A Fast Algorithm for Permutation Pattern Matching Based on Alternating Runs

Marie-Louise Bruner

(Joint work with Martin Lackner)

Vienna University of Technology

June 13, 2012
Permutation Patterns, Glasgow

Permutation Pattern Matching

Permutation Pattern Matching (PPM)

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

Permutation Pattern Matching

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

Is there an algorithm that runs faster than exponential time?

PPM is NP-complete

1993 (Bose, Buss, Lubiw): PPM is in general NP-complete.

PPM is NP-complete

1993 (Bose, Buss, Lubiw): PPM is in general NP-complete.

$\Rightarrow$ unless P = NP,
PPM cannot be solved in polynomial time

Tractable cases of PPM

- Pattern avoids both 3142 and 2413 $\mathcal{O}(kn^4)$
- $P = 12 \ldots k$ or $P = k \ldots 21$ $\mathcal{O}(n \log\log n)$
- P has length at most 4 $\mathcal{O}(n \log n)$
- Pattern and Text avoid 321 $\mathcal{O}(k^2 n^6)$

The general case

Anything better than the

$$\mathcal{O}^*(2^n)$$

runtime of brute-force search?

Parameterized Complexity Theory

Idea: confine the combinatorial explosion to a *parameter* of the input

Parameterized Complexity Theory

Idea: confine the combinatorial explosion to a *parameter* of the input

Parameterized Problem: $L \subseteq \Sigma^* \times \mathbb{N}$

Parameterized Complexity Theory

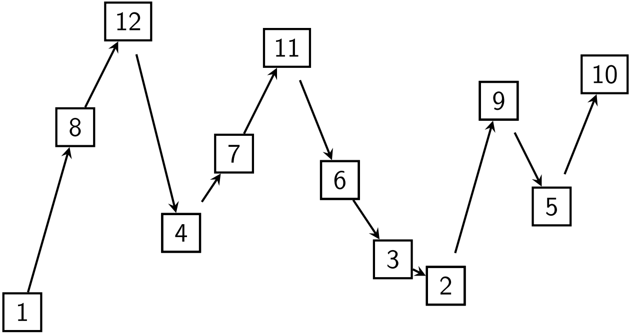
Idea: confine the combinatorial explosion to a *parameter* of the input

Parameterized Problem: $L \subseteq \Sigma^* \times \mathbb{N}$

Fixed-parameter tractability

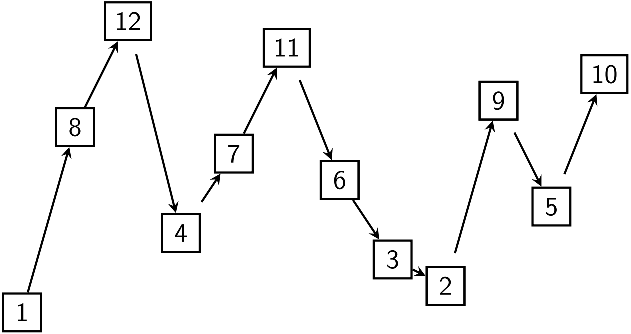
A parameterized problem L is *fixed-parameter tractable* if there is a computable function f and an integer c such that there is an algorithm solving L in time $\mathcal{O}(f(p) \cdot |I|^c)$.

Alternating runs



1 8 12 (up), 4 (down), 7 11 (up), 6 3 2 (down), 9 (up), 5 (down), 10 (up)

Alternating runs



1 8 12 (up), 4 (down), 7 11 (up), 6 3 2 (down), 9 (up), 5 (down), 10 (up)

Notation

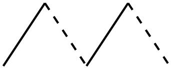
run(π)...the number of alternating runs in π,
$r(i) = j$ if i lies in the j-th run

The alternating run algorithm

- Matching functions:
  Reduce the search space
- Dynamic programming algorithm:
  Checks for every matching function whether there is a compatible matching

