2) Let $M_{\kappa,\mathbf{x}} := \{y_{r(\kappa)} \mid \mathbf{y} \in X_\kappa \wedge (y_i = x_i \; \forall i \neq r(\kappa))\}$. If κ is the largest element in its run, each $\mathbf{x} = (x_1, \ldots, x_{\mathsf{run}(P)}) \in X_\kappa$ is replaced by the tuple $(x_1, \ldots, x_{r(\kappa)-1}, \min(M_{\kappa,\mathbf{x}}), x_{r(\kappa)+1}, \ldots, x_{\mathsf{run}(P)})$.

As a consequence there are no two tuples in X_κ that only differ at the $r(\kappa)$-th position in this case.

Termination. For a given matching function F, the algorithm described in Part 2 terminates as soon as we have reached X_k. Observe that X_k is always empty if a previous X_κ was empty. If for any F the data structure X_k is non-empty, P can be matched into T.

Example 12. Let us demonstrate with the help of a simple example how the alternating run algorithm works. Consider the text T_{ex} and the pattern P_{ex}. In this example we consider the matching function F represented in Figure 1. Figure 5 depicts a successful run of the algorithm finding the matching 4 6 2 9. ⊣



The only valley in $F(r(1)) = F(2)$ is 2, therefore $N_{1,(0,0,0)} = \{2\}$.

There are 3 representative elements larger than 2 in $F(1)$: 8, 4 and 3. Since 2 is not the largest element in its run in P_{ex}, condition (C2) implies that 3 is ruled out. Thus $N_{2,(0,2,0)} = \{8, 4\}$. Rule (R1) yields $X_2 = X'_2$. Rule (R2) is not applicable.

We have $N_{3,(8,2,0)} = \{11\}$ and $N_{3,(4,2,0)} = \{7, 6\}$ implying $X'_3 = \{(11, 2, 0), (7, 2, 0), (6, 2, 0)\}$. Rule (R1) discards $(11, 2, 0)$ in favor of $(7, 2, 0)$. Finally, Rule (R2) is applicable here and discards $(7, 2, 0)$.

The only representative element larger than 6 in $F(3)$ is 9. The matching of $P_{ex} = 2314$ into T_{ex} found by the algorithm is thus 4629.

Fig. 5. The construction of $X_1, \ldots, X_4$ for our running example (T_{ex}, P_{ex})

4 W[1]-Hardness for the Parameter run(P)

Proposition 5 shows that the alternating run algorithm also yields an XP result with respect to $\mathsf{run}(P)$. The following theorem implies that this result cannot be improved to an FPT result – unless FPT = W[1]. This is shown by an fpt-reduction from the W[1]-complete CLIQUE problem.

Theorem 13. PPM *is* W[1]*-hard with respect to the parameter* $\mathsf{run}(P)$.

5 Future Work

Theorem 3 shows fixed-parameter tractability of PPM with respect to $\mathsf{run}(T)$. An immediate consequence is that any PPM instance can be reduced by polynomial time preprocessing to an equivalent instance – a kernel – of size depending solely on $\mathsf{run}(T)$. This raises the question whether even a polynomial-sized kernel exists. Another research direction is the study of further permutation statistics. The major open problem in this regard is whether PPM is fpt with respect to the length of P. Finally, our method of making use of alternating runs might lead to fast algorithms for other permutation based problems as well.

References

1. Ahal, S., Rabinovich, Y.: On complexity of the subpattern problem. SIAM J. Discrete Math. 22(2), 629–649 (2008)
2. Albert, M., Aldred, R., Atkinson, M., Holton, D.: Algorithms for Pattern Involvement in Permutations. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 355–367. Springer, Heidelberg (2001)
3. André, D.: Étude sur les maxima, minima et séquences des permutations. Ann. Sci. École Norm. Sup. 3(1), 121–135 (1884)
4. Bona, M.: Combinatorics of permutations. Discrete Mathematics and Its Applications. Chapman & Hall/CRC (2004)
5. Bose, P., Buss, J.F., Lubiw, A.: Pattern matching for permutations. Information Processing Letters 65(5), 277–283 (1998)
6. Bruner, M.L., Lackner, M.: A fast algorithm for permutation pattern matching based on alternating runs. CoRR (2012)
7. Chang, M.S., Wang, F.H.: Efficient algorithms for the maximum weight clique and maximum weight independent set problems on permutation graphs. Information Processing Letters 43(6), 293–295 (1992)
8. Flum, J., Grohe, M.: Parameterized complexity theory. Springer, Heidelberg (2006)
9. Guillemot, S., Vialette, S.: Pattern Matching for 321-Avoiding Permutations. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 1064–1073. Springer, Heidelberg (2009)
10. Ibarra, L.: Finding pattern matchings for permutations. Information Processing Letters 61(6), 293–295 (1997)
11. Kitaev, S.: Patterns in Permutations and Words. Springer, Heidelberg (2011)
12. Knuth, D.E.: The Art of Computer Programming. Fundamental Algorithms, vol. I. Addison-Wesley (1968)
13. Levene, H., Wolfowitz, J.: The covariance matrix of runs up and down. The Annals of Mathematical Statistics 15(1), 58–69 (1944)
14. Mäkinen, E.: On the longest upsequence problem for permutations. International Journal of Computer Mathematics 77(1), 45–53 (2001)
15. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics And Its Applications. Oxford University Press (2006)
16. Schensted, C.: Longest increasing and decreasing subsequences. Classic Papers in Combinatorics, pp. 299–311 (1987)
17. Simion, R., Schmidt, F.W.: Restricted permutations. European Journal of Combinatorics 6, 383–406 (1985)