Kangas, Kustaa; Koivisto, Mikko; Salonen, Sami

A Faster Tree-Decomposition Based Algorithm for Counting Linear Extensions

Published in:
Algorithmica

DOI:
10.1007/s00453-019-00633-1

Published: 01/08/2020

Document Version
Publisher's PDF, also known as Version of record

Algorithmica
https://doi.org/10.1007/s00453-019-00633-1

A Faster Tree-Decomposition Based Algorithm for Counting Linear Extensions

Kustaa Kangas[1] · **Mikko Koivisto**[2] · **Sami Salonen**[2]

Received: 30 November 2018 / Accepted: 17 September 2019

Abstract
We investigate the problem of computing the number of linear extensions of a given n-element poset whose cover graph has treewidth t. We present an algorithm that runs in time $\tilde{O}(n^{t+3})$ for any constant t; the notation $\tilde{O}$ hides polylogarithmic factors. Our algorithm applies dynamic programming along a tree decomposition of the cover graph; the join nodes of the tree decomposition are handled by fast multiplication of multivariate polynomials. We also investigate the algorithm from a practical point of view. We observe that the running time is not well characterized by the parameters n and t alone: fixing these parameters leaves large variance in running times due to uncontrolled features of the selected optimal-width tree decomposition. We compare two approaches to select an efficient tree decomposition: one is to include additional features of the tree decomposition to build a more accurate, heuristic cost function; the other approach is to fit a statistical regression model to collected running time data. Both approaches are shown to yield a tree decomposition that typically is significantly more efficient than a random optimal-width tree decomposition.

Keywords Algorithm selection · Empirical hardness · Linear extension · Multiplication of polynomials · Tree decomposition

A preliminary version of this article appeared in Proceedings of the 13th International Symposium on Parameterized and Exact Computation (IPEC'18) [19]. This work was partially supported by the Academy of Finland, Grants 276864 and 316771.

✉ Mikko Koivisto
mikko.koivisto@helsinki.fi

Kustaa Kangas
kustaa.kangas@kapsi.fi

Sami Salonen
sami.m.salonen@helsinki.fi

[1] Department of Computer Science, Aalto University, Espoo, Finland

[2] Department of Computer Science, University of Helsinki, Helsinki, Finland

Published online: 03 October 2019

1 Introduction

The concept of a partially ordered set, or *poset* for short, formalizes the idea that an element of a ground set may precede, or "be smaller than," some other element, however allowing some pairs of elements be incomparable. The concept plays a fundamental role in various areas of mathematics, with applications in theoretical and applied computer science. In this paper, we will consider a particular computational problem associated with partial orders, namely the problem of counting the so-called linear extensions of a given poset. The problem has applications in numerous areas, for example, in sequence analysis [24], sorting [29], preference reasoning [23], convex rank tests [26], partial order plans [27], and learning graphical models [28,34].

To formulate the problem more formally, consider a poset $(V, \prec)$ formed by an n-element set V and an irreflexive and transitive binary relation $\prec$ on V, called a *partial order*. Another partial order $<$ on V is a *linear extension* of $\prec$ if it contains $\prec$ and for any distinct elements $x, y \in V$ either $x < y$ or $y < x$. The problem of counting linear extensions (#LE) asks for the number of linear extensions of a given poset; #LE is equivalent to the problem of counting the topological sorts of a given directed (acyclic) graph.

Findings an efficient algorithm for #LE seems unlikely, as the problem is #P-complete [11]. However, #LE admits a fully polynomial randomized approximation scheme [13]. Currently, the best known asymptotic bounds for the expected running time are $O(\epsilon^{-2} n^3 \log^2 \ell \log n)$ [6] and $O(\epsilon^{-2} n^5 \log^2 n)$ [32], where ℓ is the number of linear extensions and ϵ the allowed relative error (with any constant success probability). These schemes, while polynomial in n and ϵ^{-1}, become prohibitively slow in practice if, say, one requires an accuracy of $\epsilon = 0.01$ and n is around one hundred.

Exact and parameterized algorithms offer an alternative paradigm to design practical algorithms for the problem. If we measure the complexity of an algorithm by the required number of arithmetic operations, which is a common practice in the literature, the best known worst-case bound is $O(2^n n)$, achieved by a simple dynamic programming algorithm. For several special instance classes better bounds are known: $O(n^w w)$ for width-w posets, $O(n^2)$ for series–parallel posets [25], and $O(n^2)$ also for posets whose cover graph is a forest [5]; the *cover graph* of a poset $(V, \prec)$ is the directed graph (V, E) where the edge set E is the transitive reduction of $\prec$ (a.k.a. Hasse diagram). If parameterized by the treewidth of the cover graph, t, the problem can be solved with $O(n^{t+3})$ arithmetic operations by an inclusion–exclusion algorithm [18]. On the other hand, the problem parameterized by t is W[1]-hard [14], and so it seems difficult to find an algorithm that would run in time $O(f(t)n^d)$ for some computable function f and constant d.

In this article, we investigate whether there exist faster exact algorithms for #LE, parameterized by the treewidth of the cover graph. We are interested in the worst-case asymptotic time complexity as well as the running time on moderate-size instances in practice. This article extends a preliminary version of the work published in a conference proceedings [19]; in the next two subsections we introduce our main contributions and highlight the features that are new in the present article.

1.1 Theoretical Contributions

Throughout this paper, we write $\tilde{O}(f)$ and $\tilde{\Omega}(f)$ as shorthands for $f \cdot \log^{O(1)} f$ and $f / \log^{O(1)} f$, respectively; in other words, the notations hide factors that are polylogarithmic in its argument. Our main result is the following:

Theorem 1 *The linear extensions of a given n-element poset can be counted with $\tilde{O}(t!\, n^{t+3})$ bit-operations, where t is the treewidth of the cover graph of the poset.*

We state the bound in terms of bit-complexity to emphasize the fact that the dominating computations deal with addition and multiplication of large integers. Indeed, the bounds stated in the previous paragraphs refer to the number of arithmetic operations with $O(n \log n)$-bit integers. Thus, in particular, for any constant t our bound improves the previous bound of Kangas et al. [18] by a factor of n, up to polylogarithmic factors. For large n and small t the improvement is relatively significant; for instance, for $t = 2$ the bound is reduced from $\tilde{O}(n^6)$ to $\tilde{O}(n^5)$. For large t, however, the present bound is inferior due to the factor $t!$. It turns out that we can, in fact, get rid of this factor under certain assumptions that are relatively mild in practice, while too complicated theoretically to be included in a succinct theorem statement.