Matching functions

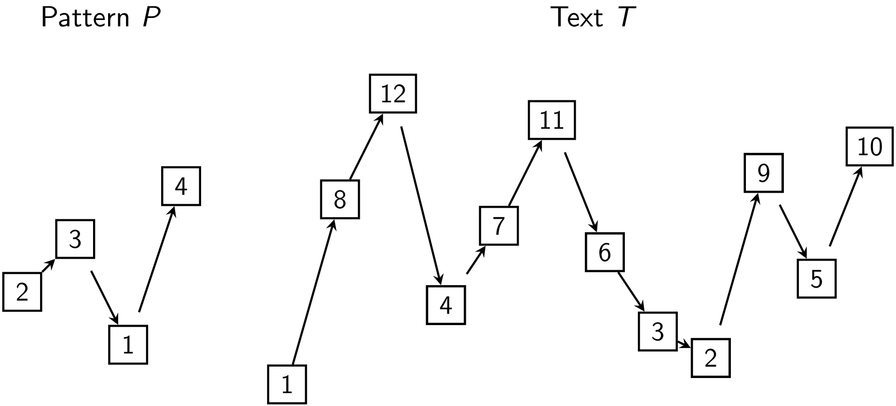
Pattern P



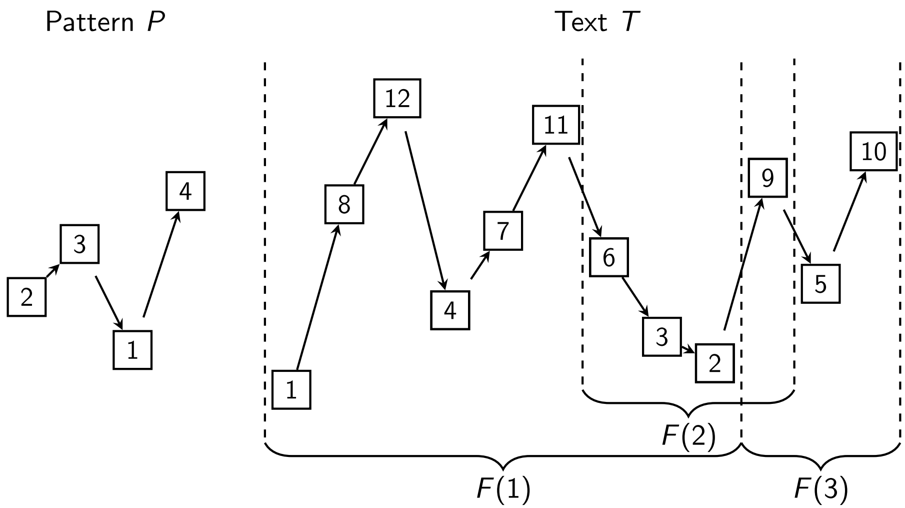
$\downarrow$ matching function $\downarrow$

Text T

Matching functions - an example



Matching functions - an example



The algorithm - finding a matching

The dynamic programming algorithm - Part 1

$X_0 := \{(0, \ldots, 0)\}$. For every $\kappa \in [k]$ the data structure X_κ is recursively constructed. For $\mathbf{x} \in X_{\kappa-1}$ we search for $\nu \in [n]$ satisfying:

- $\nu \in F(r(\kappa))$
- Among all elements that are larger than $x_{r(\kappa-1)}$ and on the correct side of $x_{r(\kappa)}$, ν has to be a valley.
- If κ is not the largest element in its run in P, there has to be another larger element in $F(r(\kappa))$ lying to the right (resp. left) of ν if κ lies in a run up (resp. down).

Place $\mathbf{x}_\nu := (x_1, \ldots, x_{r(\kappa)-1}, \nu, x_{r(\kappa)+1}, \ldots, x_{\text{run}(P)})$ in X'_κ.

The algorithm - finding a matching

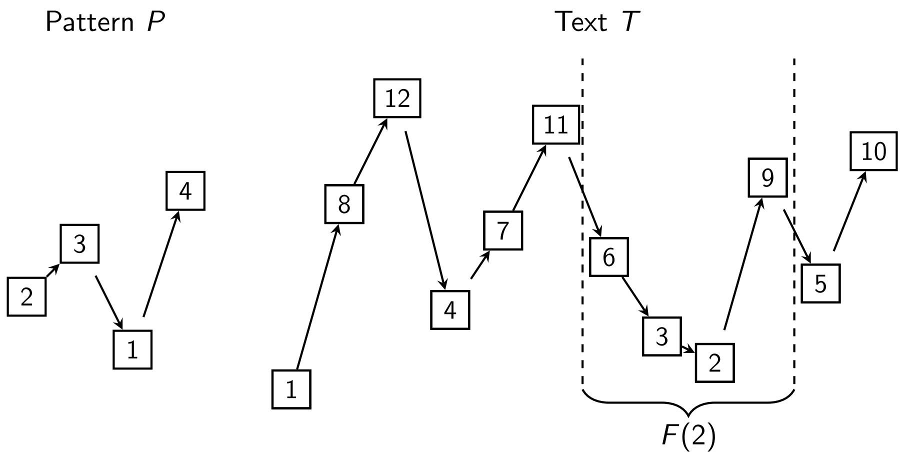
The dynamic programming algorithm - Part 1

