Counting Permutation Patterns with Multidimensional Trees

Gal Beniamini
The Hebrew University of Jerusalem, Israel

Nir Lavee
The Hebrew University of Jerusalem, Israel

Abstract

We consider the well-studied *pattern-counting problem*: given a permutation $\pi \in \mathbb{S}_n$ and an integer $k > 1$, count the number of order-isomorphic occurrences of every pattern $\tau \in \mathbb{S}_k$ in π.

Our first result is an $\widetilde{\mathcal{O}}(n^2)$-time algorithm for $k = 6$ and $k = 7$. The proof relies heavily on a new family of graphs that we introduce, called *pattern-trees*. Every such tree corresponds to an integer linear combination of permutations in $\mathbb{S}_k$, and is associated with linear extensions of partially ordered sets. We design an evaluation algorithm for these combinations, and apply it to a family of linearly-independent trees. For $k = 8$, we show a barrier: the subspace spanned by trees in the previous family has dimension exactly $|\mathbb{S}_8| - 1$, one less than required.

Our second result is an $\widetilde{\mathcal{O}}(n^{7/4})$-time algorithm for $k = 5$. This algorithm extends the framework of pattern-trees by speeding-up their evaluation in certain cases. A key component of the proof is the introduction of pair-rectangle-trees, a data structure for dominance counting.

2012 ACM Subject Classification Mathematics of computing → Permutations and combinations; Mathematics of computing → Combinatorial algorithms; Theory of computation → Pattern matching

Keywords and phrases Pattern counting, patterns, permutations

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.24

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2407.04971` [4]

Supplementary Material *Software*: `https://github.com/perm-patterns-icalp2025/pattern-counting` [5], archived at `swh:1:dir:cbdd126fda7252b2c2a212bc2aa5c16abfacc3b0`

1 Introduction

A permutation $\tau \in \mathbb{S}_k$ occurs in a permutation $\pi \in \mathbb{S}_n$ if there exist k points in π that are order-isomorphic to τ. By way of example, in $\overline{1}3\overline{42} \in \mathbb{S}_4$,[1] the overlined points form an occurrence of $132 \in \mathbb{S}_3$. The number of occurrences $\#\tau(\pi)$ of a permutation $\tau \in \mathbb{S}_k$ (a *pattern*) within a larger permutation $\pi \in \mathbb{S}_n$ has been the basis of many interesting questions, both combinatorial and algorithmic.

In a classical result, MacMahon [24] proved that the number of permutations $\pi \in \mathbb{S}_n$ that *avoid* the pattern 123 (i.e., $\#123(\pi) = 0$) is counted by the Catalan numbers. Another classical result is the well-known Erdős-Szekeres theorem [16], which states that any permutation of size $(s-1)(\ell-1)+1$ cannot simultaneously avoid both $(1, \ldots, s)$ *and* $(\ell, \ldots, 1)$. These early results gave rise to an entire field of study regarding pattern avoidance, c.f. [26, 23, 27]. One particularly noteworthy result is Marcus and Tardos' resolution of the

[1] Throughout this paper, permutations are written in one-line notation. If they are short, we omit the parenthesis.

Stanley-Wilf conjecture [25]: for any fixed pattern $\tau \in \mathbb{S}_k$, the growth rate of the number of permutations $\pi \in \mathbb{S}_n$ avoiding τ is bounded by $c(\tau)^n$, where $c(\tau)$ is a constant depending only on τ.

Pattern avoidance can also be cast as an algorithmic problem. The *permutation pattern matching* problem is the task of determining, given a pattern $\tau \in \mathbb{S}_k$ and a permutation $\pi \in \mathbb{S}_n$, whether π avoids τ. What is the computational complexity of this task? Trivial enumeration over all k-tuples of points yields an $\mathcal{O}(k \cdot n^k)$-time algorithm. This bound has been improved upon by a long line of works: Albert et al. [2] lowered the bound to $\mathcal{O}(n^{2k/3+1})$, Ahal and Rabinovich [1] to $\mathcal{O}(n^{(0.47+o(1))k})$, and finally Guillemot and Marx [21] established the fixed-parameter tractability of the problem, i.e., whenever k is *fixed*, the problem can be solved in time linear in n (see also [19] for an improvement on this result). In stark contrast, when the pattern τ is *not fixed* (i.e., when $k = k(n) \to \infty$), permutation pattern matching is known to be NP-complete, as shown by Bose, Buss and Lubiw [8].

A closely related algorithmic question is the *counting version* of permutation pattern matching. The *permutation pattern-counting* problem is the task of counting, given a pattern $\tau \in \mathbb{S}_k$ and permutation $\pi \in \mathbb{S}_n$, the number of occurrences $\#\tau(\pi)$. Once again, there is a straightforward $\mathcal{O}(k \cdot n^k)$-time algorithm – how far is it from optimal? Albert et al. lowered the bound to $\mathcal{O}(n^{2k/3+1})$ [2][2] and the current best known bound is $\mathcal{O}(n^{(1/4+o(1))k})$, due to Berendsohn et al. [7]. Berendsohn et al. also showed a barrier: assuming the exponential time hypothesis, there is no algorithm for pattern-counting with running time $f(k) \cdot n^{o(k/\log k)}$, for *any* function f.

Another intriguing line of work focuses on the pattern-counting problem, for *constant small* k. As the number of patterns $\tau \in \mathbb{S}_k$ is fixed in this regime, one can equivalently, up to a constant multiplicative factor, compute the entire $k!$-dimensional vector of *all* occurrences, $(\#\tau(\pi))_{\tau \in \mathbb{S}_k}$. This vector, which characterises the local structure of a permutation over size-k pointsets, is known as the *k-profile*. The k-profile has also featured in works aiming to understand the local structure of permutations, c.f. [6, 17, 11].

Even-Zohar and Leng [18] designed a class of algorithms with which they compute the 3-profile in $\widetilde{\mathcal{O}}(n)$-time,[3] and the 4-profile in $\widetilde{\mathcal{O}}\left(n^{3/2}\right)$-time. Improving on their result for $k = 4$, Dudek and Gawrychowski [15] gave a bidirectional reduction between the task of computing the 4-profile, and that of counting 4-cycles in a sparse graph. The best known algorithm for the latter problem has running time $\mathcal{O}(n^{2-3/(2\omega+1)})$ [28], where $\omega < 2.372$ [14] is the exponent of matrix multiplication. Consequently, Dudek and Gawrychowski obtain an $\mathcal{O}(n^{1.478})$-time algorithm for the 4-profile.[4]

Our paper continues this line of work: we design algorithms computing the 5, 6 and 7-profiles, and highlight a barrier in the way of computing the 8-profile.

1.1 Our Contribution

We introduce *pattern-trees*: a family of graphs that generalise the corner-trees of Even-Zohar and Leng [18]. Pattern-trees are rooted labeled trees, in which every vertex is associated with a set of *point variables*, along with constraints that fix their relative ordering in the plane, and every edge is labeled by a list of constraints over the ordering of points associated with its incident vertices.

[2] Their algorithm also works for the counting version.

[3] As usual, the notation $\widetilde{\mathcal{O}}(\cdot)$ hides poly-logarithmic factors.

[4] If one assumes the conjectured lower-bounds on the 4-cycle counting problem in sparse graphs, due to [12], then the reduction of Dudek and Gawrychowski [15] also implies a lower-bound on the 4-profile problem – stating that the profile cannot be computed in time $\mathcal{O}(n^{4/3-\varepsilon})$, for any constant $\varepsilon > 0$.

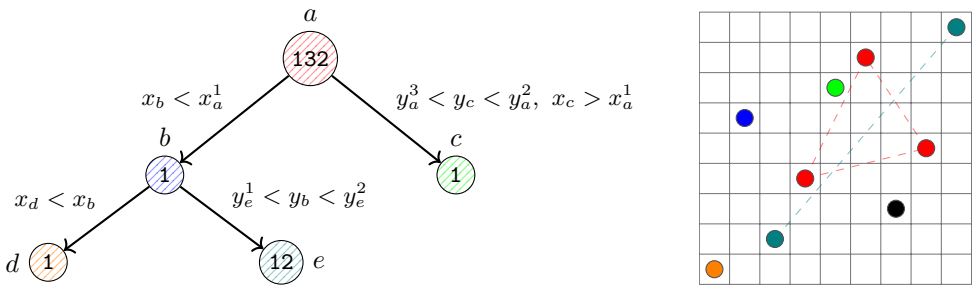


Figure 1 An embedding of a pattern-tree (left) into the permutation `162478359` $\in \mathbb{S}_9$ (right).

