A Scalable Scheme for Counting Linear Extensions

Topi Talvitie¹, Kustaa Kangas², Teppo Niinimäki², Mikko Koivisto¹
¹ University of Helsinki
² Aalto University

totalvit@cs.helsinki.fi, juho-kustaa.kangas@aalto.fi, teppo.niinimaki@aalto.fi, mkhkoivi@cs.helsinki.fi

Abstract
Counting the linear extensions of a given partial order not only has several applications in artificial intelligence but also represents a hard problem that challenges modern paradigms for approximate counting. Recently, Talvitie et al. (AAAI 2018) showed that an exponential time scheme beats the fastest known polynomial time schemes in practice, even if allowing hours of running time. Here, we present a novel scheme, relaxation Tootsie Pop, which in our experiments exhibits polynomial characteristics and significantly outperforms previous schemes. We also instantiate state-of-the-art model counters for CNF formulas; two natural encodings yield schemes that, however, are inferior to the more specialized schemes.

1 Introduction
The generic problem of computing the number of objects in an implicitly given (multi)set plays a fundamental role in artificial intelligence research. On the one hand, various instantiations of the problem appear in applications, particularly in probabilistic inference. On the other hand, the computational hardness of such problems calls for heuristics that are efficient in practice and able to beat the known, perhaps pessimistic, complexity upper bounds. In particular, researchers have sought alternatives to the popular Markov chain Monte Carlo (MCMC) method, which often does not provide practical means for controlling the accuracy and time requirement. For example, there has been recent interest in approximate counting methods that leverage the power of modern SAT solvers [Gomes et al., 2006; Chakraborty et al., 2015]. Generally, it remains largely an open question, what sort of approximate counting schemes are the most efficient in practice.

This paper focuses on the classic problem of counting linear extensions: given a partial order \(\prec\) (i.e., an irreflexive and transitive binary relation) on \(n\) elements, compute the number of those total orders on the elements that are superset of \(\prec\). The problem has applications, for example, in sequence analysis [Mannila and Meek, 2000], sorting [Peczarski, 2004], preference reasoning [Łukasiewicz et al., 2014], convex rank tests [Morton et al., 2009], partial order plans [Muise et al., 2016], and learning graphical models [Wallace et al., 1996; Niinimäki et al., 2016]. The problem is \#P-hard [Brightwell and Winkler, 1991] but admits a fully polynomial time approximation scheme based on MCMC [Dyer et al., 1991]. The best known asymptotic bounds for the expected running time are \(O\left(\epsilon^{-2}n^3\log^2 \ell \log n\right)\) [Banks et al., 2017] and \(O\left(\epsilon^{-2}n^5\log^2 n\right)\) [Talvitie et al., 2018], where \(\ell\) is the number of linear extensions and \(\epsilon\) the allowed relative error.

The large degree of the polynomial bounds raises the question of the practical value of the schemes. This question was recently addressed by Talvitie et al. [2018] who studied variants of the schemes empirically on benchmark instances. Although their polynomial-time scheme, enhanced by a number of heuristic tricks, was superior to prior schemes, it was inferior to an exponential-time scheme they coin adaptive relaxation Monte Carlo (ARMC), even when allowing hours of running time. Their finding suggests that the MCMC method, though asymptotically faster, may not lend itself to implementations that are competitive in practice.

In this paper, we refute that conclusion. We present a novel MCMC scheme that stems from modifying and combining ideas scattered in previous works. Like Banks et al. [2017] we apply the generic Tootsie Pop algorithm (TPA) of Huber and Schott [2010], however, with two major differences: First, we employ a different continuous embedding, which also requires us to design a novel sampler—our perfect sampler is the main technical contribution of this work. Second, unlike all previous efficient reductions from counting to sampling, which add constraints one after another to the input partial order until it becomes a linear extension, our reduction constructs intermediate problems between the input partial order and its relaxation we obtain by removing some constraints; this is inspired by the ARMC scheme. We demonstrate empirically using the collection of benchmark instances of Talvitie et al. [2018] that our scheme, which we call relaxation Tootsie Pop, is superior to all previous schemes. Moreover, its running time appears to grow polynomially, even if we currently have no proof for a polynomial bound.