X'_κ still consists of too many elements. In order to obtain the desired runtime bounds, we apply the following rules:

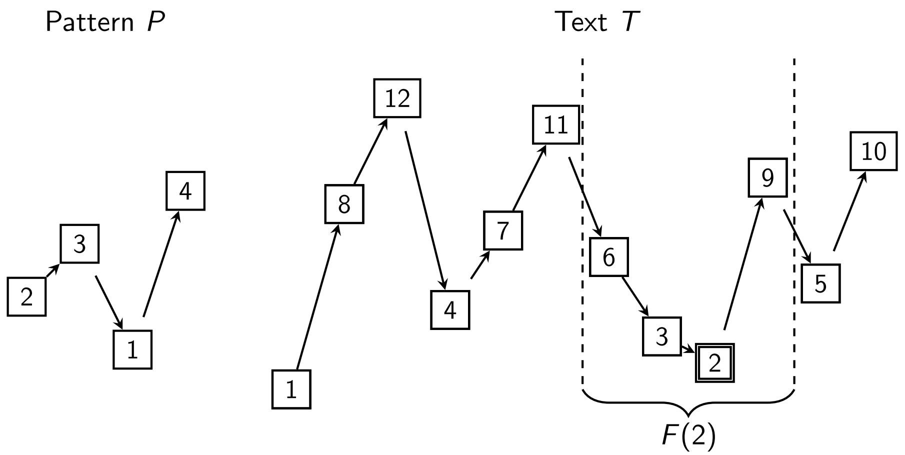
- We call a *vale* a subsequence of T consisting of a consecutive run down and up. If the entries of $\mathbf{x}_\nu$ and $\mathbf{y}_\mu$ all lie within the same vales, it is sufficient to keep one of the two.
- If κ is the largest element in its run in P, it is enough to keep one choice for every $\mathbf{x}$ in $X_{\kappa-1}$.

The remaining $\mathbf{x}_\nu$'s then form X_κ.

