Journal of Artificial Intelligence Research 81 (2024) 643-681 Submitted 04/2024; published 11/2024

Approximate Counting of Linear Extensions in Practice

Topi Talvitie TOPI.TALVITIE@HELSINKI.FI
Mikko Koivisto MIKKO.KOIVISTO@HELSINKI.FI
Department of Computer Science
University of Helsinki, Finland

Abstract

We investigate the problem of computing the number of linear extensions of a given partial order on n elements. The problem has applications in numerous areas, such as sorting, planning, and learning graphical models. The problem is #P-hard but admits fully polynomial-time approximation schemes. However, the polynomial complexity bounds of the known schemes involve high degrees and large constant factors, rendering the schemes only feasible when n is some dozens. We present novel schemes, which stem from the idea of not requiring provable polynomial worst-case running time bounds. Using various new algorithmic techniques and implementation optimizations, we discover schemes that yield speedups by several orders of magnitude, enabling accurate approximations even when n is in several hundreds.

1. Introduction

Counting problems ask the number of objects in an implicitly given set, each object possibly equipped with a multiplicity or weight. The interest of artificial intelligence researchers in counting problems stems not only from their numerous applications in AI methods, but also from the computational challenge they tend to present. Indeed, various important counting problems have proven presumably computationally intractable, #P-hard (Valiant, 1979). Moreover, fully polynomial-time randomized approximation schemes, *fprases*, either are unlikely to exist (e.g., the decision variant is NP-hard) or have proven difficult to discover or prohibitively slow in practice. As a remedy, a branch of AI research has turned its attention to *heuristic* methods for solving "practical instances" of hard counting problems: instead of worrying about the worst-case asymptotic complexity, the interest is in algorithms that often terminate sufficiently fast and return either the exact count or an approximation along with a guaranteed error bound; for examples of previous works in this direction, we refer to Gomes et al. (2006), Thurley (2006), Chakraborty et al. (2013), Ermon et al. (2013), Kuck et al. (2019a, 2019b), Pavan et al. (2023).

When aiming at high-performing heuristics, one has to consider trade-offs between generality and efficiency. If an algorithm can, in principle, solve arbitrary instances of a very expressive problem—such as Boolean satisfiability or integer linear programming, or their counting variants—then on some classes of instances the algorithm will be outperformed by specialized algorithms tailored for those instance classes. Oftentimes, those classes correspond to well established prototypic hard problems, such as maximum clique, graph coloring, travelling salesman, network reliability, or matrix permanent, which are important in their own right. Creating novel heuristics for prototypic problems is a way to advance AI research: Not only particular problems will be solved more efficiently, but the discoveries

may also put forward ideas and techniques the can be adopted more generally. Recent examples are the adaptive partitioning technique (Kuck et al., 2019a) and the deep rejection sampling method (Harviainen et al., 2021) introduced in the context of estimating the matrix permanent.

In this paper, we investigate a representative counting problem: computing the number of linear extensions of a given partial order on n elements; an equivalent problem is to count the topological orderings of a given acyclic digraph. The problem has applications in various areas, ranging from sorting (Peczarski, 2004) and planning (Muise et al., 2016) to convex rank tests (Morton et al., 2009), discovery of sequential patterns (Mannila & Meek, 2000), and learning graphical models (Wallace et al., 1996; Niinimäki et al., 2016). The complexity status of the problem remained a recognized open problem until Brightwell and Winkler (1991) proved #P-completeness. Before that, a fpras had been discovered as a special case of approximating the volume of convex bodies (Dyer et al., 1991). Brightwell and Winkler also gave a faster fpras running in time $O(\varepsilon^{-2} n^9 \log^6 n)$ for any relative error $\varepsilon \leq 1$ and fixed confidence. This result relies on approximately uniform sampling of linear extensions along a rapidly mixing Markov chain due to Karzanov and Khachiyan (1991). A series of later works (Bubley & Dyer, 1999; Wilson, 2004; Banks et al., 2018) have managed to improve the bound to $O(\varepsilon^{-2} n^5 \log^3 n)$.

Despite the progress in worst-case asymptotic complexity bounds, the practical value of the fprases has remained unclear. Concerning the practical feasibility, the rapid growth of the polynomial bound is discouraging. Indeed, one can expect a relatively simple *exact exponential* algorithm to run faster in the feasible range of n, even though its worst-case bound is $O(2^n n)$ (De Loof et al., 2006); for example, if $\varepsilon = 0.05$ and $n = 30$, we have $\varepsilon^{-2} n^5 \ln^3 n \approx 4 \times 10^{11}$, while $2^n n \approx 3 \times 10^{10}$. Furthermore, the state-of-the-art exact algorithms exploit the particular structure of a given problem instance and often run substantially faster, being feasible up to around $n = 60$ in practice (Kangas et al., 2016). Moreover, allowing approximation, we may expect variants of exponential algorithms to scale up even further. This raises the main question we study in this paper: *When the interest is in the practical performance of counting linear extensions, is it better to implement a fpras or an exponential-time scheme?*

We will show that the best performance is, in fact, obtained by a hybrid that is an offspring of both a polynomial-time and an exponential-time scheme, combining approximation schemes and exact computations. For a proper comparison of existing schemes, we classify them into three approaches and present algorithmic enhancements for each, aiming at schemes that are fast in practice. First, in Section 3, we introduce an approximation scheme that exploits the exact exponential time algorithms of Kangas et al. (2016). The idea is to use rejection sampling: we sample linear extensions of a "simpler" partial order obtained as an appropriate relaxation of the original input. We call the scheme *adaptive relaxation Monte Carlo (ARMC)*. Second, in Section 4, we revisit the *telescoping product estimator* by Brightwell and Winkler (1991). We give an improved construction that yields, not only speedups in practice, but also a factor of $\log n$ reduction in the total worst-case time complexity, and thereby asymptotically the fastest known scheme we are aware of, running in time $O(\varepsilon^{-2} n^5 \log^2 n)$. Third, in Section 5, we revisit the *Tootsie Pop Algorithm* (TPA), which can be viewed as a continuous version of the generic, discrete telescoping product estimator by Huber and Schott (2010) and Banks et al. (2018). We give an im-

provement to the generic TPA, as well as a novel instantiation of it to approximate counting of linear extensions. Furthermore, we give efficient implementations using modern hardware equipped with multiple cores, vector instructions, and a graphical processing unit (GPU). Our code is available at `https://github.com/ttalvitie/linext`.

Parts of this work have been published in preliminary form (Talvitie et al., 2018a, 2018b). The main technical extensions here are the enhancement of the general TPA and the modifications needed for efficient vectorization and parallelization. We have also omitted some material: we exclude the comparison to schemes that employ SAT solvers, for these schemes appeared to be uncompetitive in our preliminary study (Talvitie et al., 2018b).

The remainder of this article is organized as follows. Section 2 introduces the key concepts related to partial orders and some basic facts about sampling methods we will frequently need in later sections. After presenting the schemes in Sections 3–5, we report on experimental results in Section 6. We conclude in Section 7 by discussing the lessons learned and directions for future research.

2. Preliminaries

We review some basic concepts and results on partially ordered sets, randomized approximation methods, and Markov chains. The concepts and the introduced notation will be frequently used in the later section. This section does not present new results.

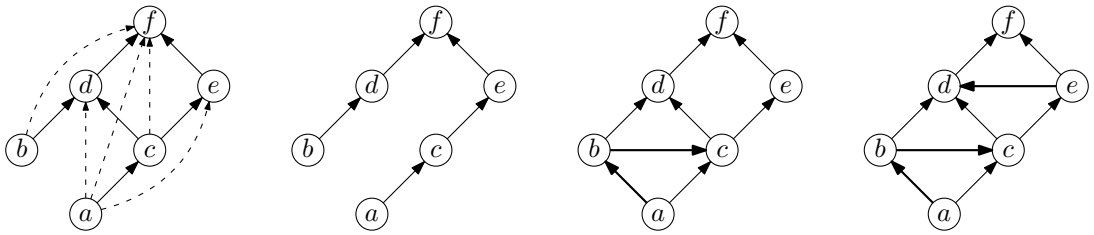
2.1 Partial Orders, Relaxations, Extensions, and Other Basic Concepts

Let V be a finite set and $\prec$ a binary relation on V. We call $\prec$ a *partial order* on V if it is irreflexive and transitive, and a *linear order* on V if in addition every two distinct elements $a, b \in V$ are *comparable*, i.e., $a \prec b$ or $b \prec a$. If $a \prec b$ we say that a *precedes* b. If $\prec'$ is another partial order on V that is a superset (resp. subset) of $\prec$, then $\prec'$ is an *extension* (resp. *relaxation*) of $\prec$. A maximal extension is called a *linear extension*, for it is a linear order. The unique minimal relaxation is the empty relation.

We call the pair $P = (V, \prec)$ a *partially ordered set*, or *poset*, and V its *ground set*. Any subset $S \subseteq V$ *induces* a poset $P[S] := (S, \prec_S)$, where $\prec_S$ is the restriction of $\prec$ to S, that is, $\prec_S$ is the intersection of $\prec$ and $S \times S$. By an extension or relaxation of a poset we refer to the poset obtained by equipping the ground set with an extension or relaxation of the partial order relation. The *cover graph* of $P = (V, \prec)$ is the directed graph (V, E) obtained as the transitive reduction of the graph $(V, \prec)$, that is, $(a, b) \in E$ exactly when $a \prec b$ and there is no $c \in U$ such that $a \prec c \prec b$.

We denote by $\mathcal{L}(P)$ the set of linear extensions of P, and by $\ell(P) := |\mathcal{L}(P)|$ their number. For an induced subposet $P[S]$, we may write simply $\ell(S)$ instead $\ell(P[S])$ when the underlying partial order is clear in the context.

There are two frequently occurring ways to combine two smaller posets into a larger poset. Let Q and R be two posets on disjoint ground sets A and B, respectively. Consider a poset $P = (A \cup B, \prec)$ such that $P[A] = Q$ and $P[B] = R$. We say that P is the *series composition* of Q and R if the elements in A precede the elements in B; observe that then $\ell(P) = \ell(Q)\ell(R)$. Similarly, P is the *parallel composition* of Q and R if the elements in A



(a) poset (b) relaxation (c) extension (d) linear extension

Figure 1: We visualize partial orders $\preceq$ as DAGs where each edge $x \to y$ indicates the ordering constraint $x \prec y$ in the partial order. We often omit the edges that follow by transitivity. In the partial order of (a), these edges are shown as dashed lines. (b) is a relaxation of (a), as it is obtained by removing the constraints $c \preceq d$ and $a \preceq d$ from the partial order relation. The extension (c) is obtained from (a) by adding the constraints $a \preceq b$ and $b \preceq c$; we also need to add $b \preceq e$ to satisfy transitivity. After adding the constraint $e \preceq d$ to the relation, we get a linear extension (d) of the poset (a) corresponding to the ordered list (a, b, c, e, d, f). This is one of the 7 linear extensions of the poset (a); the other six are (a, b, c, d, e, f), (a, c, b, d, e, f), (a, c, b, e, d, f), (a, c, e, b, d, f), (b, a, c, d, e, f), and (b, a, c, e, d, f).

are incomparable with the elements in B; now $\ell(P) = \binom{|V|}{|A|}\ell(Q)\ell(R)$, since interleaving any linear extensions of Q and R results in a linear extension of P.

Figure 1 illustrates the key concepts.

2.2 Basics of Randomized Approximations

Let ε and δ be nonnegative reals. Call a random variable Z an (ε, δ)-*approximation* of a real $r \neq 0$ if Z is within the factor $1 + \varepsilon$ of r with probability at least $1 - \delta$, that is,

$$\Pr\big(1/(1+\varepsilon) \leq Z/r \leq 1+\varepsilon\big) \geq 1 - \delta\,.$$

Observe that then, with probability at least $1 - \delta$, we also have $|Z/r - 1| \leq \varepsilon$, bounding the relative error, and $|\ln(Z/r)| \leq \varepsilon$, bounding the absolute error of logarithms, which are two slightly weaker conditions sometimes taken as the defining conditions; cf., e.g., Mitzenmacher and Upfal (2005, Def. 10.1) and Jerrum, Sinclair, and Vigoda (2004).

Thanks to the symmetry in the definition, we have the following.

Lemma 1 (Multiplicative inverse)**.** *Let Z be an (ε, δ)-approximation of r. Then $1/Z$ is an (ε, δ)-approximation of $1/r$.*

Since $1/(1+\varepsilon) \leq 1 - \varepsilon_0$ exactly when $\varepsilon_0 \leq \varepsilon/(1+\varepsilon)$, we have the following implication.

Lemma 2 (Relative error)**.** *Suppose $\Pr\big(|Z/r - 1| \leq \varepsilon/(1+\varepsilon)\big) \geq 1 - \delta$. Then Z is an (ε, δ)-approximation of r.*

Chernoff bounds are a powerful tool for showing that a random variable concentrates around its mean. Bounds for binomial variables are among the most frequently used; for a proof, see, e.g., Mitzenmacher and Upfal (2005).

Lemma 3 (Binomial tails)**.** *Let Z be a binomial random variable with mean $\mu > 0$. Let $a \geq 0$. Then $\Pr(Z \geq \mu + a) \leq \exp\{-a^2/(2\mu + a)\}$ and $\Pr(Z \leq \mu - a) \leq \exp\{-a^2/(2\mu)\}$.*

Similar concentration bounds are known for the Poisson distribution, considered as folklore; for a short proof, we refer to Banks et al. (2018, Lemma 8).

Lemma 4 (Poisson tails)**.** *Let Z be a Poisson random variable with mean $\lambda > 0$. Let $a \geq 0$. Then $\Pr(Z \geq \lambda + a) \leq \exp\{-a^2/(2\lambda + 2a)\}$ and $\Pr(Z \leq \lambda - a) \leq \exp\{-a^2/(2\lambda)\}$.*

Sometimes it is useful to boost a constant success probability of an estimate by taking a median of multiple independent copies (Jerrum, Valiant, & Vazirani, 1986).

Lemma 5 (The median trick)**.** *Let $m \geq 12 \ln \delta^{-1}$. Let $Z_1, Z_2, \ldots, Z_m$ be independent $(\varepsilon, 1/4)$-approximations of r and Y their median. Then Y is an (ε, δ)-approximation of r.*

Proof. The median is too small or large only if at least $m/2$ of the variables Z_i are. By Lemma 3, for a $\mathrm{Bin}(m, 1/4)$ variable, this occurs with probability at most $\mathrm{e}^{-m/12} \leq \delta$. □

How many independent samples of a $\{0,1\}$-valued random variable are needed for obtaining an (ε, δ)-approximation of its mean p? By applying Lemma 3 for the sample average, we get that $m := \lceil 2p^{-1}(1+\varepsilon)\varepsilon^{-2} \ln(2/\delta) \rceil$ samples will suffice. This result, however, cannot be used directly, as the mean p is supposed to be unknown. Dagum, Karp, Luby, and Ross (2000) were the first to show how to efficiently circumvent this obstacle:

Theorem 6 (Optimal Monte Carlo)**.** *Let $X_1, X_2, \ldots$ be independent $\{0,1\}$-valued random variables, each with mean p. For any positive integer m, let $S_m = X_1 + X_2 + \cdots + X_m$. Let $\varepsilon \in (0,1)$ and $\delta > 0$. Let T be the smallest integer such that*

$$S_T = m_{\varepsilon,\delta} := \lceil 1 + 4(\mathrm{e} - 2)(1+\varepsilon)\varepsilon^{-2} \ln(2/\delta) \rceil .$$

Then S_T/T is an (ε, δ)-approximation of p and the expected value of T is at most S_T/p.

This bound on the required sample size is within a constant of the optimum (Dagum et al., 2000). Gajek, Niemiro, and Pokarowski (2013) improve the leading constant factor of the upper bound from $4(\mathrm{e} - 2) \approx 2.873$ to 2, with an asymptotically matching lower bound. Huber (2017) presents the Gamma Bernoulli approximation scheme (GBAS), in which the sum S_T is not divided by the negative binomial random variable T, but by a Gamma random variable; this estimator also achieves the leading factor of 2, along with certain other desirable properties.