Using an algorithm derived from pattern-trees, we obtain our first result.

▶ **Theorem 1.1.** *For every $1 \leq k \leq 7$, the k-profile of an n-element permutation can be computed in $\widetilde{\mathcal{O}}\left(n^2\right)$ time and space.*

Our proof of Theorem 1.1 relies on embeddings of trees into permutations. Consider the *number of distinct embeddings* of the points of a pattern-tree T into the points in the plane associated with a permutation $\pi \in \mathbb{S}_n$, in which the embedding satisfies all constraints defined by the tree (as demonstrated in Figure 1). We show that this quantity can be expressed as a *fixed* integer linear combination of permutation pattern-counts, irrespective of π. Interpreted as a formal sum of patterns, this is simply a vector in $\mathbb{Z}^{\mathbb{S}_{\leq k}}$, where k is the number of point variables in the tree. These vectors are associated with pairwise compositions of linear extensions of partially ordered sets, whose Hasse diagrams can be partitioned in a particular way.

The subspaces spanned by the vectors of trees, over the rationals, are central to our proof. It is not hard to show that when the subspace of a set of trees is full-dimensional, one can derive from those trees an algorithm for the k-profile. To this end, we design an *evaluation algorithm*: given a pattern-tree T and an input permutation $\pi \in \mathbb{S}_n$[5], the algorithm computes the number of occurrences of T in π, denoted $\#T(\pi)$. The complexity of this algorithm depends on properties of the tree. In our proof of Theorem 1.1, we construct a family of trees evaluable in $\widetilde{\mathcal{O}}\left(n^2\right)$-time, which are of full dimension for $\mathbb{S}_{\leq 7}$.

Compared to previous results, Theorem 1.1 offers an improvement whenever $k \in \{5, 6, 7\}$. The best known bound for the k-profile problem is $\mathcal{O}(n^{k/4+o(k)})$, due to Berendsohn et al. [7]. Their approach relies on formulating a binary CSP, and bounding its tree-width. It is well known that binary CSPs can be solved in time $\mathcal{O}(n^{t+1})$ [13, 20], where n is the domain size, and t is the tree-width of the constraint graph. In the algorithm of [7], the tree-width is bounded by $k/4 + o(k)$, where the $o(k)$-term is greater than one. Therefore, their algorithm has at least cubic running time when $k \geq 4$.

The relationship between properties of pattern-trees and the dimensions of the subspaces spanned by them is still far from understood (see Section 5). Corner-trees, which are exactly the pattern-trees whose evaluation is quasi-linear, were shown in [18] to have full rank for $\mathbb{S}_{\leq 3}$, and rank only $|\mathbb{S}_4| - 1 = 23$, restricted to $\mathbb{S}_4$. Intriguingly, we show that the family of pattern-trees with which our proof of Theorem 1.1 is obtained, whose evaluation complexity is quadratic, have full rank for $\mathbb{S}_{\leq 7}$, and rank only $|\mathbb{S}_8| - 1 = 40319$ restricted to $\mathbb{S}_8$. We observe

[5] Throughout this paper we operate on n-element permutations as input. Such inputs are assumed to be presented to the algorithm *sparsely*, e.g., as a length-n vector representing the permutation in one-line notation.

several striking resemblances between the two vectors spanning the orthogonal complements, for $\mathbb{S}_4$ and $\mathbb{S}_8$ respectively, in terms of their symmetries. In fact, we extend a characterisation of [15] regarding the symmetries for $\mathbb{S}_4$ to the case of $\mathbb{S}_8$ (see Section 3.4).

Our second result is a sub-quadratic algorithm for the 5-profile.

▶ **Theorem 1.2.** *The 5-profile of an n-element permutation can be computed in time $\widetilde{\mathcal{O}}\left(n^{7/4}\right)$.*

The proof of Theorem 1.2 is obtained by speeding-up the evaluation algorithm of pattern-trees. The original algorithm for pattern-trees has an integral exponent in its complexity, which is determined by properties of the tree. We show that trees with certain topological properties, i.e., containing a particular set of "gadgets", can be evaluated faster. The family of trees constituting all corner-trees, and their augmentation by our gadgets, span a full-dimensional subspace over $\mathbb{S}_5$. This allows us to break the quadratic barrier for the 5-profile.

One of the key ingredients, both in the original evaluation algorithm and in its extended version, is a data structure known as a multidimensional segment-tree, or rectangle-tree [22, 9].[6] A d-dimensional rectangle-tree holds (possibly weighted) points in $[n]^d$, and answers sum-queries over rectangles $\mathcal{R} \subseteq [n]^d$ (i.e., Cartesian products of segments) in poly-logarithmic time.

The gadgets appearing in the proof of Theorem 1.2 are sub-structures related to the patterns `3214` and `43215`. For the former, we extend an algorithm of [18] into a weighted variant, and provide an evaluation algorithm of complexity $\widetilde{\mathcal{O}}(n^{5/3})$. We then further extend this into an algorithm for the latter gadget, of complexity $\widetilde{\mathcal{O}}(n^{7/4})$. The latter proof is involved, and requires the introduction of a new data structure, which we call a *pair-rectangle-tree*. A pair-rectangle-tree is an extension of rectangle-trees that can facilitate more complex queries, in particular, regarding the dominance counting of a set of points in a rectangle. These queries are also more general than those supported by the 2-dimensional dominance counting data-structures due to [22, 10]. We remark that the original pattern-tree evaluation algorithm can only compute *equivalent* gadgets in quadratic time. That is, the evaluation algorithm is not always optimal.

1.2 Paper Organization

In Section 3 we introduce pattern-trees. Our construction for $3 \leq k \leq 7$ can be found in Section 3.3, and the case $k = 8$ is dealt with in Section 3.4. A straightforward application of pattern-trees for general k is given in Section 3.5. Section 4 revolves around our construction of an $\widetilde{\mathcal{O}}\left(n^{7/4}\right)$-time algorithm for the 5-profile. The augmentation of the pattern-trees evaluation algorithm can be found in Section 4.2, and the particular gadgets used in the 5-profile are obtained in Section 4.3 and Section 4.4. The data structure we introduce for dominance counting in rectangles, pair-rectangle-tree, is given in Section 4.5. Finally, in Section 5 we discuss open questions and possible extensions of this work.

The proofs of Theorem 1.1 and Theorem 1.2 are accompanied by computer-assisted computations. All such computations are available in [5].

[6] A 2-dimensional version of this data structure features in both [18] and [15].

2 Preliminaries

2.1 Permutations

A permutation $\pi \in \mathbb{S}_n$ over n elements is a bijection from $[n]$ to itself, where $[n] := \{1, 2, \ldots, n\}$. Throughout this paper, we express permutations using one-line notation, and if the permutation range is sufficiently small, we omit the parentheses. For instance, 123 is the identity permutation over 3 elements. Associated with any permutation $\pi \in \mathbb{S}_n$ is a set of n points in the plane, $p(\pi) := \{(i, \pi(i)) : i \in [n]\}$, which we refer to as the *points of* π. In the other direction, any set of n points in the plane defines a permutation $\pi \in \mathbb{S}_n$, provided that no two points lie on an axis-parallel line. Given such a set $S \subset \mathbb{R}^2$, we use the notation $S \cong \pi$ to indicate that the points are *order-isomorphic* to π.

An *occurrence* of a *pattern* $\tau \in \mathbb{S}_k$ in a permutation $\pi \in \mathbb{S}_n$ is a k-tuple $1 \leq i_1 < \cdots < i_k \leq n$ such that the set of points $(i_j, \pi(i_j))$ is order-isomorphic to τ. That is, $\pi(i_j) < \pi(i_l)$ if and only if $\tau(j) < \tau(l)$ for all $j, l \in [k]$. The number of occurrences of τ in π is denoted by $\#\tau(\pi)$.

The dihedral group D_k is the symmetry group of the regular k-gon. The group D_4 naturally acts on the symmetric group $\mathbb{S}_n$, by acting on $[1, n]^2$. Formally, for any element $g \in D_4$ and permutation $\pi \in \mathbb{S}_n$, we have that $(g.\pi) \in \mathbb{S}_n$ is the permutation for which $g.(p(\pi)) \cong g.\pi$. Our algorithms usually receive permutations as input, and compute some combination of pattern-counts. To this end, it is sometimes helpful to first act on the input with an element $g \in D_4$ (as a preprocessing step), and only then invoke the algorithm as usual. In this way, if an algorithm computes the count $\#\tau(\pi)$, then after the action we obtain $\#\tau(g.\pi) = \#(g^{-1}.\tau)(\pi)$.