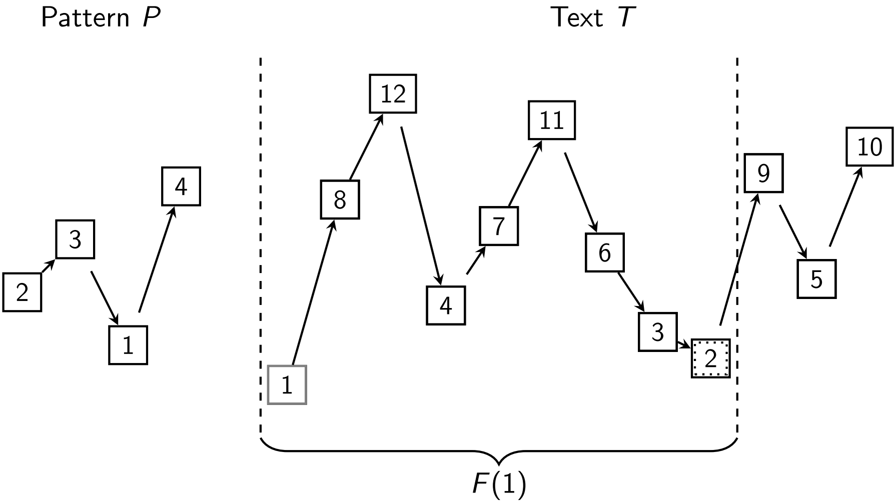
The algorithm - finding a matching



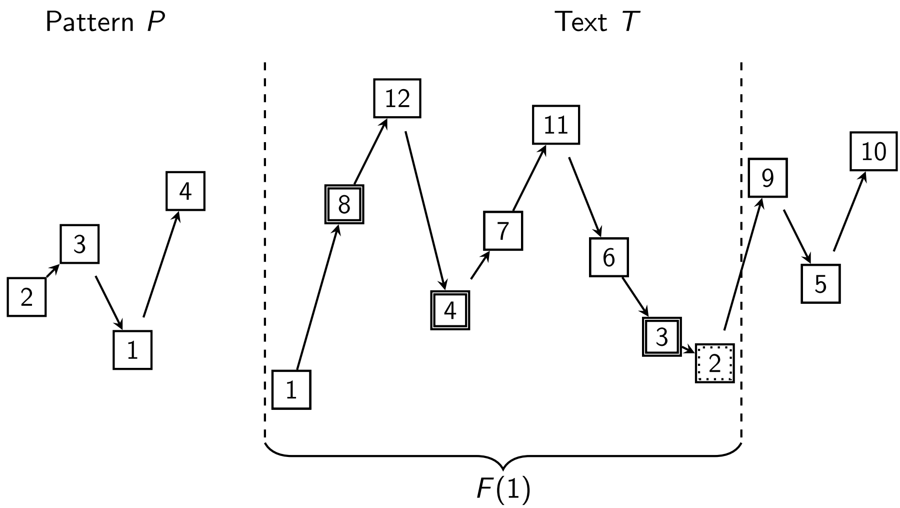
The algorithm - finding a matching



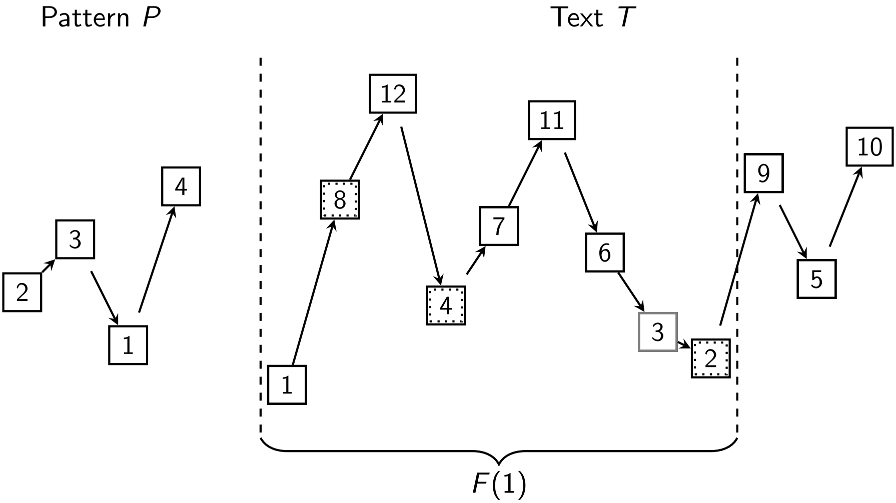
The algorithm - finding a matching



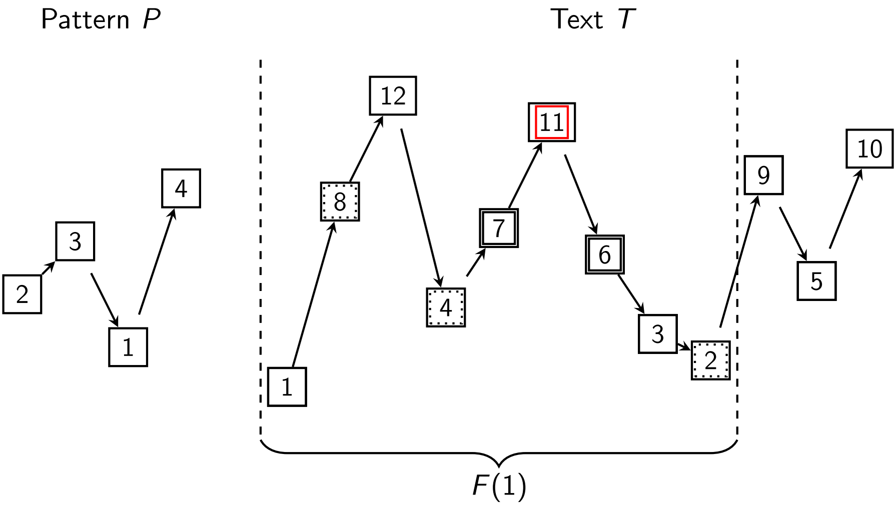
The algorithm - finding a matching



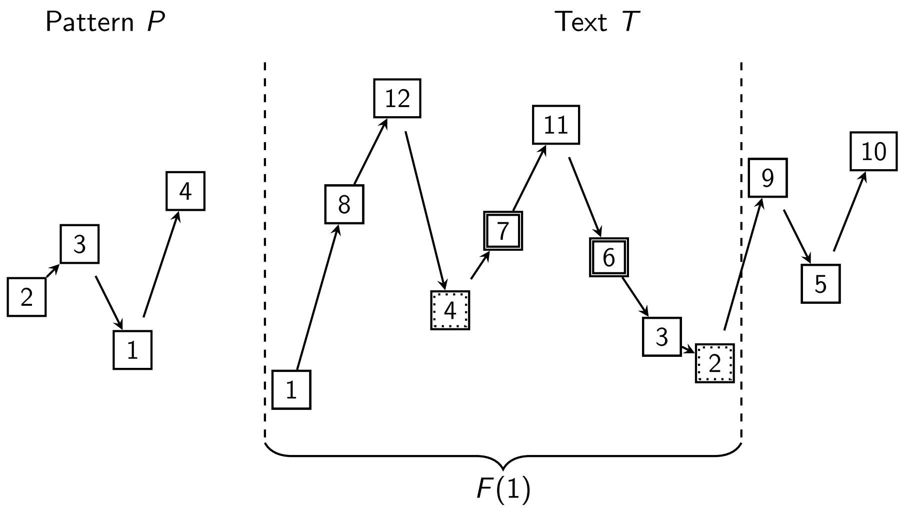
The algorithm - finding a matching



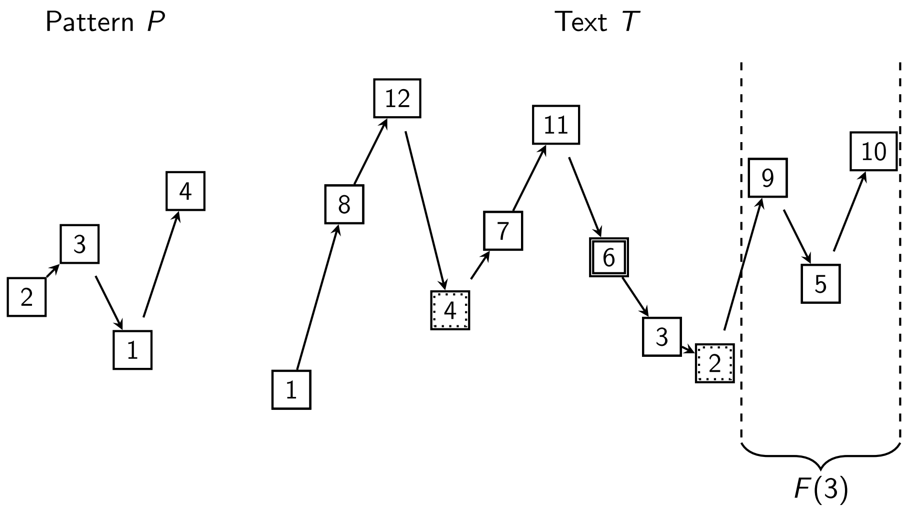
The algorithm - finding a matching



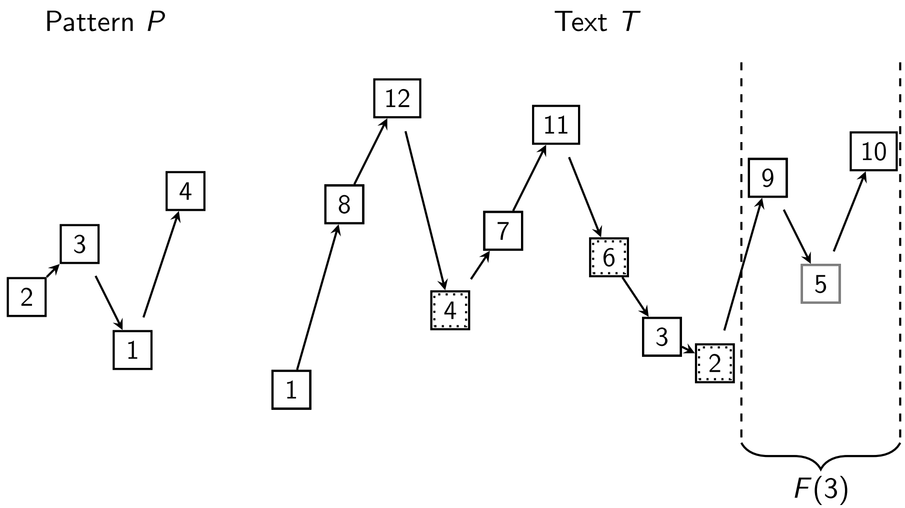
The algorithm - finding a matching



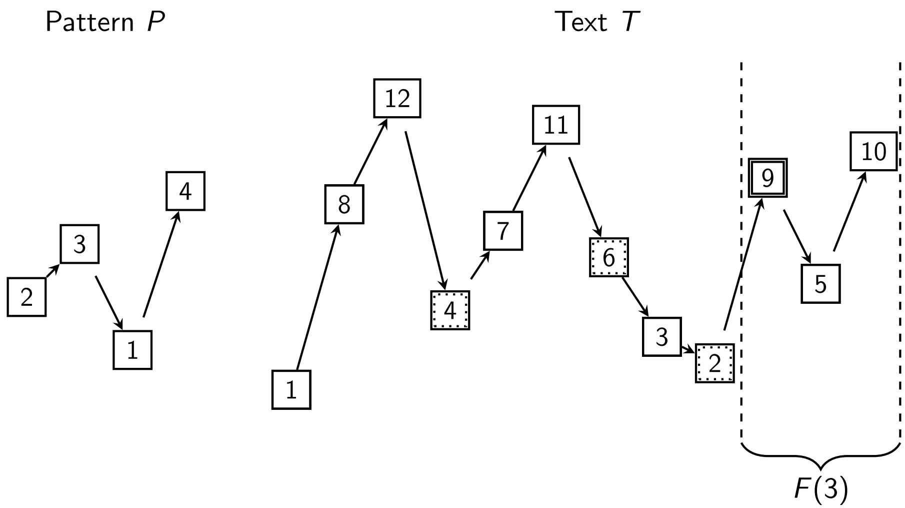
The algorithm - finding a matching



The algorithm - finding a matching



The algorithm - finding a matching



Runtime

Matching functions: $\sqrt{2}^{\mathrm{run}(T)}$

Dynamic programming algorithm: $\mathcal{O}^*(1.2611^{\mathrm{run}(T)})$

Runtime

Matching functions: $\sqrt{2}^{\mathsf{run}(T)}$

Dynamic programming algorithm: $\mathcal{O}^*(1.2611^{\mathsf{run}(T)})$