Perhaps more interestingly, our algorithm is radically different from the inclusion–exclusion algorithm. In the latter, the idea is to view a linear extension as a bijective mapping and then remove the global bijectivity constraint by inclusion–exclusion, similarly to previous applications to matrix permanent [31], Hamiltonian path [20], and set partitioning [21], but incurring only a polynomial overhead. Once the bijectivity constraint is removed, what remains is a collection of simpler subproblems with local constraints. The subproblems can be handled by standard routines that exploit small treewidth [7,12]; see Sect. 2.2 for some additional details.

The present algorithm, in contrast, takes care of the bijectivity constraint within dynamic programming along a tree decomposition and is, with this respect, similar to a folklore $t^{O(t)} n$-time algorithm for the Hamiltonian path problem. However, #LE being W[1]-hard one may expect it to require a significantly larger dynamic programming table. We give a formulation, where each node of a tree decomposition is associated with $O(n^{t+1})$ counts. This formulation leads to a challenge: a step in the dynamic program that combines two (or more) arrays of such counts appears to require, in the worst case, a quadratic number of arithmetic operations, $\Theta(n^{2t+2})$, if implemented in a straightforward manner. Fortunately, we discover that the key ingredient of the step takes a form of multidimensional convolution, which we can compute efficiently using known results for fast multiplication of multivariate polynomials.

Concerning space complexity we only make a couple of observations here: Both our algorithm and the inclusion–exclusion algorithm by Kangas et al. [18] require $\tilde{O}(n^{t+2})$ bits of space. One could reduce this by a factor about linear in n by carrying the computations modulo several small relative primes and constructing the final output using the Chinese remainder theorem. For comparison, the simple dynamic programming algorithm also requires lots of space, $\tilde{\Omega}(2^n)$ bits.

In addition to some minor changes in exposition, the present article gives a more detailed treatment of the needed result for multiplication of multivariate polynomials.

Specifically, we include the calculations omitted in the preliminary version [19] and also correct a minor error in the statement of the needed result (Fact 2, Sect. 2).

1.2 Empirical Contributions

We also address the practical value of the algorithm. Given that the present algorithm is technically more convoluted than the inclusion–exclusion algorithm, it is natural to ask, whether the obtained improvement in the asymptotic worst case time requirement is reflected as significant expedition in practice. Our interest is particularly in instances where t is small (at most four) and n ranging up to a few hundred.

A well known challenge in practical implementation of tree-decomposition based algorithms is that finding an optimal-width tree decomposition may be insufficient for minimizing the computational cost: the running time of the dynamic programming algorithm can be sensitive to the shape of the tree decomposition. Bodlaender and Fomin [9] addressed this issue from a theoretical viewpoint by studying the complexity of finding a tree decomposition that minimizes a sum of costs associated with each node of a tree decomposition. In their *f-cost* framework the cost of a node is allowed to depend only on the width of the node (i.e., the size of the associated bag; see Sect. 2). Recently, Abseher et al. [1,2] presented a more general and more practical heuristic approach. Their `htd` library [1] allows a user to generate a variety of optimal-width tree decompositions and also (locally) optimize a given cost function. Moreover, they proposed and evaluated [2] a method to learn an appropriate cost function, or regression model, from empirical running time data on a collection of "training" instances. The method can be viewed as an instantiation of the method of empirical hardness models [22] for the algorithm selection problem [30].

Following these ideas we have implemented and tested our algorithm for #LE using a collection of synthetically generated instances (posets) together with a variety of tree decompositions generated by `htd` for each instance. We will report on and discuss our observations, which suggest that selecting the tree decomposition using a learned regression model can make a difference, at least for the smallest treewidth ($t = 2$): compared to the median running time over generated tree decompositions, the selected one typically yields almost an order-of-magnitude speedup.

The present work extends the preliminary study [19] with two additions. First, we include a direct comparison to an implementation of the inclusion–exclusion algorithm, `VEIE` [18]. Second, we also include in the experiments a new heuristic cost function, and show that its performance is competitive to that of the learned model.

1.3 Organization

The rest of this article is organized as follows. Some basic terminology, notation, and facts are given in Sect. 2. Section 3 is devoted to proving Theorem 1. In Sect. 4 we describe some implementation details and report on empirical results. Section 5 concludes by summarizing and discussing the main observations.

2 Preliminaries

We denote by $\mathbb{N}$ the set of natural numbers $\{0, 1, 2, \ldots\}$. For two sets S and U we write S^U for the set of functions from U to S. If $m \in \mathbb{N}$ we write $[m]$ for the set $\{1, \ldots, m\}$. By S^m we denote the set of m-tuples $\mathbf{a} = (a_i)_{i=1}^m$ with $a_i \in S$. The *restriction* of a function $\alpha : U \to S$ to a subset $A \subset U$ is defined in the standard manner and denoted by $\alpha_{|A}$; conversely, we say that α is an *extension* of $\alpha_{|A}$. We also denote by $\alpha^{v \mapsto i}$ the extension of α that we obtain by mapping an added element $v \notin U$ to i. We use the Iverson's bracket notation: for a proposition P, the expression $[P]$ evaluates to 1 if P is true, and to 0 otherwise.

Let $\mathbf{a} = (a_1, \ldots, a_k) \in \mathbb{N}^k$. We denote $\mathbf{a}! := a_1! \cdots a_k!$ and $|\mathbf{a}| := a_1 + \cdots + a_k$. Since $|\mathbf{a}|!/\mathbf{a}!$ is a multinomial coefficient, we have the following.

Fact 1 *If* $\mathbf{a}$ *is a tuple of nonegative integers, then* $\mathbf{a}!$ *divides* $|\mathbf{a}|!$.

Another fact we need concerns the complexity of multiplying two multivariate polynomials. We assume that a polynomial is represented by a list of its coefficients. It is well known that the multiplication takes nearly linear time when parameterized by the sum of the *maximum degrees*. We derive the following formulation of the result from a more general bound by van der Hoeven and Lecerf [33, Cor. 1] in Appendix.