Our main focus in this paper is the computation of $\#\tau(\pi)$ for all $\tau \in \mathbb{S}_k$, where $\pi \in \mathbb{S}_n$ is given as input and k is fixed. This collection of counts is defined as follows.

▶ **Definition 2.1.** *The* k-profile *of a permutation* $\pi \in \mathbb{S}_n$ *is the vector* $(\#\tau(\pi))_{\tau \in \mathbb{S}_k} \in \mathbb{Z}^{\mathbb{S}_k}$.

2.2 Partially Ordered Sets

A partially ordered set (*poset*) $\mathcal{P}(X, \leq)$ over a ground set X is a partial arrangement of the elements in X according to the order relation $\leq$. If $\leq$ is not reflexive, we say that $\mathcal{P}$ is *strict*. A partial order $\leq^\star$ is said to be an extension of $\leq$ if $x \leq y$ implies $x \leq^\star y$ for all $x, y \in X$. If an extension $\leq^\star$ is a total order, it is called a *linear extension* of $\leq$. As usual, the set of all linear extensions of a poset $\mathcal{P}$ is denoted by $\mathcal{L}(\mathcal{P})$.

2.3 Computational Model

Throughout this paper we disregard all polylog(n)-factors, so our results hold for any choice of standard computational model (say, word-RAM). The notation $\widetilde{\mathcal{O}}\left(n^k\right)$ (adding the tilde) is used to hide poly-logarithmic factors. The algorithms presented in this paper operate on n-element permutations as input, and we remark that such inputs are assumed to be presented to the algorithm *sparsely*, e.g., as a length-n vector representing the permutation in one-line notation.

2.4 Rectangle-Trees

Our algorithms for efficiently computing profiles rely heavily on a simple and powerful data structure, which we refer to as a *rectangle-tree*[7] or a *multidimensional segment-tree*. Concretely, we require the following folklore fact.

▶ **Proposition 2.2** ([9, 22], see also [15]). *For any fixed dimension $d \geq 1$, there exists a deterministic data structure $\mathcal{T}$ that supports each of the following actions in $\widetilde{\mathcal{O}}(1)$ time:*

1. *Initialisation: Given $n \in \mathbb{N}$, construct an empty tree over $[n]^d$.*
2. *Insertion: Given $x \in [n]^d$ and $w = \mathcal{O}(\text{poly}(n))$, add weight w to point x.*
3. *Query: Given a rectangle $\mathcal{R} \subseteq [n]^d$, the query $\mathcal{T}(\mathcal{R})$ returns the sum of weights over all points in $\mathcal{R}$.*

One illustration of the application of rectangle-trees to pattern-counting is given by the simple (and again, folklore) case of *monotone pattern-counting*.

▶ **Proposition 2.3.** *Let $k \geq 1$ be a fixed integer and let $\pi \in \mathbb{S}_n$ be an input permutation. The pattern-counts $\#(\mathtt{1}, \ldots, \mathtt{k})(\pi)$ and $\#(\mathtt{k}, \ldots, \mathtt{1})(\pi)$ can be computed in $\widetilde{\mathcal{O}}(n)$ time.*

Proof. See the full version [4]. ◀

▶ Remark 2.4. It is also possible to count monotone patterns using 1-dimensional segment-trees, somewhat more efficiently. However, the difference is only in logarithmic factors. The multidimensional structure highlighted above will serve us in more complicated cases.

3 Pattern-Trees

In this section we introduce a family of graphs, called *pattern-trees*. Using pattern-trees we derive algorithms for computing the k-profile of a permutation. Our main result for this section (see Section 3.3) is a quadratic-time algorithm for the k-profile of a permutation, for every $k \leq 7$:

▶ **Theorem 1.1.** *For every $1 \leq k \leq 7$, the k-profile of an n-element permutation can be computed in $\widetilde{\mathcal{O}}(n^2)$ time and space.*

In Section 3.4 we consider the subspaces spanned by the same family of pattern-trees, restricted to $\mathbb{S}_8$. We show that this subspace is of dimension $|\mathbb{S}_8| - 1$, one less than required. In Section 3.5 we consider the case of general (constant) k, and show a straightforward application of pattern-trees yielding an $\widetilde{\mathcal{O}}(n^{\lceil k/2 \rceil})$-time algorithm for the k-profile.

Before we present pattern-trees, let us begin by recalling corner-trees.

3.1 Warmup: Corner-Trees

One of the main components in the work of [18] is the introduction of *corner-trees*. Corner-trees are a family of rooted edge-labeled trees. Every corner-tree of k vertices is associated with a particular vector in $\mathbb{Z}^{\mathbb{S}_{\leq k}}$; i.e., a formal integer linear combination of permutations, each of size at most k. Furthermore, there exists an efficient evaluation algorithm for corner-trees: given any input permutation $\pi \in \mathbb{S}_n$ and corner-tree T, the integer sum of permutation *pattern-counts* in π, called the vector of T, can be computed in time $\widetilde{\mathcal{O}}(n)$. We refer to this operation as *evaluating the vector of T over π*.

[7] A rectangle $\mathcal{R} \subseteq [n]^d$ is a Cartesian product of *segments*, i.e., invervals of the form $\{a, a+1, \ldots, b\} \subseteq [n]$.

▶ **Definition 3.1** (corner-tree [18]). *A corner-tree*[8] *is a rooted*[9] *edge-labeled tree, with edge labels in the set* $\{\mathrm{NE}, \mathrm{NW}, \mathrm{SE}, \mathrm{SW}\}$.

An *occurrence* of a corner-tree T in a permutation π is a map $\varphi : V(T) \to p(\pi)$, in which the image *agrees* with the edge-labels of the tree. That is, for every edge $(u \to v) \in E(T)$, $\varphi(v)$ is to the *left* of $\varphi(u)$ if the edge is labeled NW or SW, and to its right otherwise. Similar rules apply for their vertical ordering. As in [18], the *number of occurrences* of a corner-tree T in a permutation π is denoted by $\#T(\pi)$.

The *vector* of a corner-tree is a formal sum of permutation patterns with integer coefficients, representing the number of occurrences of the tree in *any* input permutation. For instance, the vector of $\circ \xrightarrow{\mathrm{SE}} \circ \xrightarrow{\mathrm{NE}} \circ$ is $\#213 + \#312$. Clearly, the vector of a corner-tree over k vertices may involve patterns of size at most k, as the tree conditions on the relative ordering of at most $|V(T)|$ points (smaller patterns may appear as well, since occurrences are not necessarily injective).

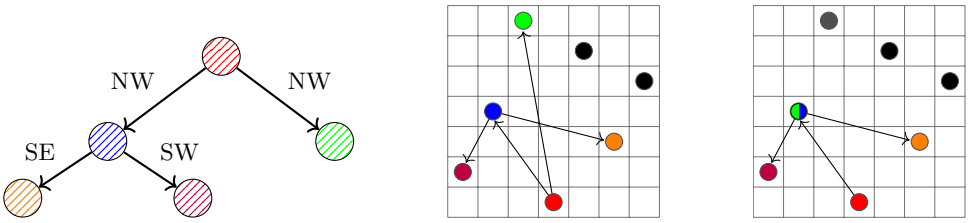


Figure 2 Two occurrences of a corner-tree (left) in $\pi = 2471635 \in \mathbb{S}_7$ (centre, right). Occurrences need not be injective; for instance, on the right, the blue and green points are identified.

Theorem 1.1 of [18] presents an algorithm for evaluating the vector of a corner-tree over an input permutation $\pi \in \mathbb{S}_n$. For expositionary purposes, under Section 3.1 we sketch a simplified version of their algorithm, phrased in terms of rectangle-trees.