2.3 Basics of Markov Chains

A sequence of random variables $X_0, X_1, \ldots$ taking values in a set Ω is a *Markov chain* if X_t is conditionally independent of $\{X_j : j < t\}$ given X_{t-1}, for all $t \geq 1$. We will only consider *homogeneous* Markov chains, in which the conditional distribution of X_t given X_{t-1} is the same for all t. If this *transition kernel* leaves a distribution π invariant (i.e., $X_{t-1} \sim \pi$ implies $X_t \sim \pi$), then π is called a *stationary distribution* of the chain. Let p_x^t denote the distribution of X_t when $X_0 = x$. If the chain is finite, irreducible, and aperiodic, then p_x^t converges to a unique stationary distribution π as t grows. On a continuous state

space, a sufficient condition for convergence to π (π-almost surely w.r.t. the initial state) is that the transition probability density, $p(x, y)$, is *reversible* with respect to π, that is, $\pi(x)p(x, y) = \pi(y)p(y, x)$ for all $x, y \in \Omega$ (Roberts & Rosenthal, 2004). The *mixing time* of the chain is the smallest t for which p_x^t is at *variational distance* at most $1/4$ from π, regardless of the initial state x; the variational distance is the supremum of $|p_x^t(B) - \pi(B)|$ over all events B. The constant $1/4$ here is sufficient for characterizing the convergence time, since the distance decays below any $\varepsilon > 0$ as soon as t exceeds the mixing time by the factor $\lceil \log_2 \varepsilon^{-1} \rceil$ (Levin & Peres, 2017, Section 4.5).

Sometimes one can turn an approximate Markov chain sampler to an *exact sampler* (or *perfect sampler*) by using a technique called *coupling from the past* (CFTP) (Propp & Wilson, 1996). In general this requires that we can construct a *bounding chain* for the Markov chain. A bounding chain is a Markov chain itself that "bounds" in which states the original Markov chain can be. In the beginning of the simulation the bounding chain allows all states. Once the bounding chain has converged to bound only one state, we know it must be the current state of the original Markov chain, regardless of the starting state. Thus, if the simulation started sufficiently far from the past, the chain will converge with high probability. What is sufficiently far, can be found by trying repeatedly further from the past (using the same random numbers at the same time points). Discovering a bounding chain that converges quickly can be challenging. On the other hand, typically a bounding chain has the desirable feature that the required number of simulation steps varies depending on the actual underlying problem instance, being in practice much smaller than any known worst-case upper bound; this is in sharp contrast to approximate Markov chain samplers.

In Section 5, we will use *monotone coupling from the past*, which constructs a bounding chain using a monotonicity property (Propp & Wilson, 1996). Let ϕ a deterministic *transition function* such that $X_t = \phi(X_{t-1}, U_t)$, where the U_t are independent random variables distributed uniformly in $[0, 1]$. We say that the stationary distribution of ϕ is the stationary distribution of the Markov chain $X_0, X_1, \ldots$ (with any initial distribution of X_0). Supposing the state space Ω admits a partial ordering $\prec$ such that $x \preceq y$ implies $\phi(x, U) \leq \phi(y, U)$ almost surely with respect to U, then ϕ is called *monotone* in relation to $\prec$ and the Markov chain is called *monotonic*. If the poset $(\Omega, \prec)$ has a unique minimal element and a unique minimal element, then Algorithm 1 can be employed to draw an exact sample from the stationary distribution (Propp & Wilson, 1996, Theorem 1). While the original treatment of Propp and Wilson (1996) assumes that the state space is finite, it is not difficult to see that the algorithm works correctly whenever the chains started from the minimal and the maximal element coalesce with a positive probability (i.e., $L = R$ on line 7; these letters refer to *left* and *right*, or lower and upper, and should not be confused with linear extensions and relaxations). For Markov chains on uncountable state spaces this property is generally challenging to establish; however, we will do exactly this in our special case (Theorem 14).

3. Adaptive Relaxation Monte Carlo

This section presents the *adaptive relaxation Monte Carlo (ARMC)* scheme for approximate counting of the linear extensions of a given poset. ARMC relies on a dynamic programming algorithm for exact counting due to Kangas et al. (2016), which also enables exact uniform sampling of linear extensions; we begin by describing this building block in Section 3.1.

Algorithm 1 Monotone coupling from the past

Input: A transition function ϕ that is monotone in relation to a partial order with unique minimal element $\mathbf{0}$ and maximal element $\mathbf{1}$, sequence of independent uniform random variables $U_t \in [0,1]$ for $t = -1, -2, \ldots$.

Output: A sample from the stationary distribution of ϕ.

1. $T \leftarrow 1$.
2. **repeat**
3. $\quad L \leftarrow \mathbf{0}$, $R \leftarrow \mathbf{1}$
4. $\quad$ **for** $t = -T$ **to** -1 **do**
5. $\quad\quad L \leftarrow \phi(L, U_t)$, $R \leftarrow \phi(R, U_t)$
6. $\quad T \leftarrow 2T$
7. **until** $L = R$
8. **return** L

The idea of ARMC is to find a relaxation of the poset so as to balance the time spent, on the one hand, by counting the linear extensions of the relaxation, and on the other hand, by sampling sufficiently many linear extensions to yield an accurate Monte Carlo estimator of the linear extensions of the original poset; we describe the Monte Carlo estimator in Section 3.2 and our heuristic algorithm for finding a good relaxation in Section 3.3.

3.1 Exact Counting and Sampling by Dynamic Programming

The algorithm of De Loof et al. (2006) counts the linear extensions by dynamic programming (DP) based on the observation that each linear extension of poset P is obtained by choosing a minimal element a of P as the first element, followed by an arbitrary linear extension of the rest of the poset $P[V \setminus \{a\}]$. This gives a recurrence over non-empty subsets S of V:

$$\ell(S) = \sum_{a \in \min P[S]} \ell(S \setminus \{a\}) \,, \tag{1}$$

with the boundary case $\ell(\emptyset) = 1$. Thus, the linear extensions of an n-element poset can be counted with $O(2^n n)$ basic operations. A more refined bound is obtained by observing that the recurrence is, in fact, only over the *upsets* of P, that is, sets S with the property that $a \in S$ and $a \prec b$ imply $b \in S$: for posets with u upsets and width w, the linear extensions can be counted with $O(uw)$ basic operations (De Loof et al., 2006). By Dilworth's theorem (Dilworth, 1950), $u \leq (n/w + 1)^w = O(n^w)$.

Kangas et al. (2016) refined the algorithm by observing that it suffices to only consider upsets that are connected: if S can be partitioned into disjoint sets A and B such that no element in A is comparable with an element in B, then each linear extension of $P[S]$ is obtained by arbitrarily interleaving some linear extensions of $P[A]$ and $P[B]$. Consequently,

$$\ell(S) = \binom{|S|}{|A|} \ell(A)\ell(B) \,. \tag{2}$$

Our implementation uses memoization, starting from the ground set V and storing the values $\ell(S)$ in a hash table for connected upsets S.

Once the DP algorithm has finished, it also enables straightforward sampling of linear extensions from the uniform distribution: If S is connected, the first element a of a linear extension is drawn from the minimal elements of $P[S]$, with the probability $\ell(S \setminus \{a\})/\ell(S)$, and the order on the remaining elements is sampled by recursing on $S \setminus \{a\}$. If S is not connected, a linear extension is sampled recursively for each connected component and the sampled extensions are then interleaved uniformly at random. In this manner a single sample can be drawn in $O(n^2)$ time.

3.2 Relaxation Monte Carlo

Given a poset P, the basic idea of ARMC is to relax P by removing ordering constraints until the counting problem becomes feasible for the exact DP algorithm. We then estimate the error introduced by the removal of constraints using Monte Carlo. Specifically, let R be a relaxation of P, and let $\mu = \ell(P)/\ell(R)$ be the probability that a linear order sampled from $\mathcal{L}(R)$ uniformly at random is also in $\mathcal{L}(P)$. To approximate $\ell(P)$, we first use the DP algorithm to compute $\ell(R)$ and then sample m linear orders from $\mathcal{L}(R)$ to compute an estimate

$$\widehat{\mu} := \frac{1}{m} \sum_{i=1}^{m} Z_i \,,$$

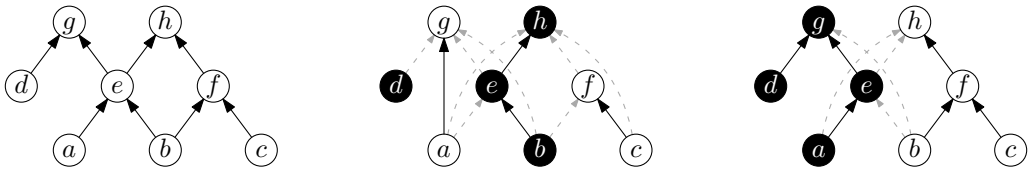
where $Z_i = 1$ if the ith sample is in $\mathcal{L}(P)$ and $Z_i = 0$ otherwise. As m grows, $\widehat{\mu}$ concentrates around μ and thus $\widehat{\mu} \cdot \ell(R)$ concentrates around $\ell(P)$. Specifically, using the optimal Monte Carlo method (Theorem 6) we obtain an (ε, δ)-approximation by sampling from $\mathcal{L}(R)$ until $m_{\varepsilon,\delta} = O(\varepsilon^{-2} \log \delta^{-1})$ of the samples have fallen in $\mathcal{L}(P)$. The smaller the fraction μ is, the more samples we need for an accurate approximation. The efficiency of this scheme thus depends crucially on how the relaxation R is chosen, as it determines both μ and the time required to run the DP algorithm.

Here, we consider *partition relaxations*. We obtain a partition relaxation $R = (V, \prec')$ by (arbitrarily) partitioning the ground set U into disjoint sets $V_1, V_2, \ldots, V_{\lceil n/k \rceil}$ of some appropriate size k (the last set potentially being smaller), and letting $a \prec' b$ if and only if $a \prec b$ and a and b belong to same set. In other words, we remove all ordering constraints between sets while keeping all constraints within the sets. Now, to compute $\ell(R)$, the DP algorithm will immediately apply the decomposition rule (2) and then solve each of the $\lceil n/k \rceil$ subproblems in time $O(2^k k^2)$, yielding in total a time requirement of $O(2^k kn)$. The parameter k controls the tradeoff: increasing k causes the DP phase to take longer, but typically brings R closer to P and thus saves time in the sampling phase.

Before we proceed to devising a procedure for finding a good partition relaxation, let us summarize the worst-case complexity of the scheme for a given partition relaxation.

Proposition 7. *Let R be a partition relaxation of P with parts of size k. Let $\mu = \ell(P)/\ell(R)$. The relaxation Monte Carlo scheme runs in expected time $O(2^k kn + n^2 \mu^{-1} \varepsilon^{-2} \log \delta^{-1})$.*

For a concrete illustration of this result, consider an arbitrary partition relaxation R with two parts. The size of the larger part is thus $k \geq n/2$. Since there are exactly $\binom{n}{k}$ ways to interleave two linear orders on k and $n-k$ elements, we have that $\mu^{-1} \leq \binom{n}{k}$. Now, putting $k = \lfloor 0.77n \rfloor$, we find that $\binom{n}{k} \leq 2^k$ and get that the relaxation Monte Carlo runs in



(a) poset (b) initial relaxation (c) optimal relaxation

Figure 2: To find the relaxation to use for the poset in (a) with parameter $k = 4$, we start from an arbitrarily chosen initial partition $V_1 = \{b, d, e, h\}$ and $V_2 = \{a, c, f, g\}$ of sets of size k. The relaxation is obtained by removing all the ordering constraints between different sets of the partition, which are shown in (b) as dashed arrows. Note that we have to add the constraint arrow $a \to g$ to the graphical representation as it no longer follows by transitivity. Initially, the heuristic value, given by the number of removed constraints, is 8. The greedy local search swaps g and h, improving the heuristic to 5, and then a and b, further improving the heuristic to 4. This leads to the case in (c), which cannot be improved further. By minimizing the heuristic value, we have successfully reduced the number of linear extensions of the relaxation from 1680 in (b) to 420 in (c).

expected time $O\big(2^{0.77n} n^2 \varepsilon^{-2} \log \delta^{-1}\big)$. Compare this to the bound $O(n^5 \log^3 n \varepsilon^{-2} \log \delta^{-1})$ of the fastest known fpras (prior to this work): supposing the hidden constant factors and additive lower order terms are approximately equal, the exponential bound is better as long as $2^{0.77n} < (n \log_2 n)^3$, which holds when $n \leq 27$. For moderate values of the accuracy parameters, say $\varepsilon = 1/2$ and $\delta = 1/4$, the exponential approximation scheme also beats the exact exponential algorithm in terms of the known worst-case running time upper bound.

3.3 Adaptive Relaxations

To complete the ARMC scheme, we give a heuristic to search for a good partition relaxation R. Suppose for a moment that the size k is fixed. Ideally, we wish to minimize $\ell(R)$. Since evaluating this objective function is still relatively expensive, we make use of an approximation and aim to minimize the number of removed ordering constraints $|\prec \setminus \prec'|$, or equivalently, maximize $|\prec'|$. Unfortunately, even with this simplified objective function the optimization problem is NP-hard, for it subsumes the minimum bisection problem (Garey, Johnson, & Stockmeyer, 1976). Instead of attempting an optimal solution, we resort to greedy hillclimbing that starts at a random partition and iteratively swaps pairs of elements between different sets that yield the largest decrease in the number of removed constraints. We have found this strategy to work well in practice, even if it may not converge to a global optimum. Figure 2 illustrates the greedy algorithm.

Consider then the remaining task of setting the parameter k. We propose the following procedure that adapts k based on the structure of the input poset. We start from a small value of k and increase it until the running times of the two phases of the algorithm are balanced. For each tested k, we find the relaxation, count its linear extensions using the DP algorithm, and finally estimate μ by sampling. When it becomes clear that the sampling

Algorithm 2 The ARMC scheme

Input: A poset P, reals $\varepsilon, \delta > 0$, integers $k_0, k_1 \geq 1$, reals $0 < \alpha \leq 1 < \beta$.
Output: An (ε, δ)-approximation of $\ell(P)$.

1. $k \leftarrow k_0 - k_1$
2. **repeat**
3. $\quad k \leftarrow k + k_1$ ▷ The first value of k is k_0
4. $\quad R \leftarrow$ a partition relaxation of P with parts of size k, obtained by greedy hillclimbing started at a random partition.
5. $\quad$ Run DP to obtain $\ell(R)$ and to enable efficient sampling from $\mathcal{L}(R)$
6. $\quad T_5 \leftarrow$ the time spent in step 5
7. $\quad$ Run sampling from $\mathcal{L}(R)$ tentatively for time $\alpha \cdot T_5$; obtain m' samples from $\mathcal{L}(P)$
8. $\quad T_7 \leftarrow$ the time spent in step 7
9. $\quad$ **if** $m' = 0$ **then**
10. $\quad\quad t_7 \leftarrow \infty$
11. $\quad$ **else**
12. $\quad\quad t_7 \leftarrow T_7 \cdot m_{\varepsilon,\delta}/m'$ ▷ Estimated time to run the sampling phase in full
13. $\quad t_5 \leftarrow \beta \cdot T_5$ ▷ Estimated time to run DP for $k + k_1$ parts
14. **until** $t_7 < t_5$
15. Run sampling from $\mathcal{L}(R)$ in full to obtain $m_{\varepsilon,\delta}$ samples from $\mathcal{L}(P)$
16. $m \leftarrow$ the number of samples from $\mathcal{L}(R)$
17. **return** $\ell(R) \cdot m_{\varepsilon,\delta}/m$

phase will take significantly more time than the DP algorithm, we give up and try a larger value of k instead. If we increment k in sufficiently large steps, the failed attempts will not introduce notable overhead, since the time complexity of each attempt grows exponentially in k. Algorithm 2 describes the above procedure in more detail and completes it to a scheme that outputs an estimate of $\ell(P)$.