Fact 2 *Two* k*-variate polynomials whose maximum degrees sum up to* $n \geq k$ *and whose coefficients are* $\tilde{O}(n)$*-bit integers can be multiplied with* $\tilde{O}(n^{k+1})$ *bit-operations.*

While the above result will suffice for proving Theorem 1, it does not exploit the property that the multivariate polynomials we encounter are, in fact, sparse in the sense that they have a *total degree* at most n. Also in this case, multiplication only requires nearly linear time, now amounting to $\tilde{O}\left(\binom{n+k}{k} nk\right)$ bit-operations, assuming a sufficiently large prime p (exponential in n) and a primitive element of the finite field $\mathbb{F}_p$ are given for free [33, Cor. 4]. This assumption is, however, relatively strong: we do not know, whether it can be satisfied within the time complexity bound, or even in time polynomial in n, without some number theoretic hypotheses. On the other hand, the needed numbers only depend on n and finding them can be considered as a precomputation that is efficient in practice; for further details, see the discussion of by van der Hoeven and Lecerf [33, Sect. 5.3]. The improved bound for multiplying sparse multivariate polynomials saves a factor of about $k!$, which suffice for reducing the factor $t!$ in the bound of Theorem 1, as mentioned in the previous section.

2.1 Tree Decomposition

Definition 1 (*Tree decomposition*) A *tree decomposition* of a graph $G = (V, E)$ is a pair (T, B) where $T = (I, F)$ is a tree and B maps each node $x \in I$ to a *bag* $B_x \subseteq V$ such that

1. for each $v \in V$, the node set $\{x \in I : v \in B_x\}$ induces a nonempty subtree of T,
2. for each $uv \in E$, there exists a node $x \in I$ with $u, v \in B_x$.

The *width* of the tree decomposition is the largest bag size minus one, $\max_{x \in I} |B_x| - 1$, and the *treewidth* of a graph is the minimum width over all its tree decompositions.

For a graph of treewidth t, we can find a tree decomposition of width t in time $t^{O(t^3)}n$ using Bodlaender's algorithm [8] or in time $\tilde{O}(n^{t+2})$ using the approach of Arnborg et al. [4].

A tree decomposition is *rooted* if the edges of the tree are directed so that there is a unique node, the *root*, that has no parent. Clearly, one obtains a rooted tree decomposition by simply choosing one node as the root and directing the edges accordingly.

Definition 2 (*Nice tree decomposition*) A rooted tree decomposition (T, B) of a graph $G = (V, E)$ is *nice* if each node x of T is of one of the following types:

(leaf)	x has no children and $\lvert B_x\rvert = 1$;
(introduce)	x has a unique child y and $B_x = B_y \cup \{v\}$ for some $v \in V \setminus B_y$;
(forget)	x has a unique child y and $B_x = B_y \setminus \{v\}$ for some $v \in B_y$;
(join)	x has exactly two children y, z and $B_x = B_y = B_z$.

We can convert a given tree decomposition of width t and $O(n)$ nodes into a nice tree decomposition of width t and $O(tn)$ nodes in time $t^{O(1)}n$ [10].

2.2 Counting Linear Extensions via Inclusion–Exclusion

Kangas et al. [18] showed that the number of linear extensions of a poset $(V, \prec)$ whose cover graph is $G = (V, E)$ is given by the formula

$$\sum_{k=1}^{n} \binom{n}{k} (-1)^{n-k} \sum_{\tau} \prod_{uv \in E} [\tau(u) < \tau(v)],$$

where τ runs over all functions from V to $[k]$.

Supposing G has treewidth t, it is well known that there is an elimination ordering $v_1, \ldots, v_n$ of the vertices, such that when removing the vertices from the graph in this order and always connecting the neighbors of the removed vertex, the size of the largest clique in each obtained graph is $t+1$. The n-dimensional inner summation over the variables $\tau(v_i)$, for $i \in [n]$, can be processed iteratively along such an ordering, the ith one-dimensional summation over $\tau(v_i)$ requiring $O(n_i k^{t+1})$ additions and multiplications of $O(n \log n)$-bit numbers, for some $n_1, \ldots, n_n$ that sum up to $O(n)$. In total, the evaluation of the inclusion–exclusion formula thus requires $\tilde{O}(n^{t+4})$ bit-operations. We omit a more detailed treatment of the algorithm, as the method applied for computing the inner summation is standard. (The original analysis of Kangas et al. [18] uses a looser bound of $n_i = O(n)$ for each i, arriving at a bound that is larger by a factor of n.)

3 The Algorithm: Proof of Theorem 1

We implement a standard recipe of tree-decomposition based algorithms. The outline of the algorithm is as follows.

A1 Compute the cover graph of the input poset.

A2 Find a minimum-width nice tree decomposition of the cover graph.
A3 Run dynamic programming over the nice tree decomposition.

We will next consider each step in detail. We will see that the last step dominates our asymptotic running time bounds.

3.1 Computing the Cover Graph

The cover graph $G = (V, E)$ is obtained by computing the transitive reduction of the input poset $(V, \prec)$. The transitive reduction can be computed in time $O(|V| \cdot |\prec|) = O(n^3)$ [3], which is $O(n^{t+2})$ for all $t \geq 1$.

3.2 Finding a Minimum-Width Nice Tree Decomposition

As mentioned in Sect. 2, if the cover graph has treewidth t, then a width-t nice tree decomposition of the cover graph can be found in $\tilde{O}(n^{t+2})$ time.

3.3 Dynamic Programming

Suppose now that a width-t nice tree decomposition (T, B) of the cover graph G is available. Our idea will be to associate each node of T with an array of numbers such that (i) the numbers at the root node are sufficient for computing the number of linear extensions and (ii) the array of a node can be computed from the arrays of its child nodes.

The following notation will be useful. Denote by V_x the set of vertices covered by the subtree of T rooted at x, that is, V_x is the union of the bags B_y of nodes y to which there is a directed path from x. Write n_x for the size $|V_x|$ and E_x for set of edges in the induced graph $G[V_x]$.

Now, for each node $x \in T$ and injection $\alpha \in [n_x]^{B_x}$, define $\ell_x(\alpha)$ as the number of bijections $\pi \in [n_x]^{V_x}$ such that $\pi(v) = \alpha(v)$ for all $v \in B_x$, and $\pi(u) \prec \pi(v)$ whenever $uv \in E_x$. In other words, $\ell_x(\alpha)$ is the number of ways to extend α to a linear extension of the induced poset $(V_x, \prec \cap (V_x \times V_x))$, where we view a linear extension as a bijection from V_x to $[n_x]$ that satisfies the ordering constraints.

We begin by showing that the values $\ell_x(\alpha)$ are sufficient for computing the number of linear extensions of the poset, that is, they satisfy the listed conditions (i) and (ii). After that we consider the time requirement of computing the values $\ell_x(\alpha)$ for each node of the nice tree decomposition.

Consider first the root node.

Lemma 1 (Root) *We have* $\ell(V) = \sum_\alpha \ell_r(\alpha)$, *where* α *runs over all injections in* $[n_x]^{B_r}$.