▶ **Proposition 3.2** (Theorem 1.1 of [18]). *The vector of any corner-tree with a constant number of vertices can be evaluated over an input permutation $\pi \in \mathbb{S}_n$ in time $\widetilde{\mathcal{O}}(n)$.*

Proof Sketch. Let T be a corner-tree and let $\pi \in \mathbb{S}_n$ be a permutation. To start, construct a 2-dimensional rectangle-tree (see Section 2.4), and insert the points $p(\pi)$ with weight 1, in time $\widetilde{\mathcal{O}}(n)$. Associate this tree with the *leaves* of T. Next, traverse the vertices of T in post-order. At every internal vertex u, construct a new (empty) rectangle-tree $\mathcal{T}_u$, and associate it with u. Then, iterate over every point in π, and at each point perform one *rectangle query* to the rectangle-tree associated with each of u's children, querying the rectangle corresponding to the edge label in T written on the parent-child edge. For example, if $u \to v$ is labeled SW, the iteration over a point $(i, \pi(i)) \in p(\pi)$ queries the rectangle $[1, i-1] \times [1, \pi(i)-1]$. Store the product of all answers to these queries in $\mathcal{T}_u$, at the position of the current permutation point. It can be shown that the sum of all values at the root's tree (i.e., a full rectangle query) is the number of occurrences, $\#T(\pi)$. ◀

3.2 Pattern-Trees

We introduce *pattern-trees*: a family of graphs that generalise the corner-trees of [18]. In pattern-trees, every *vertex* is labeled by a permutation, and every *edge* is labeled by a list of constraints. The permutations written on the vertices fix the exact ordering of the

[8] For convenience, we consider corner-trees to be edge-labeled, rather than vertex-labeled as in [18].
[9] Hereafter, whenever we consider rooted trees, we orient their edges away from the root.

points corresponding to them, and the edge-constraints are similarly imposed over the points corresponding to the two incident vertices. As with corner-trees, pattern-trees serve two purposes: firstly, every pattern-tree is associated with a set of constraints over permutation points, the number of satisfying assignments to which can be expressed as a formal integer linear combination of patterns (that is, a vector). Secondly, we present an algorithm for evaluating this vector over an input permutation. This allows us to efficiently compute certain pattern combinations not spanned by corner-trees.

▶ **Definition 3.3** (pattern-tree). *A pattern-tree T is a rooted edge- and vertex-labeled tree, where:*

1. *Every vertex $v \in V(T)$ is:*
   - *Labeled by a permutation $\tau_v \in \mathbb{S}_r$, for some integer $r \geq 1$.*
   - *Associated with two sets of fresh variables,*

     $$x_v := \{x_v^1, \ldots, x_v^r\}, \text{ and } y_v := \{y_v^1, \ldots, y_v^r\},$$

     where we denote $p_v^i := (x_v^i, y_v^i)$ for every $i \in [r]$, and $p_v := \{p_v^i : i \in [r]\}$.
2. *Every edge $(u \to v) \in E(T)$ is labeled by:*
   - *Two strict posets, $\mathcal{P}_{uv}^x = (x_u \sqcup x_v, <)$ and $\mathcal{P}_{uv}^y = (y_u \sqcup y_v, <)$.*
   - *A set $E_{uv} \subseteq p_u \times p_v$ of equalities between the points of u and those of v.*

The *size* $s(v)$ of a vertex v is the size r of the permutation $\tau_v \in \mathbb{S}_r$ with which it is labeled. The *maximum size* of a pattern-tree, denoted $s(T)$, is the maximum over all vertex sizes. The *total size*, denoted $\Sigma(T)$, is the sum over all vertex sizes. Under this notation, a corner-tree is a pattern-tree of maximum size one. Lastly, $p(T) := \bigsqcup_{v \in V(T)} p_v$ is the set of all $\Sigma(T)$ points in the tree.

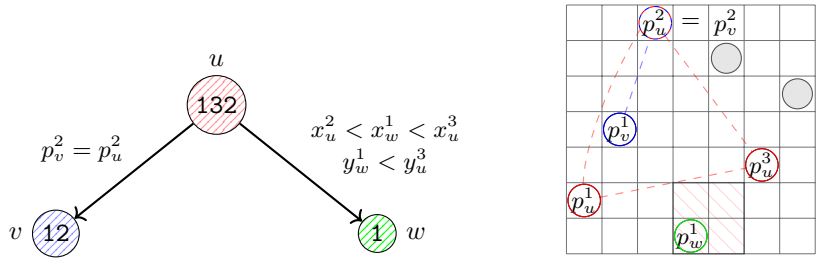


Figure 3 An occurrence of a pattern-tree T (left) in the permutation $\pi = \texttt{2471635} \in \mathbb{S}_7$ (right). Every set of coloured points on the right induces the permutation with which the similarly coloured vertex on the left is labeled ("vertex constraints"). All of the edge-constraints are also satisfied: points p_v^2 and p_u^2 are *identified*, and point p_w^1 (green) must reside within the red shaded square. This tree corresponds to a linear combination, $\#\texttt{1423} + \#\texttt{2413} + 2 \cdot \#\texttt{12534} + \cdots + \#\texttt{24513}$, of patterns in $\mathbb{S}_4$ and $\mathbb{S}_5$. The tree has total size $\Sigma(T) = 6$ and maximum size $s(T) = 3$.

Pattern-Tree Constraints. Any pattern-tree T defines constraints $\mathcal{C}(T)$ over *points* $p(T)$:

1. Every vertex v labeled by $\tau_v \in \mathbb{S}_r$ contributes the following inequalities,[10]

   $$x_v^1 < x_v^2 < \cdots < x_v^r, \text{ and } y_v^i < y_v^j \text{ for all } i, j \in [r] \text{ such that } \tau_v(i) < \tau_v(j).$$

2. Every edge $u \to v$ contributes the inequalities in $\mathcal{P}_{uv}^x$ and $\mathcal{P}_{uv}^y$, and the equalities in E_{uv}.

[10] These vertex-constraints enforce the pattern τ_v over the points p_v.

Hereafter, we partition $\mathcal{C}(T)$ into two parts: its *equalities*, which define an equivalence relation $E^T := \bigsqcup_{u \to v} E_{uv}$ over the points $p(T)$, and its *inequalities*, which define *strict* posets,

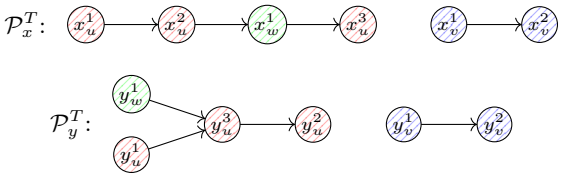
$$\mathcal{P}_x^T = \Big(\bigsqcup_{v \in V(T)} x_v, < \Big), \text{ and } \mathcal{P}_y^T = \Big(\bigsqcup_{v \in V(T)} y_v, < \Big).$$

Given an equivalence relation $E \supseteq E^T$, the posets $\mathcal{P}_x^E$ and $\mathcal{P}_y^E$ are the strict posets obtained from $\mathcal{P}_x^T$ and $\mathcal{P}_y^T$ by replacing every coordinate variable corresponding to a point $p \in p(T)$ by a single variable corresponding to the equivalence class of p in E.

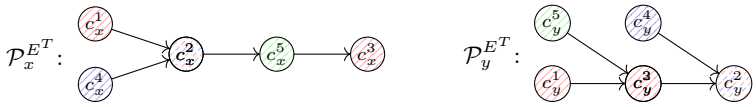
▶ **Example 3.4.** The pattern-tree T appearing in Figure 3 corresponds to the constraints

$$\mathcal{C}(T) = \{x_u^1 < x_u^2 < x_u^3,\ y_u^1 < y_u^3 < y_u^2,\ x_u^2 < x_w^1 < x_u^3,\ y_w^1 < y_u^3,\ p_v^2 = p_u^2,\ x_v^1 < x_v^2,\ y_v^1 < y_v^2\}$$

whose posets are:



Applying $E^T = \{c_1 = \{p_u^1\},\ c_2 = \{p_v^2, p_u^2\},\ c_3 = \{p_u^3\},\ c_4 = \{p_v^1\},\ c_5 = \{p_w^1\}\}$ yields the posets:



Pattern-Tree Occurrences. As with corner-trees, we define *pattern-tree occurrences*.

▶ **Definition 3.5.** *An occurrence $\varphi : p(T) \to p(\pi)$ of a pattern-tree T in a permutation $\pi \in \mathbb{S}_n$ is a map whose image $\varphi(p(T))$ conforms to the constraints $\mathcal{C}(T)$.*

An illustration of a pattern-tree occurrence is shown in Figure 3. Note that, as with corner-trees, occurrence maps need not be injective. We remark that some pattern-trees may have no occurrences, in any permutation $\pi \in \mathbb{S}_n$. For example, $u \circ\!\!\xrightarrow[p_u = p_v,\, x_u < x_v]{}\!\!\circ\, v$ is infeasible.

Pattern-Tree Vectors. As with corner-trees, one can associate a vector with every pattern-tree T, which is a formal integer linear combination of pattern-counts representing the number of occurrences of T in any input permutation $\pi \in \mathbb{S}_n$.