3.4 Implementation

In the experiments, we set the user parameters as follows: $k_0 = 20, k_1 = 5$ and $\alpha = 0.1$, $\beta = 10$; we observed these values to perform well on average. As a practical optimization, we also keep the previous relaxation found in step 4 if it has fewer linear extensions than the new relaxation. We restart 50 times to get slightly better relaxations. Furthermore, if k becomes so large that the DP algorithm starts running out of memory, we proceed directly to the sampling phase with the best relaxation found so far. We parallelize the DP phase by running DP for each component of the relaxation in a different thread. We also parallelize the sampling phase into multiple threads, and construct each sample from $\mathcal{L}(R)$ iteratively such that we can in most cases detect that the sample is not in $\mathcal{L}(P)$ without having to completely construct it.

4. Telescoping Product Estimators

The relaxation Monte Carlo method, described in the previous section, is relatively ineffective if the relaxation is far away from the original poset, resulting in a worst-case exponential-time scheme. There is a known technique for improving this scheme: bridge the gap between the relaxation and the poset by a chain of relaxations where adjacent relaxations are close to each other. Then it suffices to estimate the ratio of the linear extension counts for any two adjacent relaxations, each estimate of which now requires a relatively small number of samples. The challenge is to efficiently sample random linear extensions of a relaxation that is close to the original poset and thus infeasible for the dynamic programming approach; here rapidly mixing Markov chains can come to rescue. This is an instantiation of a generic strategy for approximate counting of solutions to self-reducible combinatorial problems (Jerrum & Sinclair, 1997).

Brightwell and Winkler (1991) implemented this strategy for approximate counting of linear extensions, with one significant twist: they introduced a chain of *extensions* from the original poset to some linear extension, instead of a chain of relaxations to some simple relaxation. We next describe the original, basic version of their method (Section 4.2), and a way to enhance it, leading to the best worst-case bound we are aware of (Section 4.3). Both schemes assume the availability of an efficient sampler of linear extensions for a given poset—we begin (Section 4.1) with a review of the most relevant results in the literature.

4.1 Markov Chains on Linear Extensions

Karzanov and Khachiyan (1991) studied a simple Markov chain on linear extensions. The *Karzanov–Khachiyan chain* makes an attempt to swap the positions of two random adjacent elements in the current linear order; if the elements are comparable, the attempt is rejected and the chain stays in the current state. Karzanov and Khachiyan proved that the chain mixes in time $O(n^6 \log n)$, but later investigations have shown that the worst-case mixing time is within a constant factor of $n^3 \ln n$ (Bubley & Dyer, 1999; Wilson, 2004).

The *insertion chain* is another simple Markov chain. It removes a random element from the ordering and reinserts it to a random position. Bubley and Dyer (1999) give a lower bound $\Omega(n^2)$ and an upper bound $O(n^5 \log^2 n)$ for the worst-case mixing time. The *shuffle chain* (Talvitie, Niinimäki, & Koivisto, 2017) is a more complicated Markov chain. It selects a random interval of positions, finds the connected components of the poset induced by the elements in the interval, and randomly reorders the elements in the interval such that within each component the ordering is unchanged. The worst-case mixing time is $O(n^4 \log^2 n)$ (Talvitie et al., 2017, Prop. 1). The insertion chain and the shuffle chain may mix much faster in practice than what is guaranteed by the worst-case upper bounds (Talvitie et al., 2017). However, the unavailability of better bounds (or stopping rules) currently renders these chains inferior to the Karzanov–Khachiyan chain.

Huber (2006) gives a bounding chain for the Karzanov–Khachiyan chain, yielding a perfect sampler that draws linear extensions exactly from the uniform distribution. The idea is to keep track of an upper bound for the position of each element in the order. The expected running time of the resulting CFTP algorithm is $O(n^3 \log n)$. If implemented as described in the original article, the algorithm always takes the same number of steps for a fixed number of elements n (and a fixed output sequence of the random number

generator). A modification that initializes the bounding chain adaptively according to the input partial order can terminate after a smaller number of steps (Mark L. Huber, personal communication, August 12, 2016). We refer to this perfect sampling method as the *exact Karzanov–Khachiyan chain*.

Later Huber (2014) gave another type of bounding chain, which is related to the insertion chain but operates in a continuous state space, the unit n-cube $[0,1]^n$. Each point in this space (except a zero-measure subset) corresponds to a unique linear order on V determined by the ordinary order on reals. In particular, drawing a point from $\{x \in [0,1]^n : x_a \leq x_b \text{ whenever } a \prec b\}$ corresponds to drawing a linear extension of P. A straightforward Gibbs sampler makes a transition by drawing an element a from V and then resampling the coordinate x_a. This roughly corresponds to reinserting an element to a randomly selected location as in the insertion chain. This chain is monotonic and yields an efficient CFTP algorithm. Huber shows that the expected running time of the algorithm is $O(\Delta^2 n \log n)$ on height-2 partial orders where every element is comparable with at most Δ other elements; no bound is given for the general case.

4.2 The Basic Brightwell–Winkler Estimator

Brightwell and Winkler (1991) constructed an estimator of $\ell(P)$ as follows. Let $P_0, P_1, \ldots, P_k$ be a sequence of posets on the same ground set V such that P_i equals P if $i = 0$ and is otherwise obtained from P_{i-1} by adding some ordering constraint $a_i \prec b_i$ and those that follow by transitivity, ending in some linear order P_k. Since $\ell(P_k) = 1$, we can write $\mu := 1/\ell(P)$ as the telescoping product $\mu = \prod_{i=1}^{k} \mu_i$, where $\mu_i := \ell(P_i)/\ell(P_{i-1})$. Now, let $\widehat{\mu}_{i,m}$ be a zero–one Monte Carlo estimator of μ_i, that is, the proportion of members of $\mathcal{L}(P_i)$ in m independent samples drawn uniformly at random from $\mathcal{L}(P_{i-1})$. We get an estimator

$$\widehat{\mu}_m := \prod_{i=1}^{k} \widehat{\mu}_{i,m}$$

whose expected value is μ and whose variance decreases as m grows.

Proposition 8. *Let $\varepsilon, t \in (0,1]$. Let $\mu_i \geq t$ for $i = 1, 2, \ldots, k$ and $m \geq 5k\,(1-t)t^{-1}\varepsilon_0^{-2}$, where $\varepsilon_0 := \varepsilon/(1+\varepsilon)$. Then $\widehat{\mu}_m$ is an $(\varepsilon, 1/4)$-approximation of $1/\ell(P)$.*

Proof. Observe that $\widehat{\mu}_{i,m}$ is a random variable with the expected value μ_i and variance $\mu_i(1-\mu_i)/m$. By Chebyshev's inequality,

$$\varepsilon_0^2 \Pr(|\widehat{\mu}_m - \mu| > \varepsilon_0 \mu) \leq \frac{\mathbf{Var}(\widehat{\mu}_m)}{\mathbf{E}[\widehat{\mu}_m]^2} = \prod_{i=1}^{k} \left(1 + \frac{\mathbf{Var}(\widehat{\mu}_{i,m})}{\mathbf{E}[\widehat{\mu}_{i,m}]^2}\right) - 1 = \prod_{i=1}^{k} \left(1 + \frac{1-\mu_i}{m\mu_i}\right) - 1\,.$$

Now, using the lower bounds of μ_i and m, we get $(1-\mu_i)/(m\mu_i) \leq \varepsilon_0^2/(5k)$. Putting $z := \varepsilon_0^2/5 \leq 1/5$ and applying the inequality $(1 + z/k)^k \leq 1 + z + z^2 \leq 1 + (6/5)z$ gives $\Pr(|\widehat{\mu}_m/\mu - 1| > \varepsilon_0) \leq (6/5)(1/5) < 1/4$, whence the claim follows by Lemma 2. □

To bound the ratios μ_i from below by a positive constant, Brightwell and Winkler (1991) employ an iterative method. The next poset P_i is obtained from P_{i-1} by choosing a pair of incomparable elements (a, b) and adding one of the constraints $a \prec b$ and $b \prec a$ along

with the constraints that follow then by transitivity. Making the right choice among the two possible directions for the added constraint is critical, as one of them might yield a very small μ_i. Let P_i' and P_i'' denote the two possible outcome posets and μ_i' and μ_i'' the respective ratios. Clearly, μ_i' and μ_i'' add up to 1 and thus the larger of the two is at least $1/2$. Consequently, $O(\log \delta_0^{-1})$ samples of linear extensions of P_{i-1} allow us to estimate μ_i', say to within a relative error of $1/3$, and thereby decide the ordering so that $\mu_i \geq 1/3$ holds with probability at least $1 - \delta_0$. By setting $\delta_0 := \delta/k$ and using the union bound, we get that the lower bound holds for all $i = 1, 2, \ldots, k$ with probability at least $1 - \delta$.

A direct implementation of this idea guarantees that the constructed sequence of posets has length $k = O(n^2)$, as one can add at most $n(n-1)/2$ constraints before reaching a linear order. This bound can be improved by observing that the way the sequence of posets is constructed is analogous to comparison sorting (Brightwell & Winkler, 1991): deciding whether to add the constraint $a \prec b$ or $b \prec a$ corresponds to comparing the elements a and b. By emulating an $O(n \log n)$-time comparison sorting algorithm, we get that after $k = O(n \log n)$ steps, one linear extension of the original poset is singled out.

In summary, by Proposition 8 above and the median trick (Lemma 5), we obtain an (ε, δ)-approximation of $\ell(P)$ by drawing $O(k^2 \epsilon^{-2} \log \delta^{-1})$ linear extensions of certain posets on n elements, each draw taking an expected time of $O(n^3 \log n)$. The total expected running time is thus $O(\varepsilon^{-2} n^5 \log^3 n \log \delta^{-1})$. We postpone a more detailed proof to the next subsection where we further reduce the time bound by a factor of $\log n$.

4.3 An Enhanced Brightwell–Winkler Estimator

So far our analysis only used the fact that the ground set V of each considered poset P_i has n elements—no structural properties were assumed. We now add a new observation: we can construct the posets in such a way that for large i the poset P_i becomes a series composition of two or more smaller posets, and so the linear extensions of P_i are obtained by simply concatenating the linear extensions of the component posets. This will yield both theoretical savings in computational complexity as well as practical savings in running time.

For what follows, it is crucial that we consider the Quicksort algorithm, instead of an arbitrary fast comparison based sorting algorithm. When the algorithm chooses its first pivot element $p \in V$, it performs $|V| - 1$ comparisons; in other words, we will add the respective constraints, as well as those that follow by transitivity, yielding a poset $P_i = (V, \prec_i)$ with a partition of $V \setminus \{p\}$ into the predecessors A and successors B of p. We get that $a \prec_i p \prec_i b$ for all $a \in A$ and $b \in B$, and thus P_i is a series composition of the posets induced by the sets A, $\{p\}$, and B. Denoting $P_i' := P_i[A]$ and $P_i'' := P_i[B]$, we have $\ell(P_i) = \ell(P_i')\ell(P_i'')$. Consequently, we can estimate the ratio of interest, $1/\ell(P_i)$, by estimating $1/\ell(P_i')$ and $1/\ell(P_i'')$ independently and taking their product. This means that the product estimator $\widehat{\mu}_m$ will include factors from both branches. We may also branch using the reduction rule for parallel compositions: if the constructed poset decomposes into independent components A and B, we branch respectively and multiply the estimate $\widehat{\mu}_m$ by the factor $1/\binom{|A|+|B|}{|A|}$. Naturally, these branching rules are applied recursively, effectively resulting in a tree-structured decomposition into smaller subproblems rather than a sequence of same size problems. Figures 3 and 4 illustrate the simulation of Quicksort.

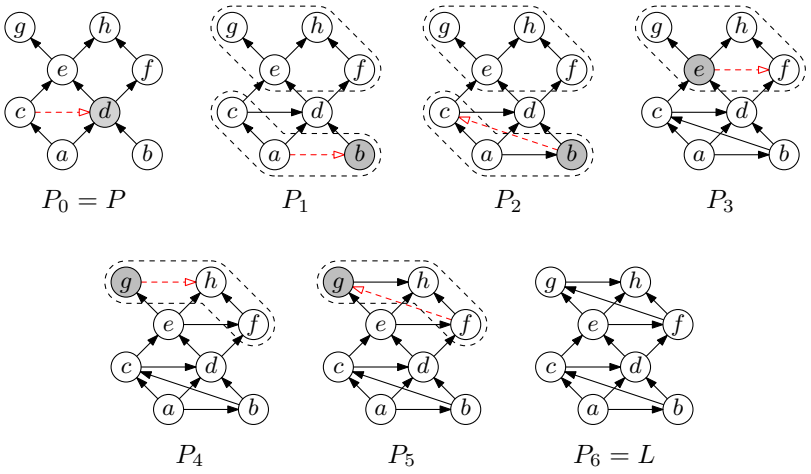


Figure 3: An example of the increasing sequence of posets $(P_i)_{i=0}^{6}$ obtained by simulating the Quicksort algorithm. The algorithm begins with the original poset $P_0 = P$, and picks a pivot element d. The only element that is incomparable with d is c, and by sampling linear extensions of P_0, the algorithm decides that adding the constraint in the direction $c \to d$ instead of direction $d \to c$ to form the next poset P_1 results in smaller reduction in the number of linear extensions. After this, the pivot d has partitioned the poset into its predecessors $\{a, b, c\}$ and successors $\{e, f, g, h\}$. The algorithm goes on to recursively order the set of predecessors, choosing b as the pivot and comparing it with the other elements, and the set of successors, first using pivot e and then g, eventually reaching a linear extension L. This sequence gradually reduces the number of linear extensions from $\ell(P) = 29$ to 1, because $(\ell(P_i))_{i=0}^{6} = (29, 15, 10, 5, 3, 2, 1)$.

Suppose we emulate a Quicksort implementation that always chooses a pivot that is a median of the elements. To find a median of n elements, one needs at most $3n$ comparisons (Dor & Zwick, 1999). Thus, after $4n - 1$ comparisons, the obtained poset P_{4n-1} is a series composition of two posets, consisting of the $n/2$ elements that respectively are smaller (or equal to) or larger than the pivot. After $2n - 1$ further comparisons in both branches, the poset becomes a series composition of 4 posets on $n/4$ elements, and so forth. Let us summarize and complete the analysis of this enhanced algorithm:

Theorem 9. *There is a randomized algorithm that given $\varepsilon, \delta \in (0, 1/2)$ and a poset on n elements, computes an (ε, δ)-approximation of the number of linear extensions of the poset in $O(\varepsilon^{-2} n^5 \log^2 n \log \delta^{-1})$ expected time.*

Proof. We analyze the complexity of the described algorithm; see Algorithm 3. Let P be the input poset on n elements. The algorithm operates on $k = O(n \log n)$ posets, of which $O(n)$ have at most n elements, $O(n)$ have at most $n/2$ elements, and so forth.

Per poset, $O(\log 2k/\delta)$ random linear extensions are generated to perform the comparison and to decide the ordering constraint to be added. This guarantees that $\mu_i \geq 1/3$ for all $i = 1, 2, \ldots, k$ with probability at least $1 - \delta/2$.

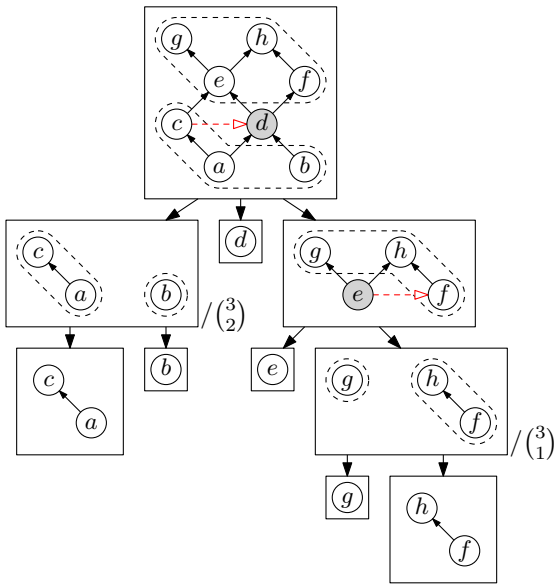


Figure 4: The tree obtained when using the Quicksort algorithm and the decomposition rules to the same poset P as in the example of Figure 3. After adding the constraint $c \to d$, the poset decomposes into three parts. The first one decomposes further into incomparable parts $\{a, c\}$ and $\{b\}$, which are linear posets. The second part is also linear, as it consists of only the pivot d. In the third part we have to add the constraint $e \to f$, but after that everything decomposes into linear parts without the need for additional sampling. Compared to the telescopic product algorithm without the decomposition rules, this method reduces the number of factors we need to estimate by sampling from six to two, and in the second sampling operation, the number of elements in the poset is 4 instead of 8.