Proof Since $V_r = V$, $E_r = E$, and $n_r = n$, we have that

$$\sum_\alpha \ell_r(\alpha) = \sum_\pi \prod_{uv \in E} [\pi(u) < \pi(v)] = \ell(V),$$

where α and π run over all injections in $[n]^{B_r}$ and $[n]^V$, respectively. □

Next we will show separately for each node type of the nice tree decomposition, how the values $\ell_x(\alpha)$ are determined by the corresponding values for the child node or child nodes of x. For all but join nodes the results are immediate, and we omit the proofs.

Lemma 2 (Leaf) *If x is a leaf node, then $\ell_x(\alpha) = 1$ for the unique injection α in $[n_x]^{B_x}$.*

For an introduce node, we simply restrict the injection α to the bag of its child and check that the ordering constraint holds.

Lemma 3 (Introduce) *If x is an introduce node with child y and $B_x = B_y \cup \{v\}$, then*

$$\ell_x(\alpha) = \ell_y(\alpha_{|B_y}) \prod_{\substack{u \in B_x \\ uv \in E}} [\alpha(u) < \alpha(v)].$$

For a forget node, we extend the injection α to the bag of its child by mapping the new vertex to some value that is not in the image of α.

Lemma 4 (Forget) *If x is a forget node with child y and $B_x = B_y \backslash \{v\}$, then*

$$\ell_x(\alpha) = \sum_{a \in [n_y] \backslash \alpha(B_x)} \ell_y(\alpha^{v \mapsto a}).$$

To handle a join node, we introduce some convenient notation. Let α be an injection from a k-element set S to a range of integers $[m]$. Label the elements of S so that $\alpha(v_1) < \cdots < \alpha(v_k)$. For $i = 1, \ldots, k-1$, denote by α_i the number of integers between $\alpha(v_i)$ and $\alpha(v_{i+1})$, that is, $\alpha_i := \alpha(v_{i+1}) - \alpha(v_i) - 1$; in addition, denote $\alpha_0 := \alpha(v_1) - 1$ and $\alpha_k := m - \alpha(v_k)$. Observe that $\alpha_0 + \cdots + \alpha_k = m - k$. Furthermore, if β is another injection from S' to $[m']$, write $\beta \sim \alpha$ if β and α specify the same linear order on $S \cap S'$, that is, $\beta(u) < \beta(v)$ if and only if $\alpha(u) < \alpha(v)$ for all $u, v \in S \cap S'$.

Lemma 5 (Join) *If x is a join node with children y and z, then*

$$\ell_x(\alpha) = \sum_{\beta} \sum_{\gamma} [\alpha \sim \beta \sim \gamma] \prod_{i=0}^{|B_x|} [\alpha_i = \beta_i + \gamma_i] \binom{\alpha_i}{\beta_i} \ell_y(\beta)\, \ell_z(\gamma),$$

where β and γ run over all injections in $[n_y]^{B_y}$ and $[n_z]^{B_z}$, respectively.

Proof By definition,

$$\ell_x(\alpha) = \sum_{\pi} \prod_{uv \in E_x} [\pi(u) < \pi(v)],$$

where π runs over all bijections from V_x to $[n_x]$ that extend α. Observe that by the tree decomposition properties, the sets $V_y \setminus B_x$ and $V_z \setminus B_x$ are disjoint and their union is $V_x \setminus B_x$. Thus we may represent any bijection $\pi : V_x \to [n_x]$ that extends α uniquely by a pair of injections $\beta' : V_y \to [n_x]$ and $\gamma' : V_z \to [n_x]$ whose restrictions to B_x are equal to α and whose images $\beta'(V_y)$ and $\gamma'(V_z)$ cover $[n_x]$. We get that

$$\ell_x(\alpha) = \sum_{\beta'} \sum_{\gamma'} \big[\beta'(V_y) \cup \gamma'(V_z) = [n_x]\big] \prod_{uv \in E_y} \big[\beta'(u) < \beta'(v)\big] \prod_{uv \in E_z} \big[\gamma'(u) < \gamma'(v)\big],$$

where β' and γ' run over all injections that extend α in $[n_x]^{V_y}$ and $[n_x]^{V_z}$, respectively.

Consider then a mapping that "compresses" any such injection β' into a bijection $\beta'' : V_y \to [n_y]$ by letting $\beta''(v) := |\{u \in V_y : \beta'(u) \leq \beta'(v)\}|$; let γ'' denote the bijection obtained similarly from an injection γ'. Let β denote the restriction of β'' to B_x and γ the restriction of γ'' to B_x. We have that β' and γ' extend α if and only if $\beta \sim \alpha$ and $\gamma \sim \alpha$. Thus we get that

$$\ell_x(\alpha) = \sum_{\substack{\beta'' \\ \beta \sim \alpha}} \sum_{\substack{\gamma'' \\ \gamma \sim \alpha}} \prod_{i=0}^{|B_x|} [\alpha_i = \beta_i + \gamma_i] \binom{\alpha_i}{\beta_i} \prod_{uv \in E_y} \big[\beta''(u) < \beta''(v)\big] \prod_{uv \in E_z} \big[\gamma''(u) < \gamma''(v)\big],$$

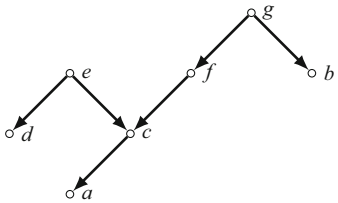
where β'' and γ'' run over all bijections in $[n_y]^{B_y}$ and $[n_z]^{B_z}$, respectively. The product of the binomial coefficients $\binom{\alpha_i}{\beta_i}$ is the number of pairs (β', γ') that map to the same pair (β'', γ''), that is, the number of interleavings of the $\beta_i + \gamma_i$ elements to the range of α_i elements.

To complete the proof, it suffices to write the summation over β'' as a double-summation: the outer summation being over all injections $\beta : B_y \to [n_y]$ and the inner summation being over all bijections $\beta'' : V_y \to [n_y]$ that extend β; similarly for the summation over γ''. □

Example 1 (Join in a tree) Consider the example illustrated in Fig. 1. The cover graph is a tree with vertex set $V = \{a, b, c, d, e, f, g\}$. In a nice tree decomposition (not shown) the root node x is a join of nodes y and z, with $B_y = B_z = B_x = \{c, f\}$. The vertex sets associated with the nodes are $V_x = V$, $V_y = \{a, c, d, e, f\}$, and $V_z = \{b, c, f, g\}$. For the shown injection α, the value of $\ell_x(\alpha)$ is obtained by a sum of $\ell_y(\beta) \cdot \ell_z(\gamma)$ over valid pairs (β, γ), multiplied by the number of possible interleavings, which is given by a product of binomial coefficient. Shown is one pair (β, γ), for which $\ell_y(\beta) = 1$ and $\ell_z(\gamma) = 1$ and the number of interleavings is equal to $\binom{2}{1}\binom{2}{2}\binom{1}{0} = 4$.