▶ **Lemma 3.6.** *Let T be a pattern-tree. The number of occurrences of T in an input permutation $\pi \in \mathbb{S}_n$ is given by the following sum of pattern-counts, each of size at most $\Sigma(T)$:*

$$\#T(\pi) = \sum_{E \supseteq E^T} \sum_{\substack{\sigma \in \mathcal{L}(\mathcal{P}_x^E) \\ \tau \in \mathcal{L}(\mathcal{P}_y^E)}} \#\left(\tau\sigma^{-1}\right)(\pi).$$

Proof. See the full version [4]. ◀

Evaluating a Pattern-Tree. It remains to construct an evaluation algorithm for the vector of a pattern-tree. To present our algorithm, we require some notation.

1. Points: To every set of points $S := \{s_1, \ldots, s_r\} \subseteq p(\pi)$, where $\pi \in \mathbb{S}_n$ is a permutation and $(s_1)_x < \cdots < (s_r)_x$, we associate a $2r$-dimensional point,

$$p(S) := \big((s_1)_x, \ldots, (s_r)_x, (s_1)_y, \ldots, (s_r)_y\big) \in [n]^{2r}$$

2. Rectangles: To every combination of an edge $(u \to v) \in E(T)$ in a pattern-tree T, where u and v are of sizes r and d respectively, and set of points $S := \{s_1, \ldots, s_r\} \subseteq p(\pi)$, we associate a $2d$-dimensional rectangle,

$$\mathcal{R}^S_{uv} := \mathcal{R}^{S,x}_{uv} \times \mathcal{R}^{S,y}_{uv} \subseteq [n]^{2d}, \text{ where } \mathcal{R}^{S,x}_{uv}, \mathcal{R}^{S,y}_{uv} \subseteq [n]^d.$$

The i-th segment of $\mathcal{R}^{S,x}_{uv}$ contains the x coordinates that x^i_v can take under the constraints of $u \to v$, when x^j_u is assigned $(s_j)_x$. Namely, the intersection of the following segments:

$$\underbrace{\bigcap_{j:(p^j_u, p^i_v) \in E_{uv}} \{(s_j)_x\}}_{\text{equals}}, \quad \underbrace{\bigcap_{j:(x^i_v < x^j_u) \in \mathcal{P}^x_{uv}} \{1, \ldots, (s_j)_x - 1\}}_{\text{less-than}}, \quad \underbrace{\bigcap_{j:(x^i_v > x^j_u) \in \mathcal{P}^x_{uv}} \{(s_j)_x + 1, \ldots, n\}}_{\text{greater-than}}$$

The y-segments are similarly defined.

Observe that the rectangle $\mathcal{R}^S_{uv}$ is the set of permissible locations for the points p_v, subject to the edge-constraints on the edge $u \to v$, when the points p_u are mapped to $p(S)$. That is, it enforces both the equalities (left) and inequalities (centre, right) written on the edge $u \to v$.

The evaluation algorithm now follows.

Algorithm 1 Bottom-Up Evaluation of Pattern-Tree Vector.

Input: A pattern-tree T, and a permutation $\pi \in \mathbb{S}_n$.

1. Traverse the vertices of T in post-order. For every vertex u labeled by $\tau_u \in \mathbb{S}_r$:
   a. Construct a new (empty) rectangle-tree $\mathcal{T}_u$ of dimension $2r$.
   b. Iterate over all sets $S := \{s_1, \ldots, s_r\} \subseteq p(\pi)$. If $S \cong \tau_u$, then:
      i. For every child v of u, issue the query $\mathcal{T}_v(\mathcal{R}^S_{uv})$.
      ii. Add the weight $\prod_{v:\ u \to v} \mathcal{T}_v(\mathcal{R}^S_{uv})$ (or 1, if u is a leaf) to point $p(S)$ in $\mathcal{T}_u$.
2. Return the answer to the query $\mathcal{T}_z(\mathcal{R})$, where z is the root of T, and $\mathcal{R} = [n]^{2|\tau_z|}$.

▶ **Theorem 3.7.** *Let T be a pattern-tree of constant total size, and let $\pi \in \mathbb{S}_n$ be a permutation. The vector of T can be evaluated over π in $\widetilde{\mathcal{O}}\left(n^{s(T)}\right)$ time, where $s(T)$ is the maximum size of T.*[11]

Proof. The running time of Algorithm 1 is $\widetilde{\mathcal{O}}\left(n^{s(T)}\right)$, since every operation takes $\widetilde{\mathcal{O}}\left(1\right)$ time (recall that $\Sigma(T) = \mathcal{O}(1)$), except step (1b.), which we perform in time $\mathcal{O}(n^r)$, by trivial enumeration. It remains to prove its correctness. We do so, by induction on the height of the tree.

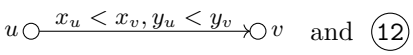
[11] The space-complexity is also $\widetilde{\mathcal{O}}\left(n^{s(T)}\right)$, since at every vertex of size r, we insert $\leq \binom{n}{r}$ points to a rectangle-tree.

Let $u \in V(T)$ be a vertex, let $\tau_u \in \mathbb{S}_r$ be its permutation label, and let $T_{\leq u}$ be the sub-tree rooted at u. We claim that for every $S := \{s_1, \ldots, s_r\} \subseteq p(\pi)$, the weight of $p(S)$ in $\mathcal{T}_u$ is the number of occurrences $\varphi : p(T_{\leq u}) \to p(\pi)$ in which the points $p(u)$ are mapped to S. That is, for every $1 \leq i \leq r$, it holds that $\varphi(p_u^i) = s_i$.

In the base-case, u is a leaf, and Algorithm 1 simply enumerates over all sets S of cardinality r, adding weight 1 whenever $S \cong \tau_u$. So the claim holds. For the inductive step, let u be an internal vertex. For every child v of u, by the induction hypothesis, the query $\mathcal{T}_v(\mathcal{R}_{uv}^S)$ counts the number of occurrences $\varphi_v : p(T_{\leq v}) \to p(\pi)$ in which there exists a point $A = (a_1, a_2, \ldots) \in \mathcal{R}_{uv}^S$ such that $\varphi_v(p_u^i) = a_i$, for every i. That is, the number of occurrences of the tree in which we add the u as the root to the tree $T_{\leq v}$, where the occurrence maps p_u^i to s_i for every $i \in [r]$. These occurrences are *independent* for every child v of u, therefore picking any combination of them yields a new occurrence of $T_{\leq u}$ in π, the total number of which is indeed the product $\prod_{v:\ u \to v} \mathcal{T}_v(\mathcal{R}_{uv}^S)$.

The proof now follows, as in the rectangle-tree $\mathcal{T}_z$ corresponding to the root z of T, every point S has weight which is the number of occurrences of T in π in which p_z is mapped to S. Therefore, the sum of all points in $\mathcal{T}_z$ yields the total number of occurrences. ◀

▶ Remark 3.8. The algorithm presented in Theorem 3.7 is not necessarily the most efficient way to compute the vector of a pattern-tree, for several reasons. Firstly, many trees may correspond to the same vector, and these trees need not have the same maximum size. For example, both



correspond to the vector #12. Secondly, as we will see in Section 4, there exist vectors for which bespoke *efficient* algorithms can be constructed, whose running time is strictly smaller than the maximum size of *any* pattern-tree with the same vector.

3.3 $\widetilde{\mathcal{O}}(n^2)$ Algorithm for the k-Profile, for $1 \leq k \leq 7$

The corner-trees of [18] are very efficiently computable. However, asymptotically, there are quite few of them: the number of rooted unlabeled trees over k vertices is only exponential in k (see, e.g., [23] for a more accurate estimate), and clearly so is the number of corner-tree edge labels. Therefore, as $k \to \infty$, even if asymptotically almost all corner-trees vectors were linearly independent over $\mathbb{S}_{\leq k}$, they would nevertheless contribute only a *negligible* proportion with respect to the full dimension, $|\mathbb{S}_{\leq k}| = \sum_{r=1}^{k} r!$.

In contrast, it is not hard to see that pattern-trees are *fully expressive*: for every pattern $\tau \in \mathbb{S}_k$, there exists a pattern-tree T with $s(T) = k$, whose vector is precisely that pattern (in fact, $s(T) = \lceil k/2 \rceil$ suffices, see Section 3.5). To design *efficient* algorithms for the k-profile, we are interested in finding families of pattern-trees of *least maximum size*, whose corresponding vectors are linearly independent.