In addition, we make an attempt to apply state-of-the-art SAT solver based schemes for counting linear extensions. Specifically, we instantiate the algorithms of Chakraborty et al. [2016], Thurley [2006], and Lagiez and Marquis [2017] using two alternative, natural problem encodings.

Section 2 begins by formulating the problem of counting linear extensions and reviewing existing approaches; we also
Figure 1: The cover graph of a poset (left) and the cover graph of one linear extension of the poset, shown in emphasized arcs (right).

give our instantiations of the SAT solver based method. Section 3 presents the relaxation Tootsie Pop scheme, and Section 4 the required perfect sampler. The experiments are summarized in Section 5. We conclude in Section 6 by discussing the significance of our findings for counting linear extensions in particular and for approximate counting in general.

2 The Problem and Existing Techniques

Let \( A \) be a finite set of \( n \) elements and let \( \prec \) be an irreflexive and transitive binary relation on \( A \). We say that \( \prec \) is a partial order on \( A \) and call the pair \( P = (A, \prec) \) a partially ordered set (poset). The elements \((x, y) \in \prec\) are called order constraints of \( P \) and denoted simply \( x \prec y \). We depict \( P \) with its cover graph, a directed acyclic graph where the vertex set is \( A \) and the arc set is the transitive reduction of \( \prec \) (Fig. 1). If for all distinct \( x, y \in A \) we have either \( x \prec y \) or \( y \prec x \), we say that \( P \) is a linear order.

Let \( \prec' \) be another partial order on the set \( A \). If \( \prec \subseteq \prec' \), we say that \( P' = (A, \prec') \) is an extension of \( P \) or, conversely, that \( P \) is a relaxation of \( P' \). If \( P' \) is an extension of \( P \) and also a linear order, we say that \( P' \) is a linear extension of \( P \) (Fig. 1). We denote by \( \mathcal{L}(P) \) the set of linear extension of \( P \) and by \( \ell(P) := |\mathcal{L}(P)| \) the number of its linear extensions.

For a quantity \( q \), an estimate \( \tilde{q} \), and parameters \( \epsilon, \delta > 0 \), we say that \( \tilde{q} \) is an \((\epsilon, \delta)\)-approximation of \( q \) if

\[
\Pr\left((1 + \epsilon)^{-1} < \frac{\tilde{q}}{q} < 1 + \epsilon\right) \geq 1 - \delta.
\]

We consider the problem of giving an \((\epsilon, \delta)\)-approximation of \( \ell(P) \) for a given poset \( P \) and parameters \( \epsilon, \delta \).

We begin with an overview of known schemes for approximate counting.

2.1 Telescopic Product

Brightwell and Winkler [1991] presented an approximate linear extension counting scheme based on uniform sampling of linear extensions. The method forms a sequence of posets \((P_0, P_1, \ldots, P_k)\) starting from the original poset \( P_0 = P \) and iteratively adding more constraints until reaching a linear order \( P_k \). Thus \( \mathcal{L}(P_0) \supset \mathcal{L}(P_1) \supset \cdots \supset \mathcal{L}(P_k) \) and we can write \( \ell(P) \) as a telescopic product \( \prod_{i=1}^{k} \ell(P_i)/\ell(P_{i-1}) \). Each factor \( \ell(P_i)/\ell(P_{i-1}) \) is estimated by drawing samples uniformly from \( \mathcal{L}(P_{i-1}) \) and taking the fraction of the samples that are also elements of \( \mathcal{L}(P_i) \).

A comparison sorting based approach results in a sequence of posets of length \( k = O(n \log n) \), and drawing in total \( O(n^2 \log^2 n e^{-2 \log \delta^{-1}}) \) linear extension samples is sufficient for obtaining an \((\epsilon, \delta)\)-approximation [Talvitie et al., 2017]. Using Huber’s [2006] \( O(n^3 \log n) \)-expected-time algorithm—asymptotically the fastest know sampler—we obtain an \( O(n^3 \log^2 n e^{-2 \log \delta^{-1}}) \) expected total running time. Talvitie et al. [2018] improved this bound by a factor of \( \log n \) by exploiting the special structure of the posets in the sequence. They also achieved a significant performance improvements in practice by switching to a Gibbs sampler of Huber [2014].