In total: $\mathcal{O}^*(1.784^{\mathsf{run}(T)})$

Runtime

Matching functions: $\sqrt{2}^{\mathrm{run}(T)}$

Dynamic programming algorithm: $\mathcal{O}^*(1.2611^{\mathrm{run}(T)})$

In total: $\mathcal{O}^*(1.784^{\mathrm{run}(T)})$

$\rightarrow$ This is a fixed-parameter tractable (FPT) algorithm,
i.e. a runtime of $f(k) \cdot n^c$.

Runtime

Matching functions: $\sqrt{2}^{\mathrm{run}(T)}$

Dynamic programming algorithm: $\mathcal{O}^*(1.2611^{\mathrm{run}(T)})$

In total: $\mathcal{O}^*(1.784^{\mathrm{run}(T)})$

$\rightarrow$ This is a fixed-parameter tractable (FPT) algorithm,
i.e. a runtime of $f(k) \cdot n^c$.

Since $\mathrm{run}(T) \leq n$, we also obtain $\mathcal{O}^*(1.784^n)$

Alternating runs in the pattern run(P)

$\mathcal{O}^*(1.784^{\mathrm{run}(T)})$ FPT, i.e. $f(k) \cdot n^c$

$\mathcal{O}^*\left(\left(\frac{n^2}{2\mathrm{run}(P)}\right)^{\mathrm{run}(P)}\right)$ XP, i.e. $n^{f(k)}$

Alternating runs in the pattern run(P)

$\mathcal{O}^*(1.784^{\mathrm{run}(T)})$ FPT, i.e. $f(k) \cdot n^c$

$\mathcal{O}^*\left(\left(\frac{n^2}{2\mathrm{run}(P)}\right)^{\mathrm{run}(P)}\right)$ XP, i.e. $n^{f(k)}$

no FPT result possible (W[1]-hardness)

Conclusion

What have we done?

We found

- a fast algorithm for the parameter $\text{run}(T)$,
- a slow algorithm for the parameter $\text{run}(P)$.

Conclusion

What have we done?

We found

- a fast algorithm for the parameter $\mathrm{run}(T)$,
- a slow algorithm for the parameter $\mathrm{run}(P)$.

Future work

- PPM parameterized by some other parameter of P? By $k = \mathrm{run}(P)$?

Conclusion

What have we done?

We found

- a fast algorithm for the parameter $\mathsf{run}(T)$,
- a slow algorithm for the parameter $\mathsf{run}(P)$.

Future work

- PPM parameterized by some other parameter of P? By $k = \mathsf{run}(P)$?
- Other permutation statistics in the text?