By Proposition 8, additional $O(k\varepsilon^{-2})$ samples of linear extensions per poset suffice for guaranteeing that the estimate $\widehat{\mu}_m$ is an $(\varepsilon, 1/4)$ approximation of $1/\ell(P)$. By the median trick (Lemma 5) and taking the inverse estimate (Lemma 1), $O(k\varepsilon^{-2} \log 2/\delta)$ samples suffice to get an $(\varepsilon, \delta/2)$ approximation of $\ell(P)$, assuming $\mu_i \geq 1/3$ for all i.

The estimate can fail to be within a relative error of ε only if at least one of the two steps fails, thus with probability at most $\delta/2 + \delta/2 = \delta$.

Since the complexity of generating a random linear extension is at least linear in the number of elements, the work for the first $O(n)$ posets on at most n elements dominate the total complexity. Given that the expected time of generating a random linear extension is $O(n^3 \log n)$, the expected time for computing an (ε, δ)-approximation of $\ell(P)$ is $O\big((n^3 \log n)(n)(n \log n)\varepsilon^{-2} \log \delta^{-1}\big)$, which simplifies to the claimed bound. □

Algorithm 3 The Telescope Tree scheme

Input: A poset P, reals $\varepsilon, \delta > 0$, an exact sampler of linear extensions M.
Output: An (ε, δ)-approximation of $\ell(P)$.

function RatioEstimate(P, P_*) ▷ *Estimate of* $\ell(P)/\ell(P_*)$
  Draw $\lceil 10\,k\,(1+\varepsilon)^2\varepsilon^{-2} \rceil$ samples of linear extensions of P drawn using M
  Let μ_* be the fraction of samples that are also extensions of P_*
  return μ_*

function Pivot(P) ▷ *Select a pivot*
  Find a median p of P by comparing elements pairwise, each using $\lceil \ln 2k/\delta \rceil$ samples of linear extensions of P drawn using M
  return p

function AddConstraints(P, p) ▷ *Add constraints to* P
  $\mu_* \leftarrow 1$, $P_* \leftarrow P$
  for each element q incomparable to p in P **do**
    Draw $\lceil \ln 2k/\delta \rceil$ samples of linear extensions of P drawn using M
    Obtain P_* by adding the more frequent ordering of p and q to P
    Add to P_* the constraints that follow by transitivity
    $\mu_* \leftarrow \mu_* \cdot$ RatioEstimate(P, P_*)
    $P \leftarrow P_*$
  return (P_*, μ_*)

function TelescopeTree(P) ▷ *Estimate of* $\ell(P)$
  if P is a parallel composition of $P[A]$ and $P[B]$, both nonempty **then**
    return TelescopeTree$(P[A])$ · TelescopeTree$(P[B])$ · $\binom{|A|+|B|}{|A|}$
  $p \leftarrow$ Pivot(P)
  $(P_*, \mu_*) \leftarrow$ AddConstraints(P, p)
  Let A and B consist of the elements that respectively precede and succeed p in P_*
  return TelescopeTree$(P[A])$ · TelescopeTree$(P[B])$ / μ_*

$k \leftarrow \lceil n \log_2 n \rceil$, where n is the number of elements in P
$m \leftarrow \lceil 12 \ln 2/\delta \rceil$
for $j \leftarrow 1, 2, \ldots, m$ **do**
  $\hat{\ell}_j \leftarrow$ TelescopeTree(P)
return the median of $\hat{\ell}_1, \hat{\ell}_2, \ldots, \hat{\ell}_m$

4.4 Implementation

For both the basic and the enhanced estimator, our practical implementations deviate from the above descriptions in certain details, keeping the approximation guarantees, of course.

Instead of using the exact median, we choose the pivot heuristically: we maximize the minimum of the number of predecessors and the number of successors. In the factorization phase, we use a fixed number of samples per comparison; in preliminary experiments, 300 was found to give a good tradeoff in most cases.

Instead of using the upper bound for the number of samples per factor given in Proposition 8, we determine the required number of samples algorithmically, by finding the smallest bound that yields a failure probability bound at most δ, using binary search:

(i) For each candidate number of samples m, find for μ_i for all i a lower bound l in the range $[0, 1/2]$ (instead of the fixed $1/3$) that yields the smallest possible bound for the total failure probability, using ternary search.

(ii) For each candidate pair (m, l), bound the total failure probability as in the proofs of Proposition 8 and Theorem 9: the probability that some factor μ_i is smaller than l is bounded using a Chernoff bound (with 300 samples); the probability that the final estimate is not within a relative error of ε is bounded using Chebyshev's inequality.

Instead Huber's (2006) $O(n^3 \log n)$-time exact sampler, we use Huber's (2014) Gibbs sampler, which appears to work faster in practice, even if we know no good general running time bound for it. In its CFTP implementation, we use the PCG32 random number generator, which can skip backwards, to avoid storing generated random numbers. We also parallelize sampling into multiple threads and use 32-bit integers for coordinates (instead of floats) to avoid overhead and to obtain maximum precision for the chosen bit depth.

5. Schemes Based on the Tootsie Pop Algorithm

The *Tootsie Pop algorithm* (TPA) by Huber and Schott (2010) is a generic sampling-based method for estimating the ratio of sizes of two nested sets. The idea is, as suggested by the name of the method,[1] to draw a random sequence of intermediate sets, starting from the larger set and ending at the smaller set. The main difference to the telescopic product estimator is that in the TPA the number of intermediate steps is not fixed but a Poisson random variable whose expected value equals the natural logarithm of the ratio of interest. Thus, taking the average length of sufficiently many independent sequences yields, through the exp-function, an estimator of the ratio. Banks et al. (2018) applied the TPA for approximate counting of linear extensions.

In this section, we first consider the TPA in general. We begin by reviewing the technical requirements and results in detail in Section 5.1. Then, in Section 5.2, we present an improvement upon the basic TPA by taking a tighter control of tail probabilities of a Poisson distribution. We turn to the application to counting linear extensions in Section 5.3. We briefly review the scheme of Banks et al. (2018), in which the discrete set of linear extensions is augmented with a continuous unit-hypercube, and then present our second contribution in this section: a novel TPA based scheme for counting linear extensions. Our scheme differs significantly from the previous scheme and, among other things, requires a routine for sampling from a particular constrained state space; Section 5.4 is devoted to

1. A Tootsie Pop is a hard candy lollypop, known for the catch phrase "How many licks does it take to get to the Tootsie Roll center of a Tootsie Pop?", first introduced in a commercial on US television in 1969.

the needed sampler. Finally, we consider various implementation issues related to efficient parallelization of the computations in Section 5.5.

5.1 The Basic TPA

Let μ be a measure over a set U. In what follows, we implicitly assume that all subsets of U we consider are measurable; for simplicity of exposition, we do not explicitly treat the underlying sigma-algebra over U. Consider the problem of estimating the measure $\mu(S_1)$ of a given set $S_1 \subseteq U$, assuming we are given another set $S_0 \subseteq S_1$ whose measure $\mu(S_0)$ is known or easy to estimate.

The TPA utilizes a continuum of intermediate sets S_β, for $\beta \in [0, 1]$, specified such that the map $\beta \mapsto \mu(S_\beta)$ is continuous in $[0, 1]$ and $S_\beta \subseteq S_{\beta'}$ for all $0 \leq \beta \leq \beta' \leq 1$. Furthermore, the algorithm assumes the availability of a routine that for a given β generates a uniform sample from S_β, and a routine that for a given sample X finds the smallest β for which S_β contains X. Armed with these properties, the TPA repeats the following procedure some number of times, m, independently:

1. Let $k := 1$ and $\beta_0 := 1$.
2. Let X be a uniform sample from $S_{\beta_{k-1}}$.
3. Let $\beta_k := \inf\big\{\beta \in [0, 1] : X \in S_\beta\big\}$.
4. If $\beta_k = 0$, return k; else let $k := k + 1$ and go to step 2.

We have that $k - 1$ is Poisson distributed with mean $r := \ln\big(\mu(S_1)/\mu(S_0)\big)$. Indeed, in each step of the algorithm we decrease $\mu(S_1)$ by a factor that is uniformly distributed in $[0, 1]$, until we reach $\mu(S_0)$; this is equivalent to decreasing $\ln \mu(S_1)$ by subtracting exponentially distributed random variables, yielding the Poisson distribution. As an estimate of the ratio $\mu(S_1)/\mu(S_0) = \mathrm{e}^r$, the algorithm outputs $\mathrm{e}^{Z/m}$, where Z is the sum of the sampled m Poisson variables.

To guarantee that the output is an (ε, δ) approximation, it suffices to ensure that $\exp\{|Z/m - r|\} \geq 1 + \varepsilon$ with probability at most δ. Applying the tail bound in Lemma 4, with $\lambda := mr$ and $a := m \ln(1 + \varepsilon)$, leaves us to solve $2 \exp\{-a^2/(2\lambda + 2a)\} \leq \delta$, yielding

$$m \geq 2(r + \varepsilon_0)\, \varepsilon_0^{-2} \ln(2\delta^{-1}), \quad \text{with} \quad \varepsilon_0 := \ln(1 + \varepsilon).$$

For small $\varepsilon > 0$ we have $\varepsilon_0 \approx \varepsilon$, and for all $\varepsilon < 1/2$ we have $\varepsilon_0 > 4\varepsilon/5$ by standard bounds.

As r is unknown, we run the TPA in two phases (Banks et al., 2018), first approximating r to within a larger factor and then using this approximation to bound m. More precisely, we use $m_0 := \lceil 2 \ln(2\delta^{-1}) \rceil$ initial samples to get an approximation r_0, and then select the smallest integer m that exceeds $2(r' + \varepsilon_0)\, \varepsilon_0^{-2} \ln(4\delta^{-1})$, where $r' := r_0 + \sqrt{r_0} + 2$. Namely, the first phase guarantees $r \leq r'$ with probability at least $1 - \delta/2$. Since the expected value of r_0 is r, the increase in the expected number of samples is negligible when $r \gg 2$. Asymptotically, the TPA consumes $O(r^2 \varepsilon^{-2} \log \delta^{-1})$ samples in total on expectation.

Remark 1. We can slightly lower the bound of Banks et al. (2018), to $r' := r_0 + \sqrt{r_0} + 1$. Namely, if $r \leq 1$, then $r \leq r'$ holds surely. Otherwise, algebraic manipulation reveals that $r \leq r'$ fails only if $r_0 < r - a_r$, where $a_r := 1/2 + \sqrt{r - 3/4}$. Since $m_0 r_0$ is Poisson distributed

with mean $m_0 r$, the failure probability is, by Lemma 4, at most $\exp\big\{-m_0 a_r^2/(2r)\big\}$, which is at most $\delta/2$, since $m_0 \geq 2\ln(2\delta^{-1})$ and $a_r^2 = r - 1/2 + \sqrt{r - 3/4} \geq r$.

5.2 An Enhanced TPA

We next present an improved scheme to bound the required number of independent samples, m, from the Poisson distribution. The main idea is to avoid using the tail bounds, which can be relatively loose for certain configurations of the key parameters r and ε. Recall that we seek an m, as small as possible, such that $\Pr(|Z/m - r| < \varepsilon_0) \geq 1 - \delta$. In principle, we could compute the exact probability by summing up the point probabilities $\Pr(Z - mr = j)$ for all integers j satisfying $|j| < m\varepsilon_0$. The issue is that we do not know the value r; we only have an upper bound r'. In what follows, we will effectively consider a range of values of the unknown $r \leq r'$.

Suppose we have guessed m and wish to determine whether m is sufficiently large; we use binary search to find the smallest sufficient m. Put $\lambda' := mr'$ and $d := m\varepsilon_0$. For all $\lambda > 0$, write Z_λ for a Poisson random variable with mean λ, and $p(\lambda) := \Pr(|Z_\lambda - \lambda| < d)$. Furthermore, let $p' := \inf\{p(\lambda) : 0 < \lambda \leq \lambda'\}$. Now, if $p' \geq 1 - \delta$, we know that the guessed m is sufficiently large, no matter what the unknown r is. The challenge is to compute the value p', or at least a good lower bound for it. If $p(\lambda)$ was a decreasing function of λ, we would simply have $p' = p(\lambda')$. But due to the discrete nature of the Poisson distribution, this is not the case. To address this issue we craft a function that bounds $p(\lambda)$ from below and is monotonically decreasing.

We begin by writing the cumulative distribution function in terms of the regularized upper incomplete gamma function Q (a well-known fact obtained by integration by parts):

$$\Pr(Z_\lambda \leq z) = Q(\lfloor z \rfloor + 1, \lambda)\,, \quad \text{where} \quad Q(s,x) := \frac{\Gamma(s,x)}{\Gamma(s)}\,;$$

here $\Gamma(s,x) := \int_x^\infty t^{s-1}\mathrm{e}^{-t}\mathrm{d}t$ is the upper incomplete gamma function. We obtain

$$p(\lambda) = \Pr(Z_\lambda < \lambda + d) - \Pr(Z_\lambda \leq \lambda - d) = Q(\lceil \lambda + d \rceil, \lambda) - Q(\lfloor \lambda - d \rfloor + 1, \lambda)\,.$$

We can show that $Q(s,x)$ is increasing in s, and thus we get a lower bound for $p(\lambda)$ by removing the ceiling and floor functions:

$$q_d(\lambda) := Q(\lambda + d, \lambda) - Q(\lambda - d + 1, \lambda)\,.$$

Furthermore, if $d \geq 1$, we can show that the lower bound $q_d(\lambda)$ is decreasing in λ, and thus $q_d(\lambda') \leq p' = \inf\{p(\lambda) : 0 < \lambda \leq \lambda'\}$. The needed arguments and calculation are given in Appendix A.1, in the proof of the following result.

Proposition 10. *Let $d \geq 1$. Then $q_d(\lambda') \leq p'$.*

Remark 2. The constraint $d \geq 1$ is not restrictive in practice. Namely, if $d < 1$, there can be at most 2 integers within a distance less than d from λ. Since the mode of Z_λ is $\lfloor \lambda \rfloor$, we obtain[2] $\Pr(|Z_\lambda - \lambda| < d) \leq \Pr(Z_\lambda = \lfloor \lambda \rfloor) + \Pr(Z_\lambda = \lfloor \lambda \rfloor + 1)$, which is at most $1/\sqrt{2\pi} + 1/\sqrt{4\pi} < 3/4$ at any $\lambda \geq 1$, and so $p' \leq p(\lambda')$ is not sufficiently large (assuming $\delta \leq 1/4$); note that $\lambda' \geq r' \geq 1$ (cf. Remark 1).

2. Using Stirling's formula, $\Pr(Z_\lambda = j) = \lambda^j \mathrm{e}^{-\lambda}/j! \leq (2\pi j)^{-1/2}(\lambda/j)^j \mathrm{e}^{j-\lambda} \leq (2\pi j)^{-1/2}$.

We summarize the proposed enhanced TPA:

1. Run the TPA for $m_0 = \lceil 2\ln(2\delta^{-1})\rceil$ samples, obtain an estimate r_0, and set $r' := r_0 + \sqrt{r_0} + 1$.
2. Using binary search, find the smallest integer $m \geq 1$ for which $d := m\ln(1+\varepsilon) \geq 1$ and $q_d(mr') \geq 1-\delta$. (Replaces the Chernoff bound based estimate of the basic TPA.)
3. Run the TPA for m samples, obtain the sum Z, and return $\mathrm{e}^{Z/m}$.