In [18], corner-trees (i.e., pattern-tree of maximum size 1) over k vertices were shown to have full rank over $\mathbb{Q}^{\mathbb{S}_{\leq k}}$ for $k = 3$, and in the cases $k = 4$ and $k = 5$, the subspaces spanned by them, restricted to $\mathbb{S}_4$ and $\mathbb{S}_5$, were found to be of dimensions only 23 and 100, respectively. Here, we show that for $k \leq 7$, pattern-trees of maximum size ≤ 2 suffice.

Proof of Theorem 1.1. Let $\mathbb{S} := \bigsqcup_{k=1}^{7} \mathbb{S}_k$. By enumeration (see Section A), there exists a family of $\sum_{k=1}^{7} k! = 5913$ pattern-trees of maximum size at most 2 and total size at most 7, whose vectors are linearly independent over $\mathbb{Q}^{\mathbb{S}}$. Let $A \in \mathbb{Q}^{\mathbb{S} \times \mathbb{S}}$ be the matrix whose rows are these vectors, and let $A^{-1} \in \mathbb{Q}^{\mathbb{S} \times \mathbb{S}}$ be its inverse. A may be computed ahead of time, as can

its inverse, for example using Bareiss' algorithm [3]. Using Theorem 3.7, evaluate every row of A over π in time $\widetilde{\mathcal{O}}\left(n^2\right)$. This yields a vector $v \in \mathbb{Z}^{\mathbb{S}}$, and the k-profiles of π, for $k \leq 7$, are obtained by computing $A^{-1}v$. ◀

This approach is not sufficient to compute the 8-profile in $\widetilde{\mathcal{O}}\left(n^2\right)$ time, as discussed in Section 3.4. See Section 3.5 for an application of pattern-trees to general fixed k.

3.4 The Case $k = 8$

Do pattern-trees over at most 8 points, and with $s(T) \leq 2$, have full dimension for $\mathbb{S}_{\leq 8}$? Using a computer program, we exhaustively enumerate all pattern-trees with the following properties,[12]

1. Every tree has $|p(T)| = 8$ points, and maximum size $s(T) \leq 2$.
2. No edge is labeled with an equality.

In [18] it was shown that pattern-trees with 4 vertices and maximum size 1 (corner-trees) span a subspace of dimension only $|\mathbb{S}_4| - 1 = 23$, when restricted to $\mathbb{S}_4$. Our pattern-trees extend this result: the subspace spanned by the above family of pattern-trees, with 8 points and maximum size ≤ 2, is of dimension exactly $|\mathbb{S}_8| - 1 = 40319$, when restricted to $\mathbb{S}_8$. The two vectors spanning the orthogonal complements of the subspaces for $\mathbb{S}_4$ and $\mathbb{S}_8$, $v_4 \in \mathbb{Q}^{\mathbb{S}_4}$ and $v_8 \in \mathbb{Q}^{\mathbb{S}_8}$ respectively, bear striking resemblance. See the full version [4] for the details.

3.5 $\widetilde{\mathcal{O}}\left(n^{\lceil k/2 \rceil}\right)$ Algorithm for the k-Profile

We end this section by considering the problem of computing the k-profile via pattern-trees, for arbitrary (fixed) k. In the following proposition, we show that families of pattern-trees of maximal size $s(T) = \lceil k/2 \rceil$ suffice for computing the k-profile, through Algorithm 1. See Section 5 for a discussion on the relationship between $s(T)$ and k.

▶ **Proposition 3.9.** *Let $\pi \in \mathbb{S}_n$ be an input permutation, and let $k \geq 2$ be a fixed integer. The k-profile of π can be computed in $\widetilde{\mathcal{O}}\left(n^{\lceil k/2 \rceil}\right)$ time.*

Proof. See the full version [4]. ◀

4 $\widetilde{\mathcal{O}}\left(n^{7/4}\right)$ Algorithm for the 5-Profile

In Section 3, we recalled that pattern-trees of maximum size 1 (i.e., corner-trees) have full rational rank for $\mathbb{S}_{\leq 3}$ [18], and proved that trees of maximal size at most 2 have full rank for $\mathbb{S}_{\leq 7}$ (see Theorem 1.1). Therefore, up to $k = 3$, the k-profile of an n-element permutation can be computed in $\widetilde{\mathcal{O}}\left(n\right)$ time, and up to $k = 7$, it is computable in $\widetilde{\mathcal{O}}\left(n^2\right)$ time. This naturally raises the question: is there a sub-quadratic time algorithm for these cases, where $k \geq 4$? We prove the following.

▶ **Theorem 1.2.** *The 5-profile of an n-element permutation can be computed in time $\widetilde{\mathcal{O}}\left(n^{7/4}\right)$.*

We remark that the case $k = 4$ has been extensively studied in [15] and [18]. There, they construct sub-quadratic algorithms of complexities $\mathcal{O}\left(n^{1.478}\right)$ and $\widetilde{\mathcal{O}}\left(n^{3/2}\right)$, respectively.

[12] See Section A for a description of the enumeration process. For $k = 8$, this yields a matrix with $|\mathbb{S}_8| = 8!$ columns, and $|\mathbb{S}_8 \times \mathbb{S}_8 \times \{T_\lambda\}| \approx 2^{37}$ rows. We remark that we explicitly do not consider pattern-trees over *more* than 8 points, and trees whose edges are labeled by equalities. Whether this is without loss of generality, i.e., could their inclusion increase the rank, is unknown to us.

4.1 Marked and Weighted Patterns

For the proof of Theorem 1.2, we introduce the following notation.

Marked Patterns. A *marked pattern* is a pattern $\tau \in \mathbb{S}_k$ associated with an index $1 \leq j \leq k$. We say that a marked pattern τ occurs at index $1 \leq i \leq n$ in $\pi \in \mathbb{S}_n$, if there exists an occurrence of τ in π, in which the j-th x-coordinate is i. When the marked pattern τ is short, we underline the j-th index to indicate that marked index. For instance, $\underline{2}1$ occurs in 132 at index 2.

The *marked pattern-count* is a 2-dimensional rectangle-tree containing the points $p(\pi)$, in which the weight of every point $(i, \pi(i))$ is the number of marked pattern occurrences at position i. For example, the tree $\mathcal{T}_2$ appearing in Proposition 2.3 is precisely the marked pattern-count $\#1\underline{2}(\pi)$.

Weighted Pattern-Counts. Let $\pi \in \mathbb{S}_n$ and let $w_1, \ldots, w_k : [n] \to \mathbb{Z}$ be *weight functions*, where $k \geq 1$ is a fixed integer. The *weighted pattern-count* of $\tau \in \mathbb{S}_k$ in π, denoted $\#_w\tau(\pi)$, is the sum of $\prod_{j=1}^{k} w_j(i_j)$ over all occurrences $1 \leq i_1 < \cdots < i_k \leq n$ of τ in π. In other words, we count occurrences where every point has weight depending on its position, rather than 1 as usual.

The two concepts of marked patterns and weighted patterns can be combined in a straightforward way: the *weighted marked pattern-count* is once again defined as a 2-dimensional rectangle-tree, as with marked pattern-counts, but where now the number of occurrences for each point $(i, \pi(i))$ is appropriately weighted.

4.2 An Improvement to the Bottom-Up Algorithm

Recall that Algorithm 1 has time complexity $\widetilde{\mathcal{O}}\left(n^{s(T)}\right)$, where $s(T)$ is an integer. As we seek sub-quadratic algorithms, and since trees of $s(T) = 1$ (i.e., corner-trees) do not have full rank for $\mathbb{S}_{\leq 5}$, we take an alternative approach. We extend the family of pattern-trees of maximum size 1 by allowing them to contain "gadgets" with specific patterns of sizes 4 and 5, defined below. To evaluate such trees efficiently, we show a variation of Algorithm 1 that handles these gadgets as special cases.

Let u be vertex of a pattern-tree T, labeled by some permutation $\tau_u \in \mathbb{S}_r$, such that (see Figure 4):

1. The incoming edge to u (if any) conditions on a single point of u, say p_u^l.
2. Each outgoing edge of u (if any) is labeled by a single equality to a point of u.