It remains to bound the running time of the algorithm.

Lemma 6 (Time complexity) *Given a width-t nice tree decomposition of the cover graph of an n-element poset, the linear extensions can be counted with $\tilde{O}(t!\, n^{t+3})$ bit-operations.*



Injection	1	2	3	4	5	6	7
α	a	b	$\boldsymbol{c}$	d	e	$\boldsymbol{f}$	g
β	a	$\boldsymbol{c}$	d	e	$\boldsymbol{f}$		
γ	b	$\boldsymbol{c}$	$\boldsymbol{f}$	g			

Fig. 1 An illustration of the recurrence for a join node with bag $\{c, f\}$; see Example 1 for a description

Proof Let x be a node in the nice tree decomposition. For brevity, denote $k := |B_x|$. If x is a leaf node, introduce node, or forget node, then the values $\ell_x(\alpha)$ for all injections $\alpha \in [n_x]^{B_x}$ can clearly be computed using $O(kn^k) = O(tn^{t+1})$ basic operations, some of which are additions of two $O(n \log n)$-bit numbers, thus using $\tilde{O}(n^{t+2})$ bit-operations (observe that the factor t is $O(\log n^{t+2})$ and can thus be omitted).

Consider then the remaining case: x is a join node. Let y and z be the two children of x. Recall that $B_x = B_y = B_z$.

Represent an injection α in $[n_x]^{B_x}$ as a pair $(\sigma, \mathbf{a})$, where $\mathbf{a} = (a_i)_{i=1}^{k+1}$ with $a_i = \alpha_{i-1}$ and σ is a bijection from B_x to $[k]$ that captures the specified linear order, that is, $\sigma(u) < \sigma(v)$ if $\alpha(u) < \alpha(v)$. Clearly, the mapping $\alpha \mapsto (\sigma, \mathbf{a})$ is a bijection when we require that $a_i \in \mathbb{N}$ and $|\mathbf{a}| = n_x - k$. Using this representation and Lemma 5, write

$$\ell_x(\sigma, \mathbf{a}) = \sum_{\mathbf{b}} \sum_{\mathbf{c}} \prod_{i=1}^{k+1} [a_i = b_i + c_i] \binom{a_i}{b_i} \ell_y(\sigma, \mathbf{b})\, \ell_z(\sigma, \mathbf{c}),$$

where $\mathbf{b}$ and $\mathbf{c}$ run over $\mathbb{N}^{k+1}$. By writing $\ell'_x(\sigma, \mathbf{a}) := \ell_x(\sigma, \mathbf{a})/\mathbf{a}!$, we get the convolution form

$$\ell'_x(\sigma, \mathbf{a}) = \sum_{\mathbf{a}=\mathbf{b}+\mathbf{c}} \ell'_y(\sigma, \mathbf{b})\, \ell'_z(\sigma, \mathbf{c}).$$

To treat this as a multiplication of multivariate polynomials, consider a fixed bijection σ and let $P_x(r_1, \ldots, r_k)$ be the k-variate polynomial where the coefficient of $r_1^{a_1} \cdots r_k^{a_k}$ equals $n! \cdot \ell'_x(\sigma, \mathbf{a})$; we define P_y and P_z similarly. Note that k variables suffice, since a_{k+1} is determined by the fixed $|\mathbf{a}|$. Here we multiplied by the factorial $n!$ to get integer coefficients (by Fact 1, since $n \geq |\mathbf{a}|, |\mathbf{b}|, |\mathbf{c}|$). We have that $n! \cdot P_x = P_y P_z$, that is, we obtain P_x by multiplying P_y and P_z and dividing each coefficient of the resulting polynomial by $n!$.

We bound the bit-complexity of the polynomial multiplication using Fact 2. The total degrees of the polynomials are at most $n - k$. Each coefficient of the polynomials is a $\tilde{O}(n)$-bit integer. Thus the multiplication takes $\tilde{O}(n^{k+1})$ bit-operations.

Multiplying the obtained bound by the number of bijections σ, we get that all $\ell_x(\sigma, \mathbf{a})$ can be computed using $\tilde{O}(k!\, n^{k+1}) = \tilde{O}(t!\, n^{t+2})$ bit-operations.

Since there are $O(tn)$ nodes in the nice tree decomposition, $\tilde{O}(t!\, n^{t+3})$ bit-operations suffice in total. □

4 Experiments

In this section we present an empirical study. We first describe our implementation of the algorithm and the test instances used in the experiments. Then we show how the performance on the algorithm depends on the way we choose the tree decomposition.

4.1 Implementation

We have implemented the algorithm of Sect. 4 in a C++ program `Countle`.[1] For multiplication of polynomials we used the C library `FLINT` [17], which offers fast multiplication of univariate polynomials; to this end, we transformed the multivariate polynomials into univariate polynomials using an appropriate Kronecker substitution, as detailed in the next paragraph. We decided to not employ the asymptotically faster algorithm mentioned after Fact 2, because it has been observed to run faster only when the total degree (which is less than n in our case) is large, say, several hundreds [33, Table 7]. For finding an optimal tree decomposition we used the C++ library `htd` [1]. We ran all experiments on machines with Intel Xeon E5540 CPUs.

Two implementation details are worth mentioning. First, we transformed the multivariate polynomials into univariate polynomials using the following Kronecker substitution. Separately for each node x of the tree decomposition and the considered vertex ordering (bijection) σ, we encoded a k-variate polynomial in variables $r_1, \ldots, r_k$ as a univariate polynomial in variable s by substituting $r_j := s^{(d_1+1)\cdots(d_{j-1}+1)}$, where each d_i is an upper bound for the degree of r_i in the polynomial. Using knowledge associated with the node x and ordering σ, we aimed at determining a value d_i that is smaller than the trivial upper bound $n_x - k$. To this end, we set d_i to the sum of the largest realized exponents of r_i in the already computed polynomials for the two child nodes of x.

Second, we wish to ignore any impossible ordering σ at a node x of the tree decomposition, and so save both time and space. The key observation is that, even if the value $\ell_x(\sigma, \mathbf{a})$ is nonzero, we can ignore it if σ assigns some two vertices in the bag B_x an order that violates the partial order $\prec$, that is, for some $u, v \in B_x$ we have $u \prec v$ and $\sigma(u) > \sigma(v)$.