Remark 3. We have empirically observed that our enhanced scheme typically reduces the number of samples m to around one third, as compared to the basic bound.

5.3 Counting Linear Extensions with Relaxation TPA

Banks et al. (2018) instantiate the TPA for approximate counting the linear extensions of a given poset P. The algorithm embeds the discrete problem into a continuous space by augmenting the set of linear extensions $\mathcal{L}(P)$ with a continuous dimension. More precisely, the outer set S_1 is $\mathcal{L}(P) \times [0,1]$ and the inner set S_0 is $\{L_0\} \times [0,1]$, where L_0 is a fixed linear extension of P; for a careful construction of the intermediate sets S_β and details of the scheme, we refer to the original work. The algorithm runs in expected time $O\big(n^3 \log n \, \log^2 \ell(P)\, \varepsilon^{-2} \log \delta^{-1}\big)$.

We employ a very different embedding. With each element $a \in V$ in a poset $P = (V, \prec)$ we associate a position $x_a \in [0,1]$ and require $x_a \leq x_b$ whenever $a \prec b$. Putting

$$S_0 := \big\{x \in [0,1]^V : x_a \leq x_b \text{ if } a \prec b\big\}$$

we obtain $\ell(P)$ as $\mu(S_0)n!$, where $\mu(S_0)$ is the n-dimensional volume of the polytope S_0. We estimate this volume using TPA by comparing it to $\mu(S_1) = 1$, where S_1 is the whole hypercube $[0,1]^V$. The sets S_0 and S_1 are connected by a continuum of sets S_β obtained by adding $\beta \in [0,1]$ as a slack variable to the inequality constraints as follows:

$$S_\beta := \big\{x \in [0,1]^V : x_a - x_b \leq \beta \text{ if } a \prec b\big\}\,.$$

Note that in our scheme $\mathcal{L}(P)$ is bridged to an extreme *relaxation*, an empty partial order, whereas the scheme of Banks et al. (2018) works in the direction of *extensions*.

To optimize our scheme further, we incorporate the idea of ARMC and use a close relaxation $R = (V, \prec')$ of P to squeeze the estimated ratio. Accordingly, we take

$$S_1 := \big\{x \in [0,1]^V : x_a \leq x_b \text{ if } a \prec' b\big\}$$

as the outer set, instead of $[0,1]^V$, the estimated ratio becoming $\mu(S_1)/\mu(S_0) = \ell(R)/\ell(P)$. To connect S_0 and S_1, we generalize the definition of S_β to

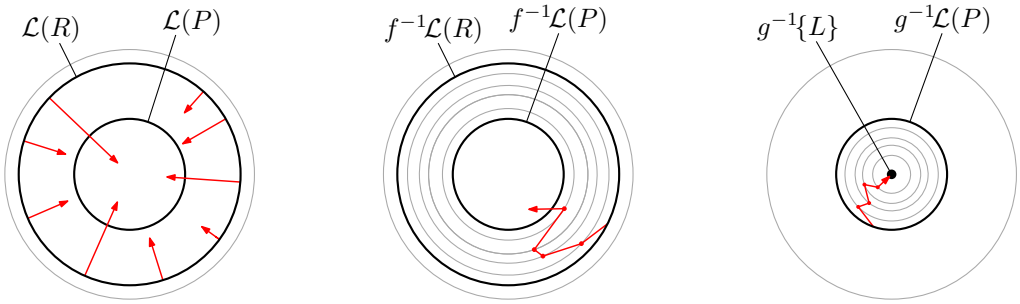
$$S_\beta := \big\{x \in [0,1]^V : (x_a - x_b \leq \beta \text{ if } a \prec b) \text{ and } (x_a \leq x_b \text{ if } a \prec' b)\big\}\,.$$

We will give an algorithm, the *constraint hypercube sampler*, for drawing exact samples from S_β in Section 5.4. Observe that given a sample $x \in [0,1]^V$, it is straightforward to find the

Algorithm 4 The Relaxation TPA scheme

Input: A poset P, reals $\varepsilon, \delta > 0$, constrained hypercube sampler M.
Output: An (ε, δ)-approximation of $\ell(P)$.

1. Find a relaxation R of P, along with $\ell(R)$, using the heuristics
2. Obtain an estimate Y of $\ell(P)/\ell(R)$ by running the (enhanced) TPA on (ε, δ) and the sets $\{S_\beta\}$ specified by P and R, using the sampler M
3. **return** $\ell(R) \cdot Y$



(a) ARMC (b) Relaxation TPA (c) Banks et al.

Figure 5: Comparison of the relaxation TPA scheme (b) to two other methods (a, c). Like (a), the scheme computes the number of linear extensions of poset P by estimating its ratio to the known number of linear extensions of a relaxation R of P. Method (c) works in the other direction: it relates the size of $\mathcal{L}(P)$ to that of $\{L\}$ where $L \in \mathcal{L}(P)$. Similarly to (c), the scheme uses an embedding f from the continuous space $[0,1]^V$ to the space of linear orders and imposes a layered structure in the continuous space, allowing it to use TPA, which estimates the ratio using decreasing random walks with respect to the layers. However, the embedding and the layers used in (b) are different from those in (c). In contrast, (a) works directly in the space of linear orders and uses Monte Carlo to estimate the ratio by repeatedly drawing samples from $\mathcal{L}(R)$. Adapted from Talvitie et al. (2018b).

smallest β such that $x \in S_\beta$: simply set $\beta := \max\{x_a - x_b : a \prec b\}$, which can be computed in time $O(n^2)$.

Algorithm 4 shows the steps of the resulting *Relaxation TPA* scheme. Figure 5 illustrates the differences of ARMC, the scheme of Banks et al. (2018), and Relaxation TPA.

Substituting $\ln \ell(R)/\ell(P)$ for the ratio r in the complexity bound of the generic TPA yields the following:

Proposition 11. *Given a relaxation R of P, with known $\ell(R)$, Relaxation TPA computes an (ε, δ)-approximation of $\ell(P)$ using on expectation $O\big((\log^2(\ell(R)/\ell(P))\,\varepsilon^{-2}\log\delta^{-1}\big)$ independent samples from the sets S_β, $0 \le \beta \le 1$.*

To find a relaxation R that is close to P, we use several heuristics. The heuristics differ from those employed in ARMC, for here the goal is to avoid the exponential growth of the running time in the number of elements. Our heuristics stem from the observation that for

many special classes of posets, the number of linear extensions is easy to compute exactly or the problem can at least be reduced to counting the linear extensions of some induced subposets. First we check if the poset already belongs to such a special class:

1. If P is a series composition of P_1 and P_2, then we recursively find the relaxations R_1 and R_2, respectively, and take R as their series composition.
2. If P is a parallel composition of P_1 and P_2, then we recursively find the relaxations R_1 and R_2, respectively, and take R as their parallel composition.
3. If the cover graph of P is a tree, then we use P as the relaxation.
4. If an exact linear extension counter finishes computing $\ell(P)$ within a short time limit, we use P as the relaxation. In our experiments we ran the counter due to Kangas et al. (2016) until its dynamic programming table has 10,000 items.

If none of these attempts succeeds, we try the next two methods and select the relaxation with fewer linear extensions.

5. We relax P by taking a spanning tree of its cover graph. We repeat the following 6 times: Assign random weights to all cover graph edges, construct a minimum spanning tree using these weights, and compute the number of linear extensions using the known exact algorithm for posets whose cover graph is a tree (Atkinson, 1990).
6. If V can partitioned into V_1, V_2, V_3 such that V_1 precedes V_2 (i.e., every $a \in V_1$ precedes every $b \in V_2$), then we recursively find relaxations $R_k := (V_k, \prec_k)$, $k = 1, 2, 3$, and combine them into relaxation $R := (V, \prec')$, where we keep the constraints between V_1 and V_2 but remove all constraints between $V_1 \cup V_2$ and V_3. We search in a greedy fashion for a partition that maximizes the value $\binom{|V_1|+|V_2|}{|V_1|}$; this would be inversely correlated with the number of linear extensions if every $\prec_k$ was empty. We try all initializations $V_1 := \{a\}$, for $a \in V$, and add elements to V_1 one by one while maintaining V_2 as the maximal set such that V_1 precedes V_2.

5.4 Continuous Relocation Chain

We now turn to the problem of generating a uniformly distributed sample from the *constrained hypercube*

$$\Omega := \left\{ x \in [0,1]^n : x_i - x_j \leq s_{ij} \text{ for all } 1 \leq i < j \leq n \right\},$$

where the numbers $s_{ij} \geq 0$ are given as input.

To sample from Ω, we give a simple Markov chain, which we call the *continuous relocation chain*. In the chain, the next state y is obtained from the previous state x by the following transition: First, sample dimension $i \in \{1, 2, \ldots, n\}$ and coordinate $u \in [0, 1]$ independently and uniformly at random, and let y' be the state obtained by setting $y'_i := u$ and $y'_j := x_j$ for $j \neq i$. Second, if $y' \in \Omega$, let $y := y'$; otherwise do nothing, $y := x$. Accordingly, the *transition function* of the chain is $\phi : (x, i, u) \mapsto y$. Since the transition is symmetric, that is, the probability density of y given x is the same if we swap the states, we have the following.

Proposition 12 (Uniform stationary distribution)**.** *The uniform distribution on Ω is a stationary distribution of the continuous relocation chain.*

The continuous relocation chain is similar to a Markov chain due to Huber (2014), which can be used for sampling from the uniform distribution on Ω in the special case where $s_{ij} \in \{0, 1\}$ for all $1 \le i < j \le n$. Huber achieved perfect simulation of the chain employing monotone coupling from the past. We will next show that the same technique applies to our continuous relocation chain for sampling from Ω in the more general case of arbitrary values $s_{ij} \ge 0$.

It suffices to show that the chain is monotone in relation to its transition function ϕ and some partial order $\sqsubset$ on Ω. Define

$$\sqsubset := \big\{(x, y) \in \Omega \times \Omega : x \ne y \text{ and } x_i \le y_i \text{ for all } 1 \le i \le n\big\}.$$

Denote by $\mathbf{0}$ and $\mathbf{1}$ the elements $(0, 0, \ldots, 0)$ and $(1, 1, \ldots, 1)$ of the hypercube $[0, 1]^n$. We prove the following in Appendix A.2.

Proposition 13 (Monotonicity)**.** *We have the following:*

(i) The relation $\sqsubset$ is a partial order on Ω.

(ii) We have $\mathbf{0}, \mathbf{1} \in \Omega$ and $\mathbf{0} \sqsubseteq x \sqsubseteq \mathbf{1}$ for all $x \in \Omega$.

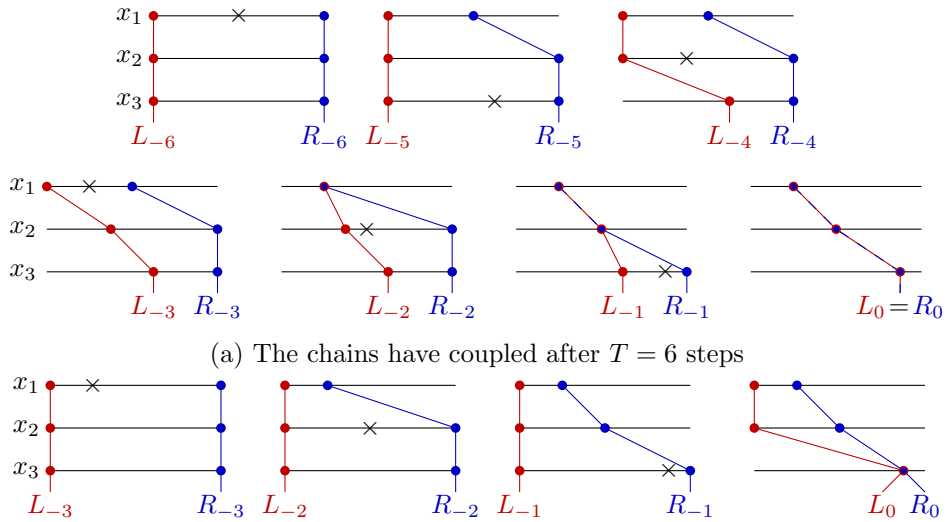
(iii) If $x \sqsubseteq y$, then $\phi(x, i, u) \sqsubseteq \phi(y, i, u)$ for all $i \in \{1, 2, \ldots, n\}$ and $u \in [0, 1]$.

The generic method of monotone CFTP now yields the following algorithm to draw a sample from Ω. We run two parallel *bounding chains* $(L_t)_{t=-T}^{0}$ and $(R_t)_{t=-T}^{0}$ that are copies of the continuous relocation chain, governed by the same sequence of random numbers i_t and u_t, but with the different initial states $L_{-T} := \mathbf{0}$ and $R_{-T} := \mathbf{1}$. The term *coupling* refers to the chain (L_t, R_t) on the state space $\Omega \times \Omega$. If it happens that $L_0 = R_0$, the chains have *coupled* and we know due to the monotonicity property of Proposition 13(iii) that a chain that started at time $-T$ in *any* initial state must also be in this state at time 0; thus this state can be returned as a sample from the stationary distribution. Otherwise, $L_0 \ne R_0$ and we need to run the chain starting further into past, in which case we double T and retry, always using the same random numbers (i.e., the dimension $i = i_t$ and the coordinate $u = u_t$) for the same transition from time $t - 1$ to time t. Figure 6 illustrates the behavior of the bounding chain.

We can show that the expected running time of this sampling algorithm is finite, as the chains will couple with positive probability for some $T > 0$; see Appendix A.3 for a proof.

Theorem 14 (Finite expected time)**.** *Monotone CFTP for the continuous relocation chain outputs a sample from the uniform distribution on Ω in finite expected time.*

One advantage of coupling from the past compared to directly simulating the chain is that, even though we do not have any good a priori bounds on the number of transitions T needed for coupling, we can still use the algorithm in practice, as it knows when to stop.



(a) The chains have coupled after $T = 6$ steps

(b) The chains have not coupled after $T = 3$ steps

Figure 6: We visualize the bounding chains $(L_t)_{t=-T}^{0}$ and $(R_t)_{t=-T}^{0}$ when using monotone coupling from the past with the continuous relocation chain for sampling from $\Omega = \{x \in [0,1]^3 : x_1 \leq x_2 \leq x_3\}$. We use the symbol $\times$ to mark the random numbers (the dimension $i \in \{1,2,3\}$ and coordinate $u \in [0,1]$) for each transition. In (a), we see that after $T = 6$ steps, the chains have coupled and we obtain a sample $L_0 = R_0$, whereas in (b) we see that $T = 3$ is not sufficient for the chains to couple with these random numbers.

5.5 Implementation

Our theoretical algorithm design and analysis necessarily were limited to a simplistic model of computation. For practical implementation, we have various opportunities to optimize the computations by taking into account the available modern computing architecture, in particular, multiple threads, vector instructions, and a GPU. Next we present some key implementation details and the modifications needed for the TPA scheme.

5.5.1 Numbers, Threads, and Restructuring of the Markov Chain

We parallelize sampling to multiple threads; each TPA sequence from $\beta = 1$ to $\beta = 0$ is always handled in a single thread, but different TPA sequences can be run independently in different threads. In the CFTP, we use the PCG32 random number generator (RNG), which can skip backwards to avoid storing generated random numbers. We use 32-bit integers for coordinates, instead of floats, to avoid overhead and to obtain maximum precision for the chosen bit depth.

In the continuous relocation chain, the usage of a random element on each iteration makes vectorization difficult and adds RNG overhead. As a remedy, we divide the iterations into groups of size $2n - 1$ such that in each of these "super-iterations," the chosen elements

are always in the order $1, 2, \ldots, n, n-1, \ldots, 1$. While the same correctness analysis works for this chain, the behavior of the chain may be sensitive to the order of the elements—for fairness of running time comparisons, we always first shuffle the input poset elements. Importantly, this chain implementation has a completely static data path: there is no indexing by variables, we only perform fixed operations on 32-bit and 64-bit registers.