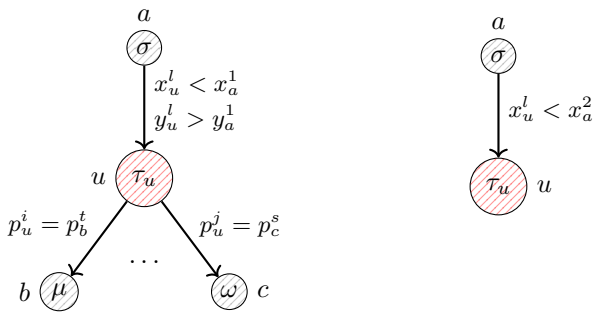


Figure 4 Two "gadgets" in a pattern-tree. The left corresponds to a weighted marked pattern-count of τ_u, marked at l. The right corresponds to a (unweighted) marked pattern-count of τ_u.

Suppose that, for the permutation $\tau_u \in \mathbb{S}_r$ and index $l \in [r]$, and given a set of weight functions $\{w_j\}_j$,[13] we are able to construct a 2-dimensional rectangle-tree representing the *weighted marked pattern-count*, $\#_w\tau_u(\pi)$ marked at l. Then, we claim that one can *modify* Algorithm 1 by replacing the rectangle-tree $\mathcal{T}_u$ associated with u, with the weighted marked pattern-count of τ, marked at l, for a particular choice of weight functions. Concretely, we make the following modifications in Algorithm 1:

Traversing u. Instead of the routine operation of Algorithm 1, when u is visited we compute a weighted l-marked pattern-count $\#_w\tau_u(\pi)$, abbreviated as $\mathcal{T}'_u$, with the following weights: for every point p^j_u, define a weight function $w_j : [n] \to \mathbb{Z}$ by

$$w_j(a) := \prod_{\substack{v:\ u\to v \\ p^j_u \text{ constrained}}} \mathcal{T}_v(\mathcal{R}^i_v)$$

where for an edge $u \to v$ labeled $p^j_u = p^i_v$, we define $\mathcal{R}^i_v$ as the rectangle in which the i-th x-segment is $\{a\}$ and all other segments are unconstrained (if p^j_u is not constrained by any outgoing edges, set its weight function to 1). By the invariant of Algorithm 1, the query $\mathcal{T}_v(\mathcal{R}^i_v)$ counts the number of occurrences of $T_{\leq v}$ in π such that $x^i_v = a$. Therefore, the resulting tree $\mathcal{T}'_u$ contains, at every point $(i, \pi(i))$, the number of occurrences of $T_{\leq u}$ in π such that $x^l_u = i$.

Querying u. In Algorithm 1 we query the rectangle-trees of vertices in two scenarios:

1. If u is an internal vertex: In the original formulation of Algorithm 1, when the parent z of u is visited, we issue queries of the form $\mathcal{T}_u(\mathcal{R}^S_{zu})$, for pointsets $S \subseteq p(\pi)$. As the edge $z \to u$ only constrains p^l_u, the rectangles $\mathcal{R}^S_{zu}$ are *degenerate*, i.e., all of their segments are complete, except the two segments corresponding to p^l_u. These queries can be answered by $\mathcal{T}'_u(\mathcal{R}^l_u)$, where $\mathcal{R}^l_u \subseteq [n]^2$ is the 2-dimensional projection of $\mathcal{R}^S_{zu}$ onto those two segments.
2. If u is the root: The final step of the algorithm performs the full rectangle query $\mathcal{T}_u([n]^{2|\tau_u|})$, which counts the occurrences of $T_{\leq u} = T$ in all of π. This can be answered by the full rectangle query $\mathcal{T}'_u([n]^2)$.

As for the correctness of this modification to Algorithm 1, it remains to show that the new queries return the same values as the original ones. Let $\mathcal{R}$ be some rectangle query to $\mathcal{T}_u$. The value of $\mathcal{T}_u(\mathcal{R})$ is the number of occurrences of $T_{\leq u}$ in π, constrained to the coordinates allowed by $\mathcal{R}$. Since in both cases, all segments in $\mathcal{R}$ are complete except possibly those corresponding to p^l_u, this counts the occurrences of $T_{\leq u}$ in π constrained only to $p^l_u \in \mathcal{R}'$, for a 2-dimensional projection $\mathcal{R}'$ of $\mathcal{R}$ to the corresponding segments. By definition of a weighted marked pattern-count, this is exactly the value of $\mathcal{T}'_u(\mathcal{R}')$.

In the remainder of this section, we design algorithms computing the pattern-counts $\#_w\mathtt{3214}$ and $\#\mathtt{43215}$ in sub-quadratic time[14]. Consequently, we can insert vertices labeled `3214` and `43215` into pattern-trees of maximum size 1 and with at most 5 points, with the aforementioned edge constraints (see τ_u in Figure 4). Using the above modification to Algorithm 1, the overall time complexity for the evaluation of such trees remains sub-quadratic.

[13] We assume that for every weight function $w_j : [n] \to \mathbb{Z}$, the value $w_j(a)$ can be computed in $\widetilde{\mathcal{O}}(1)$-time.
[14] The algorithm can be extended to the weighted count $\#_w\mathtt{43215}$, but the unweighted count is sufficient for our purposes.

4.3 Computing $\#_w 3214$ in $\widetilde{\mathcal{O}}\left(n^{5/3}\right)$ time

Theorem 1.2 of [18] describes an algorithm for counting #3214. We require a slight alteration of their algorithm, and in particular, a weighted variant.

▶ **Lemma 4.1** (weighted version of Theorem 1.2 in [18]). *Given an input permutation $\pi \in \mathbb{S}_n$ and weight functions $w_1, \ldots, w_4 : [n] \to \mathbb{Z}$, the count $\#_w 3214(\pi)$ can be computed in $\widetilde{\mathcal{O}}\left(n^{5/3}\right)$ time.*

Proof. See the full version [4]. ◀

4.4 Computing $\#43215$ in $\widetilde{\mathcal{O}}\left(n^{7/4}\right)$ time

▶ **Lemma 4.2.** *Let $\pi \in \mathbb{S}_n$ be a permutation. Then, $\#43215(\pi)$ can be computed in $\widetilde{\mathcal{O}}\left(n^{7/4}\right)$ time.*

Proof. See the full version [4]. ◀

4.5 Pair-Rectangle-Trees

The evaluation of the weighted marked pattern-count #43215 relies on a data structure that can efficiently count ascending and descending pairs in a given rectangle.

▶ **Theorem 4.3.** *(pair-rectangle-tree) There exists a data structure with the following properties:*

1. *Preprocessing: Given an input permutation $\pi \in \mathbb{S}_n$, the tree is initialised in time $\widetilde{\mathcal{O}}\left(n^2/q\right)$.*
2. *Query: Given any rectilinear rectangle $\mathcal{R} \subseteq [n] \times [n]$, return the number of descending (resp. ascending) pairs of permutations points in $\mathcal{R}$, in time $\widetilde{\mathcal{O}}(q)$.*

where $q = q(n) \in [n]$ is a parameter that can be chosen arbitrarily.

Proof. See the full version [4]. ◀

4.6 Algorithm for the 5-Profile

Proof of Theorem 1.2. The proof proceeds along the same lines as Theorem 1.1, for a different family of pattern-trees. Let $\mathbb{S} := \bigsqcup_{k=1}^{5} \mathbb{S}_k$. Extend the pattern-trees of maximum size 1 and over no more than 5 points, by allowing the new vertices described in Section 4.2. By computer enumeration, there exists a family of $\sum_{k=1}^{5} k! = 153$ linearly-independent vectors over $\mathbb{Q}^{\mathbb{S}}$, obtained from the vectors trees, along with their orbit under the action of D_4 on the symmetric group (see Section 2). The proof now follows, similarly to Theorem 1.1, and we remark that the evaluation of each pattern-tree over π takes at most $\widetilde{\mathcal{O}}\left(n^{7/4}\right)$ time, as that is the maximum amount of time spent handling any single vertex. ◀

5 Discussion

Some immediate extensions of this work, such as the application of our methods to the 8 and 9-profile, or the use of pattern-trees with maximum size 3, are computationally difficult and likely require further analysis or a different approach (say, algebraic). Several interesting open questions remain:

1. **Maximum size versus rank.** For a given integer s, denote by $f(s)$ the largest integer k such that the subspace spanned by the vectors of pattern-trees over at most k points, of maximum size s and with no equalities, is of full dimension, $|\mathbb{S}_{\leq k}|$. The results of [18] imply that $f(1) = 3$. In Section 3.3 and Section 3.4 we prove $f(2) = 7$, and in Section 3.5 we show that $f(s) \geq 2s$, for every $s \geq 1$. What is the behavior of $f(s)$? For example, do we have $f(s) \geq 4s \pm o(s)$, as attained by the technique of [7]?
2. **A fine-grained variant of $f(s)$.** For integers s and k, let $g(s, k)$ be the number of linearly independent vectors in $\mathbb{Q}^{\mathbb{S}_k}$ generated by Algorithm 1 when applied to trees of maximum size s over k permutation points with no equalities. What is the general behavior of $g(s, k)$? This generalises a question of [18] about corner-trees. The following values are presently known.