4.2 Instances

We generated random instances (posets) of different sizes for small values of treewidth t. We varied the number of elements n from 10 to 199 ($t = 2$), 109 ($t = 3$), and 59 ($t = 4$). For each pair (t, n) we generated 5 posets; following Kangas et al. [18], we let each poset be a "grid tree," constructed by randomly joining t-by-t grids along the (boundary) edges, orienting the edges so that no directed cycles are introduced, and finally taking the transitive closure.

[1] `Countle` is free and publicly available at https://bitbucket.org/samsalo/countledist/.

4.3 Growth and Variation of Running Times

The running time of `Countle` may be sensitive to the particular (nice) tree decomposition selected. Therefore, we ran the program on 50 optimal-width tree decompositions, which we generated using `htd`; we checked that for every encountered instance, `htd` indeed generated tree decompositions of optimal width. We allowed each individual run to take up to 20 min of CPU time and 30 GB of memory.

We examined the scaling of `Countle` in terms of the number of elements n and treewidth t. We observed that, while the growth of the running time follows the rate suggested by the worst case bound, there is significant variance in the running times for any fixed (n, t), due to differences in the five posets and the 50 tree decompositions per poset (compare *median* to *best* in Fig. 2). Compared to an implementation of the inclusion–exclusion algorithm by Kangas et al. [18], `VEIE`, we find that `Countle` is an order of magnitude faster. For example, `Countle` can solve a typical (median) poset with a typical (median) tree decomposition in about 20 s if $n = 100$ and $t = 2$, or if $n = 50$ and $t = 3$, while `VEIE` requires about 200 s on such instances. The same pattern also holds for $t = 4$; we note, however, that such posets can actually be handled faster by another, exponential time algorithm of Kangas et al. [18, Fig. 8]. `VEIE` does not optimize the shape of the tree decomposition beyond its width and, in fact, the analysis of Sect. 2.2 suggests that the algorithm is not very sensitive to the shape of the tree decomposition.

4.4 Selecting Among Optimal-Width Tree Decompositions

Then we investigated whether one can efficiently select a near-optimal tree decomposition from a collection of generated candidates. The observed variance in running times suggests that, if successful, this could lead to a significant expedition of `Countle`, by up to one order of magnitude. For constructing a "selector" we implemented and compared two approaches.

One is the machine learning method of Abseher et al. [2]. We applied it in a straightforward manner, as follows:

C1 We collected a data set of measured running times for multiple pairs of posets and tree decompositions. We used the procedure described in the previous section, except that we used a single poset (instead of five) for each combination of n and t. If a run was not completed within the 20-min time limit, we simply discarded the instance (and thus introduced some bias).

C2 We computed for each tree decomposition the values of several features, such as statistics of bag sizes (by node type), node depths (by node type), and distances between join nodes; for a full feature list, see Abseher et al. [2].

C3 We fitted a multivariate linear regression model, separately for each $t = 2, 3, 4$, with the features as the predictor variables and the logarithm of the running time as the response variable. We used the machine learning software `WEKA 3.6.13` [16] with default options.

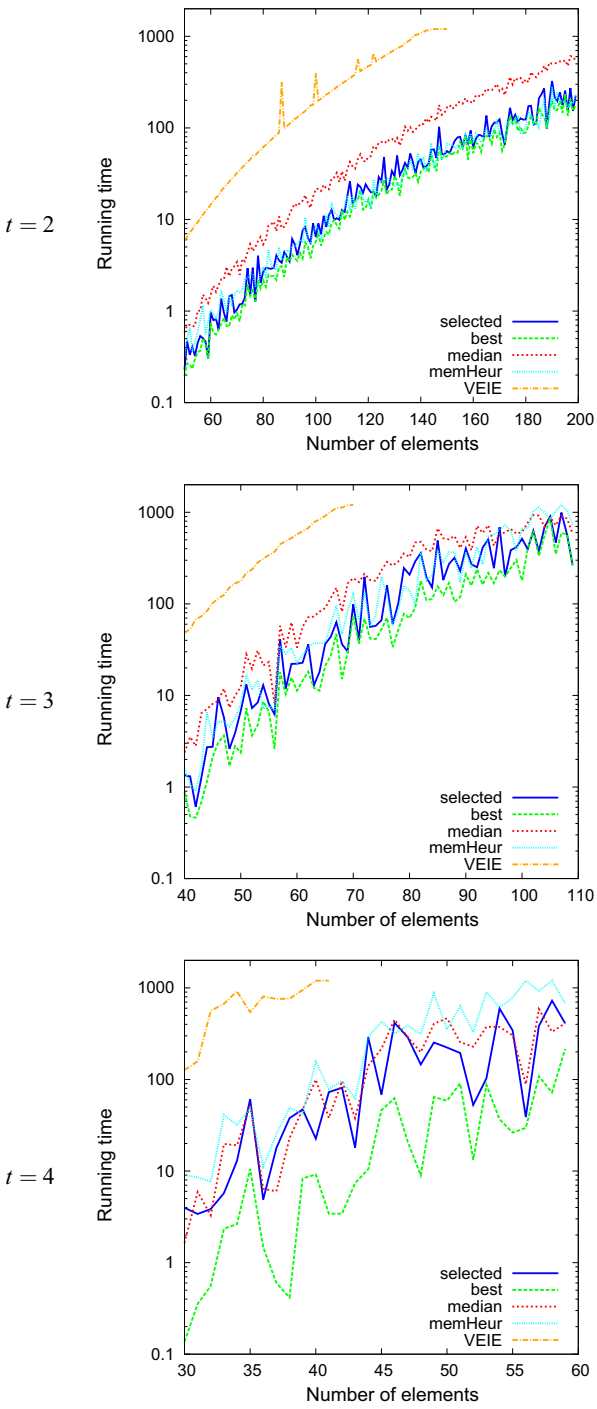


Fig. 2 The running time of `Countle` (s) on random "grid tree" posets of treewidth 2, 3, and 4, with a varying number of elements n. For each poset, we generated 50 optimal-width tree decompositions and collected the median running time and the running times for the best tree decomposition and the tree decompositions proposed by the machine learning method ("selected") and by the memory heuristic ("memHeur"). For comparison, the running times of `VEIE` [18] are also shown. The shown values are the medians of these five statistics over five independent posets for each value of n.

To select a tree decomposition for a given new poset, we first generated 50 candidate tree decompositions for the poset, and then selected the one for which the model predicted the shortest running time.