5.5.2 Vectorization

Now that all computations run on simple 32-bit integer operations (with some 64-bit operations in the implementation of PCG32, but these can be split into 32-bit parts) with a static data path, the algorithm enables relatively straightforward vectorization.

The basic principle is as follows. In our vectors, we have $L = 8$ (256-bit AVX2) or $L = 16$ (512-bit AVX512) 32-bit lanes; this means that we can do per-lane basic arithmetic/logical operations between vectors in an optimized manner. Within one thread, each lane is responsible for running a single TPA sequence at a time, that is, we have vectors for the left and right bounding chain positions of each element, the RNG state, and β values, each vector containing data for all L parallel independent computations. Thus, these L lanes can be thought of as "sub-threads" of each thread.

Most of the time we are running the bounding chain for the L states in the L lanes; as this does exactly the same things for all lanes, it vectorizes perfectly. The rest of the time, we do housekeeping tasks: if we have run the bounding chain to completion, we check coupling: if the bounding chains have coupled, then we reinitialize to the next step in the TPA sequence (or start a new sequence if we have reached $\beta = 0$); otherwise, we reinitialize with twice the number of iterations and RNG skipped back. Since the convergence time of each bounding chain is random, not all lanes need to do these tasks at the same time; as this part does not vectorize, we perform these tasks serially only for the lanes that need them. These tasks being quite rare, the effect on the total efficiency is small; we further help with this by always performing the super-iterations (of $2n-1$ iterations) in chunks of 16, and only checking the states between chunks.

5.5.3 GPU Implementation

For our purposes, the NVIDIA GeForce RTX 2080 GPU we used in the experiments is essentially 46 roughly independent cores ("symmetric multiprocessors"), each running at most 2048 threads, organized into warps of 32 threads that run in lockstep, similarly to vectorized computations with $L = 32$. Each core also has 64 kB of fast core-specific memory ("shared memory"); in addition, there is a large amount of slower global memory. While other modern general-purpose computing-capable GPUs typically have slightly different configurations, similar algorithm design principles and constraints still apply.

Each core advances the computation of only one warp on each clock cycle; still, to hide latencies, there should typically be multiple warps running alternately to get maximum efficiency. A common rule of thumb is that we need 8–32 warps, i.e., 256–1024 threads per core. By this rule, to achieve trivial parallelization, we should have between $46 \cdot 256 = 11776$ and $46 \cdot 1024 = 47104$ independent computations. In our case, the independent computations are the m independent TPA sequences. This number m is typically relatively small. For example, when considering AvgDeg(5) instances (see Sect. 6) of size 256 and $(\epsilon, \delta) = (1, 1/4)$,

then m is roughly 250; if $(\epsilon, \delta) = (1/2, 1/10)$, then m is a bit closer to the ideal range, about 5000. Furthermore, if we wish to store the chain state in the faster shared memory, we have only space for $64 \cdot 1024/(4 \cdot 2 \cdot n) = 8192/n$ concurrent TPA random walks per core. For example, if $n = 512$, then we have space only for 16 TPA walks.

Due to the aforementioned higher parallelism requirements and stricter memory constraints in GPUs compared to vectorized CPU computations, we need to introduce parallelism to the simulation of each individual bounding chain, resulting in the following scheme:

(i) Each core handles 16 TPA walks concurrently; each of them is handled by two lanes of each warp. (As the left and right bounding chains do the same computations, we move them to separate lanes to increase parallelism.)

(ii) We use 32 warps per core, operating concurrently on a shared state when running the continuous relocation chain. For the sake of efficiency, we do not want to synchronize the warps too often when running the chain; to this end, we heuristically find a proper coloring of the elements (no constraint between two elements of the same color), and in each super-iteration handle the elements color by color in parallel (each element being independent of the movement of the other elements of the same color), synchronizing the warps only when changing the color. Note that in this GPU implementation, the element order comes from the coloring, instead of being sampled uniformly at random. Ideally, there are only a few colors and thus, if n is large enough, most warps are kept busy for most of the time.

(iii) If the chain states fit into shared memory (i.e., $n \leq 512$), we use it; otherwise, we use global memory (the shared memory then works as a L1 cache).

6. Experimental Results

In the previous sections, we have developed algorithms that, given a poset and probabilistic error tolerance (ε, δ), output a real number that, with probability at least $1 - \delta$, is within a relative error at most ε of the number of linear extensions of the poset. For the time complexity of the algorithms we only gave some characterizations relying on conservative choices of some user parameters (e.g., the underlying Markov chain sampler) to obtain worst-case bounds. We have argued that the algorithms are likely to run faster in "typical instances" if replacing the conservative choices by more adaptive ones.

In this section, we present empirical results that compare the performance of the algorithms and their implementations (summarized in Section 6.1) on various problem instances (described in Section 6.2). We first compare the generic algorithm designs ARMC, Telescope product, and TPA (Section 6.3) and then the performance of different optimizations of TPA (Section 6.4).

6.1 Algorithms, Implementations, and Environment

Of the three generic algorithm designs, we implemented multiple variants for telescope product estimators and for the TPA to assess how the proposed ideas affect the running time in practice. Table 1 summarizes the tested algorithms and implementations.

Name	Description	Reference
ARMC	The ARMC scheme	Algorithm 2, Section 3.4
BasicTelescope	The basic Brightwell–Winkler estimator	Sections 4.2 and 4.4
Telescope	The Telescope Tree scheme	Algorithm 3, Section 4.4
ChernoffTPA	The relaxation TPA using Chernoff bounds	Algorithm 4, Section 5.1
TPA	The relaxation TPA using the enhanced analysis	Algorithm 4, Section 5.2
AVX2TPA	Implementation of TPA using AVX2 instructions	Section 5.5.2
AVX512TPA	Implementation of TPA using AVX512 instructions	Section 5.5.2
GPUTPA	Implementation of TPA using GPU	Section 5.5.3

Table 1: Tested algorithms and implementations

We implemented each algorithm variant in C++ (for GPUTPA we used CUDA C++); the implementations are available in `https://github.com/ttalvitie/linext`. Each implementation is multithreaded to take advantage of all the cores of the CPU. Apart from the vectorized implementations AVX2TPA, AVX512TPA, and the GPU implementation GPUTPA, none of the implementations use CPU vector extensions or GPUs, allowing fair comparison of their running times. The experiments were run on identical high-end desktop computers, each with a 4-core 3.6 GHz Intel Xeon W-2123 CPU and an NVIDIA GeForce RTX 2080 GPU.

6.2 Problem Instances

We consider the same types of benchmark poset instances as the preliminary works (Talvitie et al., 2018a, 2018b), with extended range in the poset size n and density of random posets.

We included two classes of random posets:

Bipartite(p) is a random height-2 poset $P = (A \cup B, \prec)$ with $|A| = \lceil n/2 \rceil$, $|B| = \lfloor n/2 \rfloor$, $A \cap B = \emptyset$ constructed by ordering $a \prec b$ for each $(a, b) \in A \times B$ independently with probability p, leaving a and b incomparable with probability $1 - p$. We considered the values $p \in \{0.2, 0.5\}$.

AvgDeg(k) is a random poset $P = (V, \prec)$ constructed by first randomly shuffling V into a linear order $(v_1, v_2, \ldots, v_n)$, then forming a DAG (V, E) by including each possible edge $(v_i, v_j) \in E$ for $1 \leq i < j \leq n$ independently with probability $p := k/(n-1)$, and finally taking $\prec$ as the transitive closure of E on V. With this choice of p, the expected average indegree and degree are respectively $n^{-1} \sum_{i=1}^{n} (n-i)p = k/2$ and k. We varied k in $\{3, 5, 7, 9, 13, 17\}$ but will show results only for $k \in \{3, 9, 17\}$.

In addition, we generated random posets of varying sizes from the four largest benchmark Bayesian network structures (DAGs) *Diabetes*, *Link*, *Pigs*, and *Munin* available in the Bayesian Network Repository (`https://www.cs.huji.ac.il/~galel/Repository/`); Table 2 gives some basic statistics of these DAGs. For each of these four DAGs, we obtain a random poset of size n as the transitive closure of a connected subgraph induced by a set of n nodes constructed iteratively: starting from a random node, in each iteration we pick, uniformly at random, a node that is adjacent to at least one of the already chosen nodes.

Name	Number of nodes	Number of arcs	Average indegree
Diabetes	413	602	1.46
Pigs	441	592	1.34
Link	724	1125	1.55
Munin	1041	1397	1.34

Table 2: Benchmark Bayesian network structures

For every poset type and size $8 \leq n \leq 1024$, in roughly 26 % geometric increments, we generated five independent benchmark posets. We ran every algorithm variant for every poset with a time limit of 6 hours. For ARMC, the memory limit was set to 30 gigabytes; the other algorithms are not memory-bound. To see the effect of the approximation quality parameters (ε, δ) on the performance of the algorithms, we ran all the algorithm variants with both the configuration $(\varepsilon, \delta) = (1, \frac{1}{4})$ that was used in the preliminary publications and a stricter configuration $(\varepsilon, \delta) = (\frac{1}{4}, \frac{1}{10})$.

6.3 Comparison of Algorithms

The results on the class *Bipartite*(p) reveal dramatic differences between the approximation schemes (Fig. 7). Regardless of the density, the error tolerance, or the poset size, TPA is the fastest, ChernoffTPA slightly slower (up to a factor of 5 for larger instances), Telescope is significantly slower (by several orders of magnitude for larger instances), and BasicTelescope is the slowest by a large margin. For these schemes the time requirements appear to grow polynomially in the poset size. ARMC is an outlier: the growth rate appears to be exponential (in agreement with the theory), yet the scheme is the fastest for posets of moderate size, say from 32 to 128. ARMC is also the least sensitive to the error tolerance; with the stricter configuration, its performance improves in relation to the other schemes.

On the class *AvgDeg*(k), we see the same pattern, especially for the smaller values of k (Fig. 8). We see that BasicTelescope and Telescope are not sensitive to the density of the poset and that the employed Gibbs sampling algorithm does not slow down the computations even if the sampler is designed particularly for height-2 posets. On the other hand, the perfomance of ChernoffTPA and Telescope clearly degrade as k grows. ARMC is very competitive up to some critical poset size between about 128 and 256, depending on k.

The results on posets constructed from benchmark Bayesian network structures are very similar to those on *AvgDeg*(k) with $k = 3$, that is, with expected average indegree $k/2 = 1.5$ (Fig. 9). This may not be surprising since each generating Bayesian network structure has a small average indegree, around 1.5, which is expected to be inherited by the constructed random subgraphs. There are also differences between the four benchmark structures: *Diabetes* instances clearly are the hardest for the TPA based schemes, the other three classes being close to each other; for example, on posets of size 256 under the looser error tolerance, TPA runs in some minutes on *Diabetes* instances, while taking only some seconds on the other three classes. ARMC is competitive under the stricter error tolerance, however, with significantly larger variance in the running times as compared to the other schemes or to ARMC on the instances classes *Bipartite*(p) and *AvgDeg*(k).

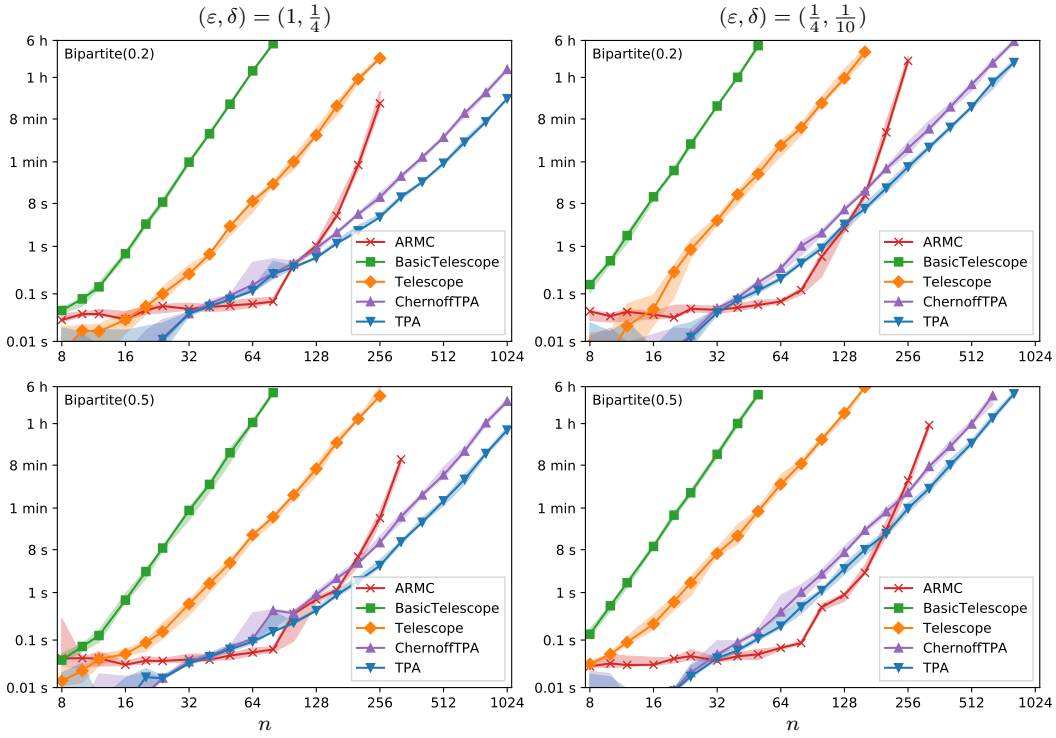


Figure 7: Empirical running times for (ε, δ)-approximation of the number of linear extensions of random height-2 posets, as a function of the poset size n. The solid line shows the median and the shaded area the range of five independent instances per n.

6.4 Comparison of TPA Implementations

We compared the four implementations of the enhanced TPA on all classes of random posets, but show here only selected results focusing on the most challenging instance classes (Fig. 10).

Clearly, GPUTPA is the slowest on the smallest instances up to sizes around 128, after which it is the fastest or second fastest of the four schemes. On the largest instances GPUTPA is the fastest, followed by AVX512TPA, AVX2TPA, and TPA, in this order; the difference of the fastest and the slowest scheme ranges from one to two orders of magnitude; an exception is the *Diabetes* class with the looser approximation guarantees, for which the differences between the schemes are small. With the tighter approximation guarantees, all schemes become slower, as expected. The size of this effect, however, depends on the scheme: on the largest instances, TPA slows down by a factor around 50, while for GPUTPA the factor is around 10—this can be explained by the relatively small number of independent computations and thus underuse of the GPU with the looser guarantees (see Section 5.5.3).