2.2 Tootsie Pop Algorithm

The Tootsie Pop algorithm (TPA) [Huber and Schott, 2010] is a scheme for computing the ratio of the measures of two nested sets \( A' \supset A'' \) in a continuous space under certain conditions. Specifically, there must exist a family of intermediate sets \{\( A_{\beta} \mid \beta \in \mathbb{R} \)\} where \( A' = A_{\beta'} \) and \( A'' = A_{\beta''} \), the family must be nested in the sense that \( A_{\beta_1} \subset A_{\beta_2} \) for any \( \beta_1 < \beta_2 \), and the measure \( \mu(A_{\beta}) \) must be a continuous function of \( \beta \).

The scheme works by iteratively constructing a random sequence \( (\beta_i)_{i=0}^n \) that starts from \( \beta_0 = \beta' \), and each subsequent element \( \beta_{i+1} \) is computed by drawing \( X_i \) uniformly at random from \( A_{\beta_i} \) and setting \( \beta_{i+1} = \min\{\beta \in \mathbb{R} \mid X_i \in A_{\beta}\} \) (whether it is easy to draw \( X_i \) and compute \( \beta_{i+1} \) depends on the set family). Now the number of iterations \( \min\{i \geq 0 \mid \beta_{i+1} \leq \beta''\} \) required to reach \( A'' \) is Poisson distributed with expected value \( r = \ln(\mu(A')/\mu(A'')) \), and we can estimate the log-ratio by averaging over multiple independent runs. In total, \( O(r^2 e^{-2 \log \delta^{-1}}) \) calls to the sampling subroutine are sufficient for an \((\epsilon, \delta)\)-approximation.

Banks et al. [2017] apply TPA to linear extension counting by transforming the discrete counting problem into a measure computation problem in a continuous space. This is achieved by affixing a continuous dimension to the set of linear extensions: \( A_{\beta} = \bigcup_{L \in \mathcal{L}(P)} \{L \times [0, w_L(\beta)]\} \) where \( \beta \mapsto w_L(\beta) \) is a continuous function with values in range \([0,1]\). The weights \( w_L(\beta) \) are chosen carefully such that uniform sampling from \( A_{\beta} \) is possible and \( A' = \bigcup_{L \in \mathcal{L}(P)} \{L \times [0,1]\} \), where \( L_{\text{home}} \) is an arbitrary linear extension of \( P \). Then \( \ell(P) = \mu(A')/\mu(A'') \) can be estimated by TPA. The sampling algorithm runs in \( O(n^3 \log n) \) time, which means that the \((\epsilon, \delta)\)-approximation algorithm for counting the number of linear extensions runs in \( O(n^3 \log^2 \ell(P) \log ne^{-2 \log \delta^{-1}}) \) expected time.
is fast. On the other hand, letting \( R \) deviate from \( P \) allows
a smaller \( k \) and thus makes the exact computation of \( \ell(R) \)
faster. An adaptive procedure is used to choose \( R \) so as to
obtain a good tradeoff between these two phases.

### 2.4 Boolean Model Counting

The Boolean model counting problem, #SAT, asks for the
number of satisfying truth assignments to a given Boolean
formula. Given a poset \( P = (A, \prec_A) \), we encode the
problem of computing \( \ell(P) \) in #SAT by producing a Boolean
formula \( \phi \) (in conjunctive normal form) whose satisfying
assignments are bijectively mapped to \( \mathcal{L}(P) \). We
investigate two natural encodings and evaluate them by employing
existing model counting software, specifically, implementa-
tions of two exact model counter [Thurley, 2006; Lagniez and
Marquis, 2017] and an algorithm by Chakraborty et al. [2016],
which outputs an \((\epsilon, \delta)\)-approximation for \( \ell(P) \) in \( O(\log^2\ell(P)/\epsilon \cdot \log\delta^{-1}) \)
calls to a generalized order constraint sampling subroutine.