The other approach is to "handcraft" a heuristic cost function based on a smaller set of features of the tree decomposition. Our cost function, we call *memory heuristic*, is simply a sum of node-wise costs. Specifically, for each node x, we define its cost as $p_x \binom{n_x}{|B_x|} n_x \log_2 n_x$, where p_x is the number of orderings σ of B_x compatible with the poset; this cost is an upper bound (up to a constant factor) of the memory requirement of the corresponding dynamic programming step, obtained from the algorithm in a straightforward manner. The rationale is that for each node the memory requirement is well aligned with the time requirement and that the node-wise bounds are more accurate than the worst case bound over all nodes.

We observe (Fig. 2) that, for $t = 2$, the learned model almost always selects a top-3 tree decomposition, which yields a nearly as short running time as the best among the 50 tree decompositions. For $t = 3$ the performance degrades: the model is usually able to select a top-10 tree decomposition, which yields a running time that is systematically better than for a typical (median) tree decomposition, yet not quite achieving the performance of the best among the generated candidates. For $t = 4$ the performance degrades further, yet being better than by selecting a random tree decomposition. In more quantitative terms, the proportions of tree decompositions (among 50) better than the selected one were 2.5%, 12%, 28% for $t = 2, 3, 4$, respectively; these numbers are medians of averages over 5 test posets (one per fixed n and t).

The memory heuristic is seen to achieve almost as good performance as the machine learning method for $t = 2$ and $t = 3$ (Fig. 2). For $t = 4$, however, memory heuristic is inferior and even yields tree decompositions that are worse than a random (median) tree decomposition.

5 Concluding Remarks

We have presented a new tree-decomposition based algorithm for counting linear extensions. The algorithm relies on fast multiplication of multivariate polynomials, thus differing radically from the inclusion–exclusion approach of Kangas et al. [18]. For any constant treewidth t the obtained asymptotic speedup is about linear in the number of elements n.

A question not settled here is whether one could save another factor of n, that is, solve the problem in time $\tilde{O}(n^{t+2})$. The present authors find this question particularly intriguing for two reasons: one is that for finding an optimal-width tree decomposition, the best known time complexity bound is $\tilde{O}(n^{t+2})$, assuming we let t grow at least logarithmically in n. The other reason is that for posets whose cover graph is a tree ($t = 1$), Atkinson's [5] algorithm takes—at least seemingly—a different approach and runs in time $\tilde{O}(n^3)$. Furthermore, Atkinson's algorithm is monotonic in the sense that all arithmetic operations are carried out with nonnegative numbers. This is in sharp contrast to both the present algorithm and the inclusion–exclusion algorithm, which crucially rely on a richer algebraic structure.

Our empirical study confirmed that the improvement in the asymptotic bound consistently transfers to the running times measured in practice. That said, the observed speedup, for n around one hundred, was by one order of magnitude rather than two. This "leak" of efficiency can, at least in part, be explained by the present algorithm's higher sensitivity to the shape of the selected tree decomposition. Indeed, we observed that the best of 50 generated tree decompositions typically yields a 5- to 10-fold speedup in relation to an average (i.e., median) tree decomposition.

We also showed that there is an efficient way to select the best or close-to-best tree decomposition using a linear regression model that was fitted to a collected data set of instances along with the measured running times, following the machine learning method of Abseher et al. [2]. However, we observed that the performance of the regression method rapidly degraded as the treewidth t increases. This suggests that the general-purpose method may not suit well for the problem of counting linear extensions. A potential reason for suboptimal performance is that the default set of features [2] does not include perhaps the most informative quantity associated with a node x in a tree decomposition, namely the term n_x^k (or some variant of it), which combines the size k of the bag of x with the number of vertices in the subtree rooted at x. This issue could be addressed by extending the feature set accordingly, or, potentially, by using some nonlinear regression model. On the other hand, we did inspect how well a single feature can predict a well-performing tree decomposition. We observed that for $t = 2$ and $t = 3$ the average depth of join nodes alone yielded predictions that were almost as good as the predictions by the full regression model with all the features.

We also experimented with a simpler solution for selecting a good tree decomposition. We manually constructed a heuristic function that adds up estimated contributions of each tree decomposition node. For each node x, our estimate relied on three parameters: the above mentioned k and n_x and the number of possible permutations of the element in the bag of the node, $p_x \leq k!$. This cost function goes beyond the *f-cost* framework [9], in which the contribution of each node can only depend on the size of the bag. We observed that for $t = 2$ and $t = 3$ this heuristic yielded almost as good tree decompositions as the machine learning method. But for $t = 4$ neither this method was able to find tree decomposition substantially better than an average one.

Since the best tree decomposition was seen to yield an order-of-magninute shorter running times than the median tree decomposition, it remains as an obvious challenge for future research to construct or learn a more accurate cost function. That being said, it has to be admitted that the presented tree decomposition based algorithm is practical only for very small treewidth, $t \leq 3$: for $t = 4$ already, a quite different, worst-case exponential-time algorithm is expected to be faster in practice [18, Fig. 8]; that algorithm is generally the faster, the denser the poset (i.e., the cover graph) is.

Acknowledgements Open access funding provided by University of Helsinki including Helsinki University Central Hospital. We thank the anonymous reviewers for constructive suggestions, which helped us improve the presentation. We also thank Teemu Hankala and Teppo Niinimäki for valuable discussions about some preliminary versions of the ideas that led to the present results.

Appendix: Proof of Fact 2

We will need the following notation. We consider polynomials in variables $z_1, \ldots, z_k$ with integer coefficients. We index a monomial by the tuple of its degrees $e = (e_1, \ldots, e_k)$. The *support* of a polynomial P is the set of tuples e for which the coefficient P_e of the corresponding monomial in P is non-zero. We denote by d_P the size of the smallest Cartesian product of the form $\times_{i=1}^{k}\{0, 1, \ldots, n_i\}$ that contains the support of P. Furthermore, we denote $h_P := \max_e l_{P_e}$, where for any integer i we write $l_i := \lceil \log_2(|i| + 1) \rceil = \tilde{O}(\log |i|)$ for its *bit-size*.

Finally, we let $\mu(l)$ denote the number of bit-operations needed to multiply two integers of bit-size at most l; we have that $\mu(l) = O\big(l(\log l)2^{\log^* l}\big)$ [15], which is $\tilde{O}(l)$.