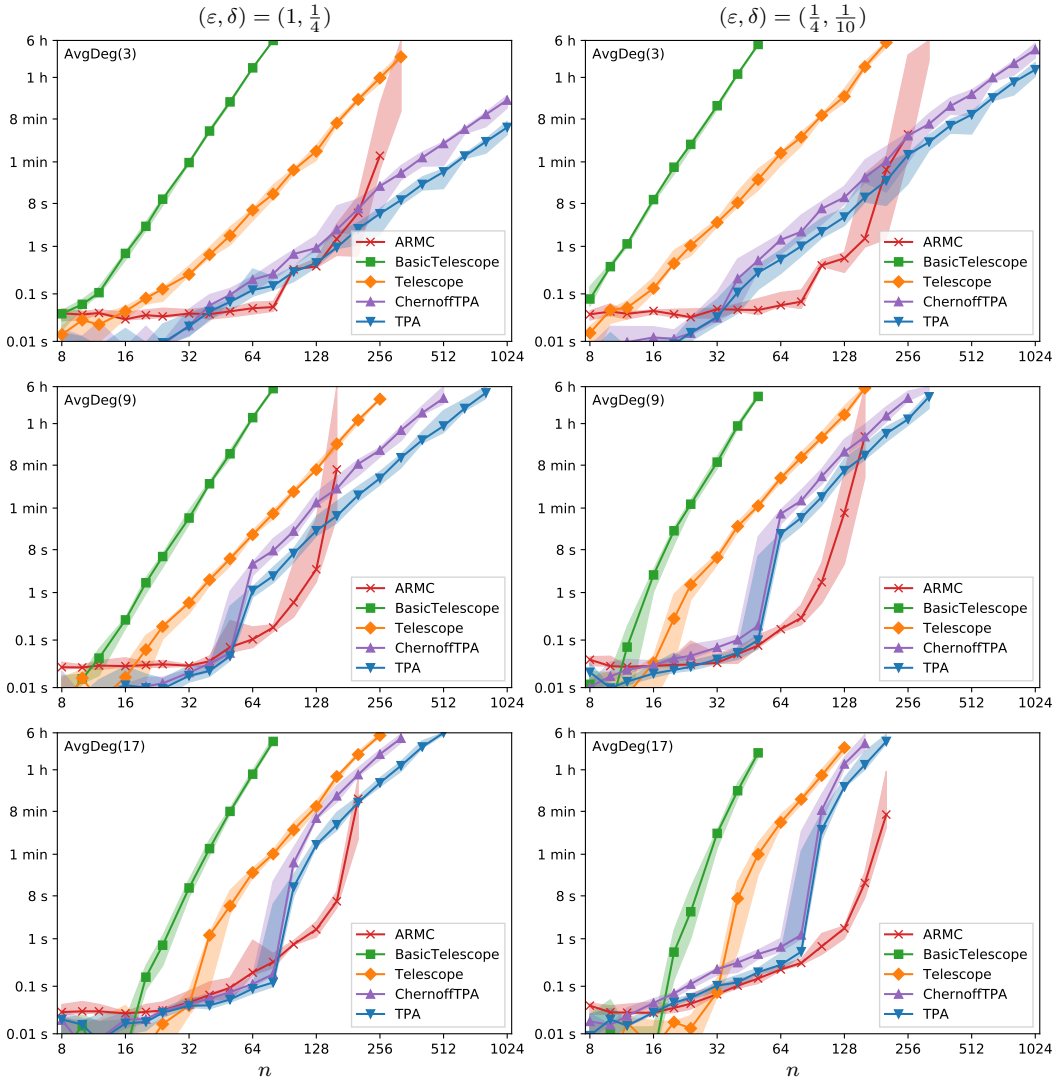


Figure 8: Empirical running times for (ε, δ)-approximation of the number of linear extensions of random posets of varying density, as a function of the poset size n. The solid line shows the median and the shaded area the range of five independent instances per n.

7. Discussion

We have studied various approaches to estimate the number of linear extensions of a given partial order. We required controlled probabilistic approximation guarantees but set no requirement for provable running time upper bounds, whether in the worst case or on average over problem instances. While this viewpoint deviates from that commonly taken in theoretical computer science, it is well aligned with artifical intelligence research. Indeed, it invites discovering algorithms and intelligent heuristics that can yield significant com-

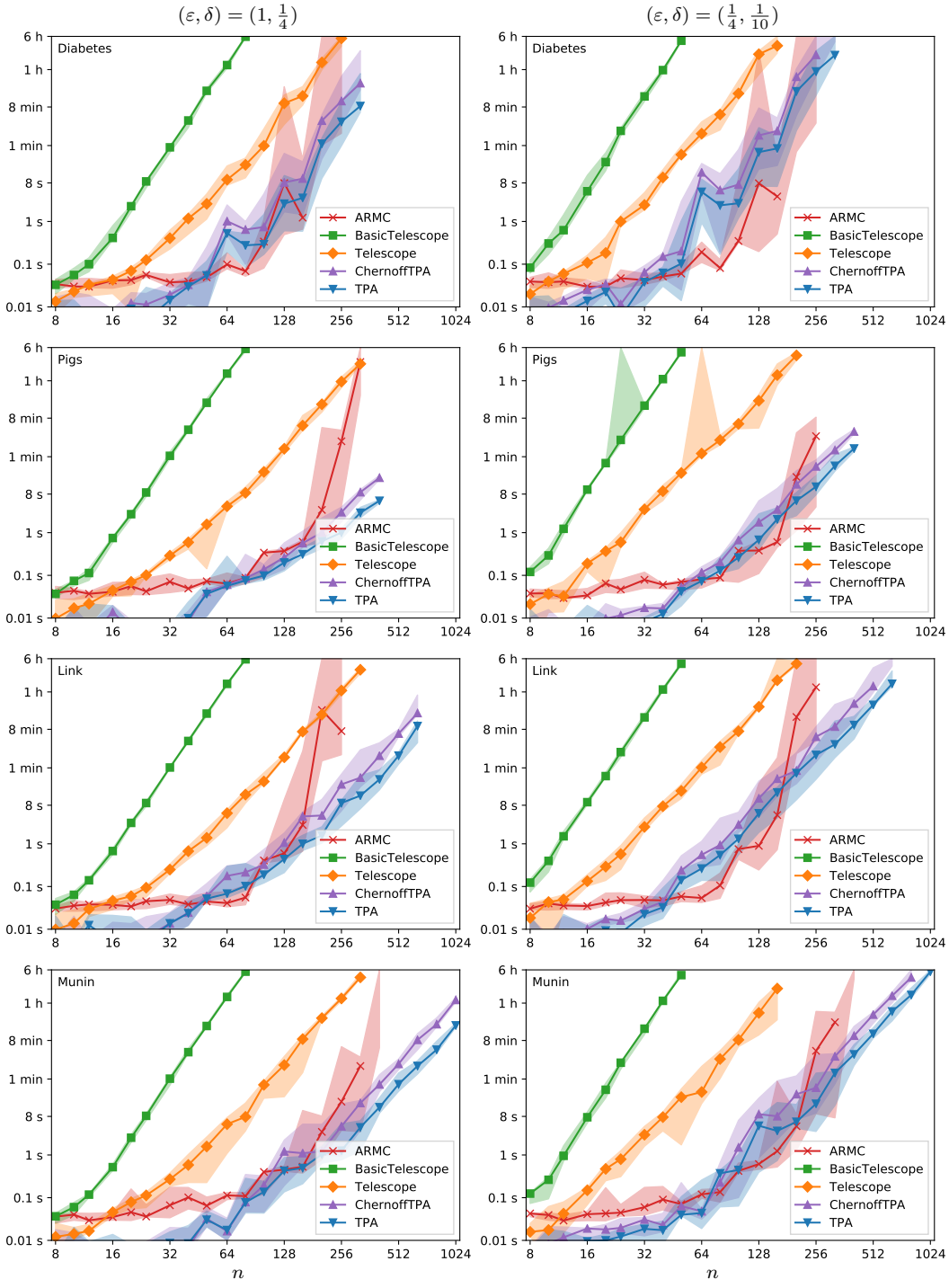


Figure 9: Empirical running times for (ε, δ)-approximation of the number of linear extensions of posets generated from Bayesian networks, for varying poset sizes n. The solid line shows the median and the shaded area the range of five independent instances per n.

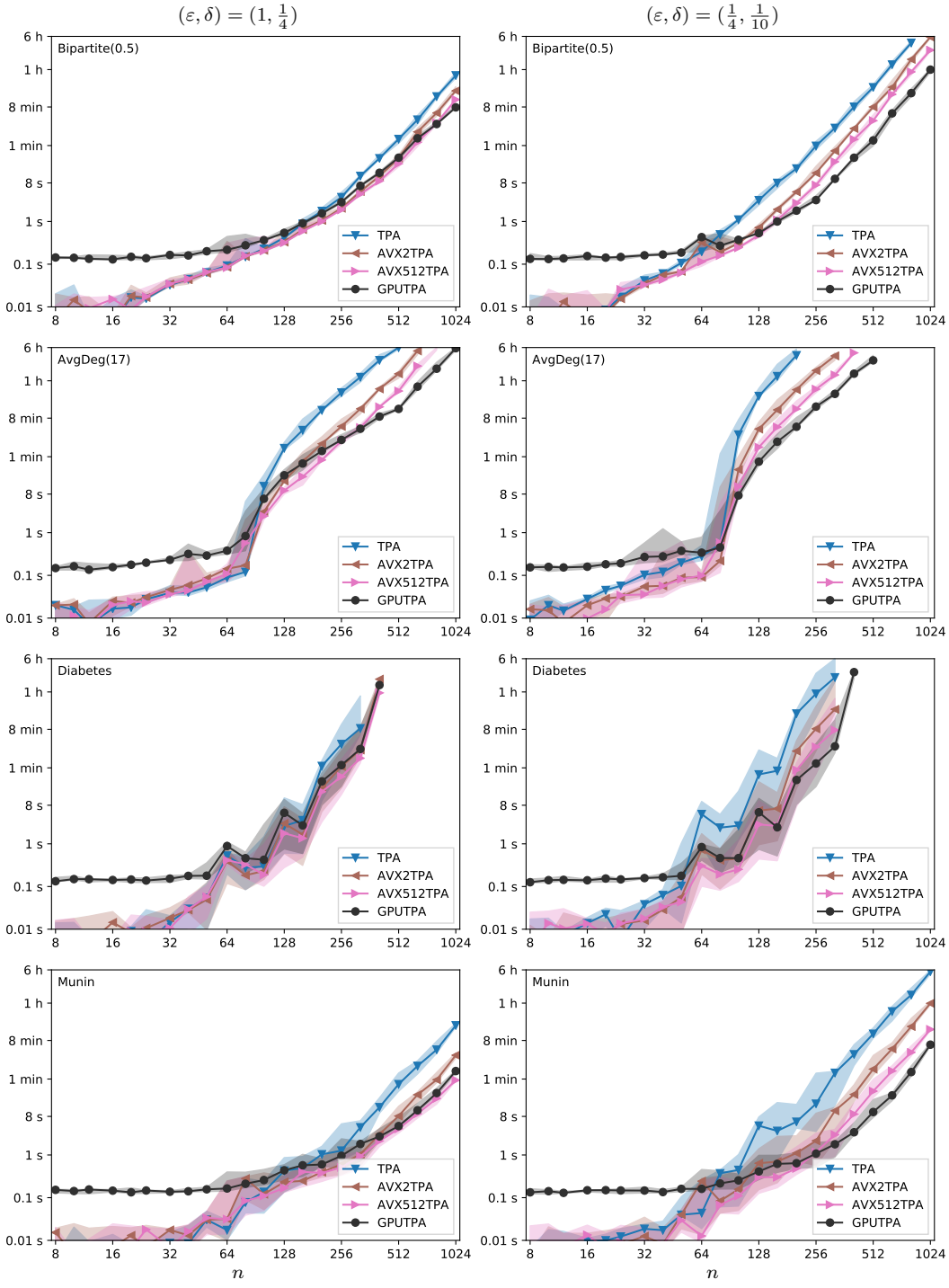


Figure 10: Empirical running times for (ε, δ)-approximation of the number of linear extensions of random posets of selected classes, as a function of the poset size n. The solid line shows the median and the shaded area the range of five independent instances per n.

putational savings in practice—with this respect counting problems are not different from decision and search problems. Even if we currently lack formal characterization of how the computational saving depends on the particular instance, empirically observed running times may lead to better analytic bounds in a longer run.

The most important theoretical findings in this work are perhaps (i) the new record in the worst-case running time bound, albeit by just a logarithmic factor, (ii) the enhancement of the generic TPA, (iii) and the somewhat specialized continuous relocation chain for exact sampling of linear extensions embedded into a continuous space. On a more conceptual level, our results reinforce the understanding that when solving hard problems in practice, worst-case asymptotic bounds are, at best, only partial measures of efficiency. Empirically we observed that the exponential ARMC scheme is often the fastest in a certain range of moderate-size instances, while on larger instances the fastest is the TPA variant that appears to scale polynomially, even though we could not prove any polynomial bounds.

There are multiple directions for future research. One intriguing question is to identify classes of instances for which we could prove analytic bounds for the TPA scheme, roughly matching the empirical observations. Or, how do the hard or the hardest instances for the TPA look like? To investigate this question, one could, for example, run a computer-assisted or automated heuristic search in the input space of partial orders. One could also simply extend the empirical study by collecting real-world instances and including various models of random instances: such an extensive empirical study is beyond the present work, which focuses on algorithmic results along with highly-optimized implementations. Having richer empirical data at hand, one could also consider the question of selecting the best algorithm for a given problem instance using machine learning (Leyton-Brown, Nudelman, & Shoham, 2009; Kerschke, Hoos, Neumann, & Trautmann, 2019).

While this paper focused on one concrete counting problem, some of the ideas and techniques might be applicable to other counting problems as well. In particular, it remains to be investigated whether our ideas could be successfully extended to the more general problem of approximating the volume of convex bodies, or vice versa, whether the advanced techniques recently developed for that problem (Lovász & Deák, 2012; Cousins & Vempala, 2016; Emiris & Fisikopoulos, 2018) are valuable also in the special case of counting linear extensions.

Acknowledgments

The authors would like to thank Kustaa Kangas and Teppo Niinimäki for their pivotal contributions and valuable collaboration for the preliminary work, which is the basis of the present article. The work was supported in part by the Academy of Finland, Grant 316771.

Appendix A. Deferred Proofs

This section presents some deferred proofs.

A.1 Proof of Proposition 10

Recall that we defined

$$q_d(\lambda) := Q(\lambda + d, \lambda) - Q(\lambda - d + 1, \lambda)$$

and that we have to show that $q_d(\lambda') \leq p' = \min\{p(\lambda) : 0 \leq \lambda \leq \lambda'\}$ for $d \geq 1$.

Our proof will use the lower incomplete gamma function

$$\gamma(s, x) := \Gamma(s) - \Gamma(s, x) = \int_0^x t^{s-1} \mathrm{e}^{-t} \mathrm{d}t \,.$$

Our first monotonicity result is well known; Gautschi (1998) attributes it to Tricomi (1950). We include a short proof for completeness.

Lemma 15. *For every $x > 0$, the function $Q(s, x)$ is increasing in $s > 0$.*

Proof. Fix $x > 0$ and $u > s > 0$. Since $\gamma(u, x) \leq x^{u-s}\gamma(s, x)$ while $\Gamma(u, x) \geq x^{u-s}\Gamma(s, x)$, we have $\gamma(u, x)/\Gamma(u, x) \leq \gamma(s, x)/\Gamma(s, x)$. This implies $Q(u, x) \geq Q(s, x)$. □

By the definition of q_d we thus have $q_d(\lambda) \leq p(\lambda)$ for all $\lambda \geq 0$. Next we show that q_d is a decreasing function.

Our second lemma concerns the monotonicity of $Q(s, x)$ when both parameters grow at the same rate. Whether we have an increasing or decreasing function depends crucially on which one the two parameters is larger.

Lemma 16. *For every $d \geq 1$, the function $q_d(x)$ is decreasing in $x > d - 1$.*

Proof. Consider first the second term of $q_d(x)$. Chojnacki (2008, Theorem 1) proved that $Q(s, s + b)$ is increasing in $s > 0$ for every $b \geq 0$. Substituting $s := x - b$ gives us that $Q(x - b, x)$ is increasing in $x > b$ for every $b \geq 0$.