Table 1 Bold values in this table are computed in this paper (new).

s \ k	1	2	3	4	5	6	7	8
1	1	2	6	23	100	463	2323	**12173**
2	**1**	**2**	**6**	**24**	**120**	**720**	**5040**	**40319**

3. **Complexity of k-profile, for $5 \leq k \leq 7$.** Can the time complexity for finding the $5, 6, 7$-profiles be improved further, perhaps by utilising techniques along the lines of Section 4? In particular, we ask whether the 6-profile can be computed in sub-quadratic time.
4. **Study of the 8-profile.** [15] shows the equivalence between the computation of the 4-profile and counting 4-cycles in sparse graphs. In Section 3.4 we show that many of the observations of [15] can be extended to $\mathbb{S}_8$. In fact, we conjecture that there exists an analogous hardness result for $k = 8$, and we refer the reader to Section 3.4 where the details are discussed.

References

1 Shlomo Ahal and Yuri Rabinovich. On complexity of the subpattern problem. *SIAM Journal on Discrete Mathematics*, 22(2):629–649, 2008. doi:10.1137/S0895480104444776.

2 Michael H Albert, Robert EL Aldred, Mike D Atkinson, and Derek A Holton. Algorithms for pattern involvement in permutations. In *Algorithms and Computation: 12th International Symposium, ISAAC 2001 Christchurch, New Zealand, December 19–21, 2001 Proceedings 12*, pages 355–367. Springer, 2001. doi:10.1007/3-540-45678-3_31.

3 Erwin H Bareiss. Sylvester's identity and multistep integer-preserving gaussian elimination. *Mathematics of computation*, 22(103):565–578, 1968.

4 Gal Beniamini and Nir Lavee. Counting permutation patterns with multidimensional trees, 2024. doi:10.48550/arXiv.2407.04971.

5 Gal Beniamini and Nir Lavee. Computer-assisted computations for Theorems 1.1 and 1.2. https://github.com/perm-patterns-icalp2025/pattern-counting, 2025.

6 Gal Beniamini, Nir Lavee, and Nati Linial. How balanced can permutations be? *Combinatorica*, 45(1):1–31, 2025.

7 Benjamin Aram Berendsohn, László Kozma, and Dániel Marx. Finding and counting permutations via csps. *Algorithmica*, 83:2552–2577, 2021. doi:10.1007/S00453-021-00812-Z.

8 Prosenjit Bose, Jonathan F Buss, and Anna Lubiw. Pattern matching for permutations. *Information Processing Letters*, 65(5):277–283, 1998. doi:10.1016/S0020-0190(97)00209-3.

9 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988. doi:10.1137/0217026.

10 Bernard Chazelle and Herbert Edelsbrunner. Linear space data structures for two types of range search. *Discrete & Computational Geometry*, 2:113–126, 1987. doi:10.1007/BF02187875.

11 Joshua Cooper and Andrew Petrarca. Symmetric and asymptotically symmetric permutations. *arXiv preprint arXiv:0801.4181*, 2008.

12 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. Finding even cycles faster via capped k-walks. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 112–120, 2017. doi:10.1145/3055399.3055459.

13 Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366, 1989. doi:10.1016/0004-3702(89)90037-4.

14 Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2129–2138. IEEE, 2023. doi:10.1109/FOCS57990.2023.00130.

15 Bartłomiej Dudek and Paweł Gawrychowski. Counting 4-patterns in permutations is equivalent to counting 4-cycles in graphs. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

16 Paul Erdös and George Szekeres. A combinatorial problem in geometry. *Compositio mathematica*, 2:463–470, 1935.

17 Chaim Even-Zohar. Patterns in random permutations. *Combinatorica*, 40(6):775–804, 2020. doi:10.1007/S00493-020-4212-Z.

18 Chaim Even-Zohar and Calvin Leng. Counting small permutation patterns. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2288–2302. SIAM, 2021. doi:10.1137/1.9781611976465.136.

19 Jacob Fox. Stanley-wilf limits are typically exponential. *arXiv preprint arXiv:1310.8378*, 2013. arXiv:1310.8378.

20 Eugene C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of the Eighth National Conference on Artificial Intelligence - Volume 1*, AAAI'90, pages 4–9. AAAI Press, 1990. URL: http://www.aaai.org/Library/AAAI/1990/aaai90-001.php.

21 Sylvain Guillemot and Dániel 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. SIAM, 2014. doi:10.1137/1.9781611973402.7.

22 Joseph JáJá, Christian W Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Algorithms and Computation: 15th International Symposium, ISAAC 2004, Hong Kong, China, December 20-22, 2004. Proceedings 15*, pages 558–568. Springer, 2005.

23 Donald E Knuth. *The Art of Computer Programming: Fundamental Algorithms, Volume 1*. Addison-Wesley Professional, 1997.

24 Percy A MacMahon. *Combinatory analysis, volumes I and II*, volume 137. American Mathematical Society, 1915.

25 Adam Marcus and Gábor Tardos. Excluded permutation matrices and the Stanley–Wilf conjecture. *Journal of Combinatorial Theory, Series A*, 107(1):153–160, 2004. doi:10.1016/J.JCTA.2004.04.002.

26 Vaughan R Pratt. Computing permutations with double-ended queues, parallel stacks and parallel queues. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 268–277, 1973. doi:10.1145/800125.804058.

27 Rodica Simion and Frank W Schmidt. Restricted permutations. *European Journal of Combinatorics*, 6(4):383–406, 1985. doi:10.1016/S0195-6698(85)80052-4.

28 Virginia Vassilevska Williams, Joshua R Wang, Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on discrete algorithms*, pages 1671–1680. SIAM, 2014.

A Enumeration of Pattern-Tree Vectors

Let s and k be two positive integers, where $s \leq k$. Consider the following enumeration process, which computes the matrix whose rows are the vectors of all pattern-trees of maximum size $\leq s$, with exactly k points, restricted to $\mathbb{S}_k$.

For every ordered partition $\lambda \vdash k$ with no part larger than s, and for every vertex-labeled tree $T \in \mathbb{T}_{|\lambda|}$,[15] let T_λ be the tree in which vertex i has size $\lambda(i)$, and is assigned point variables

$$p(v_i) = \left\{p_{r_{i-1}+1}, \ldots, p_{r_i}\right\}, \text{ where } r_i := \sum_{j \leq i} \lambda(j), \text{ and } r_0 := 0.$$

Think of T_λ as a "template" for a pattern-tree, where the topology, the sizes of vertices, and the names of their variables have been determined, but the edge-constraints have not. Next, iterate over all pairs of permutations, $\sigma, \tau \in \mathbb{S}_k$, and over all trees T_λ.

Any such combination maps to a pattern-tree in the above family. For every i and j such that p_i and p_j are associated with the same vertex v, write the constraint $x_i < x_j$ in the vertex v if $\sigma(i) < \sigma(j)$, and write $x_i > x_j$ otherwise. Do likewise for the y constraints and τ, and repeat the same operation for every pair i, j such that the points p_i and p_j are associated with adjacent vertices in T_λ – in this case, we write the inequality on the edge. Observe that the ordering of points in each vertex is fully determined, i.e., defines a *permutation*.

Therefore, for any combination of T_λ, σ and τ, we obtain a pattern-tree T for which σ is a linear extension of the x-poset, and τ is a linear extension of the y-poset. In this case, we add 1 to the vector of T, at the index of $\tau\sigma^{-1}$ (see Lemma 3.6). Once the process is completed, we obtain a matrix with $|\mathbb{S}_k| = k!$ columns, and no more than $|\mathbb{S}_k \times \mathbb{S}_k \times \{T_\lambda\}|$ rows. The row-space of this matrix over the rationals is the subspace spanned by the above family of trees, restricted to $\mathbb{S}_k$.

We remark that if for every $k' \leq k$ this process produces a matrix of full rank, then by induction, the vectors of the union of all trees in these families spans the entire subspace, for $\mathbb{S}_{\leq k}$ (no tree over $k' < k$ points has a component in $\mathbb{S}_k$).

An implementation can be found in the script accompanying Theorem 1.1 and Theorem 1.2 [5].

[15] Here $\mathbb{T}_r$ is the set of all vertex-labeled trees over r vertices.