Proposition 1 (Cor. 1 of van der Hoeven and Lecerf [33]) *Given $P, Q \in \mathbb{Z}[z_1, \ldots, z_k]$, we can compute $R = PQ$ with*

$$O\big(\mu(h\, d_R) + k\mu(\log d_R) + (d_P + d_Q)\log d_R\big) \tag{1}$$

bit-operations, where $h := h_P + h_Q + l_{\min\{d_P, d_Q\}}$.

We apply this result to two polynomials $P, Q \in \mathbb{Z}[z_1, \ldots, z_k]$ with $\tilde{O}(n)$-bit integer coefficients, $n \geq k$, and maximum degrees at most n_P and n_Q such that $n_P + n_Q \leq n$. Clearly the maximum degree of $R = PQ$ is at most n. Thus we have that $d_P \leq (n_P + 1)^k$, $d_Q \leq (n_Q + 1)^k$, and $d_R \leq (n + 1)^k$. Consequently, $h = \tilde{O}(n)$.

By Proposition 1 we get that R can be computed with

$$\tilde{O}\big(h(n + 1)^k + k^2 \log(n + 1) + (n + 1)^k\big) = \tilde{O}\big(n^{k+1}\big)$$

bit-operations. This completes the proof of Fact 2.

References

1. Abseher, M., Musliu, N., Woltran, S.: HTD: A free, open-source framework for (customized) tree decompositions and beyond. In: Proceedings of the 14th International Conference on Integration of AI and OR Techniques in Constraint Programming, volume 10335 of Lecture Notes in Computer Science, pp. 376–386. Springer, Berlin (2017)
2. Abseher, M., Musliu, N., Woltran, S.: Improving the efficiency of dynamic programming on tree decompositions via machine learning. J. Artif. Intell. Res. **58**, 829–858 (2017)
3. Aho, A.V., Garey, M.R., Ullman, J.D.: The transitive reduction of a directed graph. SIAM J. Comput. **1**(2), 131–137 (1972)
4. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebr. Discr. **8**(2), 277–284 (1987)
5. Atkinson, M.D.: On computing the number of linear extensions of a tree. Order **7**(1), 23–25 (1990)

6. Banks, J., Garrabrant, S., Huber, M.L., Perizzolo, A.: Using TPA to count linear extensions. arXiv preprint arXiv:1010.4981 (2017)
7. Bertelè, U., Brioschi, F.: Nonserial Dynamic Programming. Academic Press, New York (1972)
8. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25**(6), 1305–1317 (1996)
9. Bodlaender, H.L., Fomin, F.V.: Tree decompositions with small cost. Discrete Appl. Math. **145**(2), 143–154 (2005)
10. Bodlaender, H.L., Kloks, T.: Efficient and constructive algorithms for the pathwidth and treewidth of graphs. J. Algorithms **21**(2), 358–402 (1996)
11. Brightwell, G., Winkler, P.: Counting linear extensions. Order **8**(3), 225–242 (1991)
12. Dechter, R.: Bucket elimination: a unifying framework for reasoning. Artif. Intell. **113**(1–2), 41–85 (1999)
13. Dyer, M., Frieze, A., Kannan, R.: A random polynomial-time algorithm for approximating the volume of convex bodies. J. ACM **38**(1), 1–17 (1991)
14. Eiben, E., Ganian, R., Kangas, K., Ordyniak, S.: Counting linear extensions: parameterizations by treewidth. Algorithmica **81**(4), 1657–1683 (2019)
15. Fürer, M.: Faster integer multiplication. SIAM J. Comput. **39**(3), 979–1005 (2009)
16. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD Explor. Newsl. **11**(1), 10–18 (2009)
17. Hart, W.B.: Fast library for number theory: an introduction. In: Proceedings of the Third International Congress on Mathematical Software, pp. 88–91. Springer, Berlin (2010). `http://flintlib.org`
18. Kangas, K., Hankala, T., Niinimäki, T., Koivisto, M.: Counting linear extensions of sparse posets. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence, pp. 603–609. IJCAI/AAAI Press (2016)
19. Kangas, K., Koivisto, M., Salonen, S.: A faster tree-decomposition based algorithm for counting linear extensions. In: Proceedings of the 13th International Symposium on Parameterized and Exact Computation, IPEC 2018 (2018, to appear)
20. Karp, R.M.: Dynamic programming meets the principle of inclusion and exclusion. Oper. Res. Lett. **1**(2), 49–51 (1982)
21. Koivisto, M.: An $O^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion–exclusion. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, pp. 583–590. IEEE Computer Society (2006)
22. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Empirical hardness models: methodology and a case study on combinatorial auctions. J. ACM **56**(4), 22 (2009)
23. Lukasiewicz, T., Martinez, M.V., Simari, G.I.: Probabilistic preference logic networks. In: Proceedings of the 21st European Conference on Artificial Intelligence, Volume 263 of Frontiers in Artificial Intelligence and Applications, pp. 561–566. IOS Press (2014)
24. Mannila, H., Meek, C.: Global partial orders from sequential data. In: Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining, pp. 161–168. ACM (2000)
25. Möhring, R.H.: Computationally tractable classes of ordered sets. In: Rival, I. (ed.) Algorithms and Order, pp. 105–193. Kluwer Academic Publishers, Dordrecht (1989)
26. Morton, J., Pachter, L., Shiu, A., Sturmfels, B., Wienand, O.: Convex rank tests and semigraphoids. SIAM J. Discrete Math. **23**(3), 1117–1134 (2009)
27. Muise, C.J., Beck, J.C., McIlraith, S.A.: Optimal partial-order plan relaxation via MaxSAT. J. Artif. Intell. Res. **57**, 113–149 (2016)
28. Niinimäki, T., Parviainen, P., Koivisto, M.: Structure discovery in Bayesian networks by sampling partial orders. J. Mach. Learn. Res. **17**, 57:1–57:47 (2016)
29. Peczarski, M.: New results in minimum-comparison sorting. Algorithmica **40**(2), 133–145 (2004)
30. Rice, J.R.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)
31. Ryser, H.J.: Combinatorial Mathematics, Volume 14 of Carus Mathematical Monographs. The Mathematical Association of America, Washington (1963)
32. Talvitie, T., Kangas, K., Niinimäki, T., Koivisto, M.: Counting linear extensions in practice: MCMC versus exponential Monte Carlo. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence (2018)
33. van der Hoeven, J., Lecerf, G.: On the bit-complexity of sparse polynomial and series multiplication. J. Symb. Comput. **50**, 227–254 (2013)

34. Wallace, C.S., Korb, K.B., Dai, H.: Causal discovery via MML. In: Proceedings of the 13th International Conference on Machine Learning, pp. 516–524. Morgan Kaufmann (1996)