Now, it suffices to show that $Q(x+b, x)$ is decreasing in $x > 0$ if $b \geq 1$. We give a straightforward extension of a proof of the case $b = 1$ by van de Lune (1975, Lemmas 3 and 4). Consider the term $\gamma(x + b, x)$. Let $s := x + b - 1$. The substitution $t := s - u\sqrt{s}$ gives us

$$\int_0^x \mathrm{e}^{-t} t^s \mathrm{d}t = \int_{(b-1)/\sqrt{s}}^{\sqrt{s}} \mathrm{e}^{-s+u\sqrt{s}} (s - u\sqrt{s})^s \sqrt{s} \, \mathrm{d}u = \sqrt{s} \left(\frac{s}{\mathrm{e}}\right)^s \int_{(b-1)/\sqrt{s}}^{\sqrt{s}} \mathrm{e}^{u\sqrt{s}} \Big(1 - \frac{u}{\sqrt{s}}\Big)^s \mathrm{d}u \,.$$

This representation is motivated by the formula

$$\Gamma(s + 1) = \left(\frac{s}{\mathrm{e}}\right)^s \sqrt{2\pi s} \cdot \mathrm{e}^{\theta(s)} \,,$$

where $\theta(s) := \int_0^\infty \left(\frac{1}{2} - \frac{1}{t} + \frac{1}{\mathrm{e}^t - 1}\right) \frac{\mathrm{e}^{-ts}}{t} \mathrm{d}t$ is Binet's function, which is decreasing in $s > 0$. Indeed, we obtain

$$1 - Q(x + b, x) = \frac{\gamma(x + b, x)}{\Gamma(x + b)} = \frac{\mathrm{e}^{-\theta(s)}}{\sqrt{2\pi}} \int_{(b-1)/\sqrt{s}}^{\sqrt{s}} \exp\left\{u\sqrt{s} + s \ln\left(1 - \frac{u}{\sqrt{s}}\right)\right\} \mathrm{d}u \,.$$

Since $b \geq 1$, the range of the integral grows with s. The integrand being nonnegative, it suffices to show that it is increasing in s for all u in the range. To this end, use the series expansion $\ln(1-z) = -\sum_{k=1}^{\infty} z^k/k$ to conclude that

$$u\sqrt{s} + s\ln\left(1 - \frac{u}{\sqrt{s}}\right) = -\frac{u^2}{2} - \sum_{k=3}^{\infty} \frac{u^k}{k} s^{-k/2+1}$$

is increasing in $s \geq 0$ for any $u \geq 0$. □

A.2 Proof of Proposition 13

Recall that we have the prove the following:

(i) The relation $\sqsubset$ is a partial order on Ω.

(ii) We have $\mathbf{0}, \mathbf{1} \in \Omega$ and $\mathbf{0} \sqsubseteq x \sqsubseteq \mathbf{1}$ for all $x \in \Omega$.

(iii) If $x \sqsubseteq y$, then $\phi(x,i,u) \sqsubseteq \phi(y,i,u)$ for all $i \in \{1,2,\ldots,n\}$ and $u \in [0,1]$.

For convenience, also recall the definitions of Ω and $\sqsubset$:

$$\Omega := \left\{x \in [0,1]^n : x_i - x_j \leq s_{ij} \text{ for all } 1 \leq i < j \leq n\right\},$$

where the numbers $s_{ij} \geq 0$ are given as input;

$$\sqsubset := \left\{(x,y) \in \Omega \times \Omega : x \neq y \text{ and } x_i \leq y_i \text{ for all } 1 \leq i \leq n\right\}.$$

We consider each claim in turn.

(i) The relation $\sqsubset$ is irreflexive by the definition. To see that it is transitive, suppose $x \sqsubset y$ and $y \sqsubset z$. Let $1 \leq i \leq n$. We have $x_i \leq y_i$ and $y_i \leq z_i$, whence $x_i \leq z_i$. Since $x_j < y_j$ for some dimension j, we must have $x_j < z_j$ and thus $x \neq z$. This implies $x \sqsubset z$.

(ii) It follows from the assumption $s_{ij} \geq 0$ that $\mathbf{0}, \mathbf{1} \in \Omega$, since $0 - 0 \leq 0$ and $1 - 1 \leq 0$. Now, let $x \in \Omega \setminus \{\mathbf{0}, \mathbf{1}\}$. Thus $x_i \in [0,1]$ for each $1 \leq i \leq n$. Since $x \neq \mathbf{0}$, we have $0 < x_i$ for some i, implying $\mathbf{0} \sqsubset x$. Since $x \neq \mathbf{1}$, we have $x_j < 1$ for some j, implying $x \sqsubset \mathbf{1}$.

(iii) Let $x \sqsubseteq y$. Let $i \in \{1,2,\ldots,n\}$ and $u \in [0,1]$. To prove that $\phi(x,i,u) \sqsubseteq \phi(y,i,u)$ it suffices to show that $\phi(x,i,u)_i \sqsubseteq \phi(y,i,u)_i$, since the map ϕ does not change the jth coordinate of its first argument for $j \neq i$. For each $z \in \{x,y\}$, let

$$l_z := \max\left\{\{0\} \cup \{z_j - s_{ji} : 1 \leq j < i\}\right\},\ r_z := \min\left\{\{1\} \cup \{z_j + s_{ij} : i < j \leq n\}\right\}.$$

By the definition of Ω, we have $z_i \in [l_z, r_z]$, and if $u \in [l_z, r_z]$, then $\phi(z,i,u)_i = u$ and else $\phi(z,i,u)_i = z_i$. Because $x \sqsubseteq y$, we also have $l_x \leq l_y$ and $r_x \leq r_y$.

We now show that $\phi(x,i,u)_i \sqsubseteq \phi(y,i,u)_i$ in cases. First, if $u \in [l_y, r_x]$, then $u \in [l_x, r_x] \cap [l_y, r_y]$ and thus $\phi(x,i,u)_i = u = \phi(y,i,u)_i$. Second, if $u < l_y$ (case $u > r_x$ is symmetric), then $\phi(x,i,u)_i$ equals either x_i or u. We have $x_i \leq y_i$, as $x \sqsubseteq y$, and $u < l_y \leq y_i$, as $y_i \in [l_y, r_y]$. Combining these yields $\phi(x,y,u)_i \leq y_i = \phi(y,i,u)_i$.

A.3 Proof of Theorem 14

We show that Monotone CFTP for the continuous relocation chain outputs a sample from the uniform distribution on Ω in finite expected time.

Let $d := 2n - 1$. For all positive integers k, define A_k as the event

$$i_t = n - |n - kd - t - 1| \quad \text{and} \quad i_t - 1 \leq u_t n \leq i_t \quad \text{for all } -kd \leq t < -kd + d\,.$$

In words, A_k occurs when for d time points t starting from $-kd$, the dimension i_t takes the values $1, 2, \ldots, n, n-1, \ldots, 2, 1$ in this order and the respective coordinate u_t is in the interval from $(i_t - 1)/n$ to i_t/n.

Suppose A_k occurs in some iteration $T \geq kd$. We claim that then $L_{-kd+d} = R_{-kd+d}$, and so the algorithm terminates after that iteration. To see that the claim holds, consider first the first n steps, $i := i_t$ taking the values $1, 2, \ldots, n$, with the respective coordinate $u := u_t \in [(i-1)/n, i/n]$. By induction on i, we can show that after the ith step, the ith coordinate x_i of the state x, whether L_t or R_t, is at most i/n for each $i = 1, 2, \ldots, n$. Indeed, if the proposed move to u is accepted, then this clearly holds, as x_i is not changed in other steps; otherwise the move is not accepted due to a conflict with some x_j, $j > i$, implying that u is larger than x_i, whence $x_i \leq i/n$. Now turn to the latter $n - 1$ steps, $i := i_t$ taking the values $n-1, n-2, \ldots, 1$, again with the respective coordinate $u := u_t \in [(i-1)/n, i/n]$. In these steps, every proposal is accepted, since for the ith coordinate x_i, we already have $x_j \geq (j-1)/n$ for all $j > i$ (by induction), but also $x_j \leq j/n \leq (i-1)/n$ for all $j < i$ (by the first n steps). So the state after the steps does not depend on the initial state (whether L_t or R_t), completing the proof of the claim.

The events $A_1, A_2, \ldots$ are mutually exclusive and occur with the same probability $p = (1/n^2)^d = n^{2-4n} > 0$. Let K be the smallest k such that the event A_k occurs. Because the algorithm doubles T on every iteration, at least half of the time is spent in the last iteration, and thus the running time is $O(K)$ when viewing n as a constant. Since K is geometrically distributed with parameter p, its expected value is $1/p < \infty$.

References

Atkinson, M. D. (1990). On computing the number of linear extensions of a tree. *Order*, *7*(1), 23–25.

Banks, J., Garrabrant, S., Huber, M., & Perizzolo, A. (2018). Using TPA to count linear extensions. *Journal of Discrete Algorithms*, *51*, 1–11.

Brightwell, G., & Winkler, P. (1991). Counting linear extensions. *Order*, *8*(3), 225–242.

Bubley, R., & Dyer, M. (1999). Faster random generation of linear extensions. *Discrete Mathematics*, *201*(1), 81–88.

Chakraborty, S., Meel, K. S., & Vardi, M. Y. (2013). A scalable approximate model counter. In Schulte, C. (Ed.), *Proceedings of the Principles and Practice of Constraint Programming - 19th International Conference, CP 2013*, Vol. 8124 of *Lecture Notes in Computer Science*, pp. 200–216. Springer.

Chojnacki, W. (2008). Some monotonicity and limit results for the regularised incomplete gamma function. *Annales Polonici Mathematici*, *94*(3), 283–291.

Cousins, B., & Vempala, S. S. (2016). A practical volume algorithm. *Mathematical Programming Computation*, *8*(2), 133–160.

Dagum, P., Karp, R., Luby, M., & Ross, S. (2000). An optimal algorithm for Monte Carlo estimation. *SIAM Journal on Computing*, *29*(5), 1484–1496.

De Loof, K., De Meyer, H., & De Baets, B. (2006). Exploiting the lattice of ideals representation of a poset. *Fundamenta Informaticae*, *71*(2–3), 309–321.

Dilworth, R. P. (1950). A decomposition theorem for partially ordered sets. *Annals of Mathematics*, *51*(1), 161–166.

Dor, D., & Zwick, U. (1999). Selecting the median. *SIAM Journal on Computing*, *28*(5), 1722–1758.

Dyer, M., Frieze, A., & Kannan, R. (1991). A random polynomial time algorithm for approximating the volume of convex bodies. *Journal of the ACM*, *38*(1), 1–17.

Emiris, I. Z., & Fisikopoulos, V. (2018). Practical polytope volume approximation. *ACM Transactions on Mathematical Software*, *44*(4), 38:1–38:21.

Ermon, S., Gomes, C. P., Sabharwal, A., & Selman, B. (2013). Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, Vol. 28 of *JMLR Workshop and Conference Proceedings*, pp. 334–342. JMLR.org.

Gajek, L., Niemiro, W., & Pokarowski, P. (2013). Optimal Monte Carlo integration with fixed relative precision. *Journal of Complexity*, *29*(1), 4–26.

Garey, M., Johnson, D., & Stockmeyer, L. (1976). Some simplified NP-complete graph problems. *Theoretical Computer Science*, *1*(3), 237–267.

Gautschi, W. (1998). The incomplete gamma functions since Tricomi. In *Tricomi's Ideas and Contemporary Applied Mathematics, Atti Convegni Lincei, n. 147*, pp. 203–237. Accademia Nazionale dei Lincei, Rome.

Gomes, C. P., Sabharwal, A., & Selman, B. (2006). Model counting: A new strategy for obtaining good bounds. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI 2006*, pp. 54–61.

Harviainen, J., Röyskö, A., & Koivisto, M. (2021). Approximating the permanent with deep rejection sampling. In *Advances in Neural Information Processing Systems 34*, pp. 213–224. Curran Associates, Inc.

Huber, M. (2006). Fast perfect sampling from linear extensions. *Discrete Mathematics*, *306*(4), 420–428.

Huber, M. (2014). Near-linear time simulation of linear extensions of a height-2 poset with bounded interaction. *Chicago Journal of Theoretical Computer Science*, *Article 03*, 1–16.

Huber, M. (2017). A Bernoulli mean estimate with known relative error distribution. *Random Structures & Algorithms*, *50*(2), 173–182.

Huber, M., & Schott, S. (2010). Using TPA for Bayesian inference. *Bayesian Statistics*, *9*, 257–282.

Jerrum, M., & Sinclair, A. (1997). *Approximation Algorithms for NP-hard Problems*, chap. The Markov Chain Monte Carlo Method: An Approach to Approximate Counting and Integration, pp. 482–520. PWS.

Jerrum, M., Sinclair, A., & Vigoda, E. (2004). A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, *51*(4), 671–697.

Jerrum, M., Valiant, L. G., & Vazirani, V. V. (1986). Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, *43*, 169–188.

Kangas, K., Hankala, T., Niinimäki, T., & Koivisto, M. (2016). Counting linear extensions of sparse posets. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence, IJCAI 2016*, pp. 603–609.

Karzanov, A., & Khachiyan, L. (1991). On the conductance of order Markov chains. *Order*, *8*(1), 7–15.

Kerschke, P., Hoos, H. H., Neumann, F., & Trautmann, H. (2019). Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, *27*(1), 3–45.

Kuck, J., Dao, T., Rezatofighi, H., Sabharwal, A., & Ermon, S. (2019a). Approximating the permanent by sampling from adaptive partitions. In *Advances in Neural Information Processing Systems 32*, pp. 8860–8871. Curran Associates, Inc.

Kuck, J., Dao, T., Zhao, S., Bartan, B., Sabharwal, A., & Ermon, S. (2019b). Adaptive hashing for model counting. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence, UAI 2019*.

Levin, D., & Peres, Y. (2017). *Markov Chains and Mixing Times* (2nd edition). Providence: American Mathematical Society.

Leyton-Brown, K., Nudelman, E., & Shoham, Y. (2009). Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM*, *56*(4), 22:1–22:52.

Lovász, L., & Deák, I. (2012). Computational results of an $O^*(n^4)$ volume algorithm. *European Journal on Operations Research*, *216*(1), 152–161.

Mannila, H., & Meek, C. (2000). Global partial orders from sequential data. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 161–168.

Mitzenmacher, M., & Upfal, E. (2005). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, USA.

Morton, J., Pachter, L., Shiu, A., Sturmfels, B., & Wienand, O. (2009). Convex rank tests and semigraphoids. *SIAM Journal on Discrete Mathematics*, *23*(3), 1117–1134.

Muise, C., Beck, J. C., & McIlraith, S. A. (2016). Optimal partial-order plan relaxation via MaxSAT. *Journal of Artificial Intelligence Research*, *57*, 113–149.

Niinimäki, T., Parviainen, P., & Koivisto, M. (2016). Structure discovery in Bayesian networks by sampling partial orders. *Journal of Machine Learning Research*, *17*(57), 1–47.

Pavan, A., Vinodchandran, N. V., Bhattacharyya, A., & Meel, K. S. (2023). Model counting meets distinct elements. *Communications of the ACM*, *66*(9), 95–102.

Peczarski, M. (2004). New results in minimum-comparison sorting. *Algorithmica*, *40*(2), 133–145.

Propp, J. G., & Wilson, D. B. (1996). Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures & Algorithms*, *9*(1–2), 223–252.

Roberts, G. O., & Rosenthal, J. S. (2004). General state space Markov chains and MCMC algorithms. *Probability Surveys*, *1*, 20–71.

Talvitie, T., Kangas, K., Niinimäki, T. M., & Koivisto, M. (2018a). Counting linear extensions in practice: MCMC versus exponential Monte Carlo. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 1431–1438. AAAI Press.

Talvitie, T., Kangas, K., Niinimäki, T. M., & Koivisto, M. (2018b). A scalable scheme for counting linear extensions. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*, pp. 5119–5125. ijcai.org.

Talvitie, T., Niinimäki, T. M., & Koivisto, M. (2017). The mixing of Markov chain on linear extensions in practice. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, pp. 524–530. ijcai.org.

Thurley, M. (2006). sharpSAT – counting models with advanced component caching and implicit BCP. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, SAT 2006*, pp. 424–429.

Tricomi, F. G. (1950). Sulla funzione gamma incompleta. *Annali di Matematica Pura ed Applicata*, *4*(31), 263–279.

Valiant, L. G. (1979). The complexity of computing the permanent. *Theoretical Computer Science*, *8*, 189–201.

van de Lune, J. (1975). A note on Euler's (incomplete) Γ-function. Tech. rep. ZN 61, Stichting Mathematisch Centrum, Amsterdam.

Wallace, C. S., Korb, K. B., & Dai, H. (1996). Causal discovery via MML. In *Proceedings of the Thirteenth International Conference in Machine Learning, ICML 1996*, pp. 516–524. Morgan Kaufmann.

Wilson, D. B. (2004). Mixing times of lozenge tiling and card shuffling Markov chains. *Annals of Applied Probability*, *14*(1), 274–325.