In the first encoding, SAT #1, we let \( \mathcal{L} \) contain a variable
\( v_{xy} = \neg v_{yx} \) for each pair of distinct elements \( x, y \in A \)
to represent the order between \( x \) and \( y \); if \( v_{xy} \) is true, then \( x \)
precedes \( y \) in the linear extension, otherwise \( y \) precedes \( x \)
\((n(n-1)/2 \text{ variables})\). Let \( \mathcal{L} \) further contain the clauses
\begin{enumerate}
\item \( v_{xy} \lor \neg v_{x'y} \lor v_{xx} \) for all distinct \( x, y, z \in A \),
\item \( v_{xy} \) for all \( x, y \in A \) such that \( x \prec_A y \),
\end{enumerate}
where clauses (i) enforce transitivity and clauses (ii) guarantee
that the corresponding linear order is an extension of \( P \).
The total number of clauses is \( n(n-1)(n-2) + c \), where \( c \) is
the number of edges in the cover graph of \( P \).

In the second encoding, SAT #2, we let \( \mathcal{L} \) contain a variable
\( v_{xp} \) for each \( x \in A \) and \( p \in [n] \) with the interpretation
that \( v_{xp} \) is true if and only if \( x \) is in position \( p \) in the linear
extension \( n^2 \text{ variables})\). Let \( \mathcal{L} \) contain the clauses
\begin{enumerate}
\item \( v_{x'p} \) for all \( x \in A \) (each element has a position),
\item \( \bigvee_{x \in A} v_{xp} \) for all \( p \in [n] \) (each position has an element),
\item \( \neg v_{xp} \lor \neg v_{x'q} \) for all \( x \in A \) and \( p, q \in [n] \) such that
\( p \neq q \) (no element is in two positions),
\item \( \neg v_{xp} \lor \neg v_{x'q} \) for all \( p \in [n] \) and \( x, y \in A \) such that \( x \neq y \)
(no position has two elements),
\item \( \neg v_{x'y} \lor \neg v_{yp} \) for all \( x, y \in A \) and \( p, q \in [n] \) such that
\( x \prec_A y \) and \( p < q \),
\end{enumerate}
where clauses (i–iv) guarantee that a satisfying assignment
maps each element to exactly one position and vice versa,
and clauses (v) enforce compliance with \( P \). The total number
of clauses is \( 2n + 2n^2(n-1) + cn(n-1)/2 \).

### 3 Relaxation Tootsie Pop

In this section, we present a new linear extension counting
method based on TPA [Huber and Schott, 2010]. In order to
translate the counting problem into the continuous space, we use
the same embedding as the linear extension Gibbs sam-
pler due to Huber [2014]: For a poset \( P = (A, \prec_A) \), each
point \( p \in [0, 1]^A \) specifies a position \( p_x \in [0, 1] \) for every
element \( x \in A \), and every order constraint \( x \prec_A y \) in \( P \) maps
to a numerical constraint \( p_x \leq p_y \). The domain \( A \subseteq [0, 1]^A \)
is defined as the set of points satisfying the numerical con-
straints. The linear order \( f(p) = (A, \prec_A') \) corresponding to
a point \( p \in A \) is obtained from the ordering of the positions:
\( x \prec_A' y \) if and only if \( p_x < p_y \) (we can choose arbitrarily
the cases where \( p_x = p_y \) for \( x \neq y \) since their measure in
\( A \) is zero). Consequently, if we sample \( p \in A \) uniformly at
random, the distribution of \( f(p) \) in \( \mathcal{L}(P) \) is also uniform.

Similarly to the ARMC method [Talvitie et al., 2018],
we count the linear extensions of \( P = (A, \prec_A) \) by estimating
\( \ell(R)/\ell(P) \) where \( R = (A, \prec') \) is a relaxation of \( P \)
for which the number of linear extensions \( \ell(R) \) can be computed
quickly. However, instead of direct sampling, we use TPA
for estimating the ratio, as illustrated in Figure 2. This has
the benefit that the required number of samples for TPA is
only polylogarithmic in the ratio \( \ell(R)/\ell(P) \), compared to
the linear growth in ARMC.

We define the family of intermediate sets required by TPA
as follows:

\[
\mathcal{A}_\beta = \left\{ p \in [0, 1]^A \mid \begin{array}{l}
x \prec_A' y \Rightarrow p_x - p_y \leq 0, \\
x \prec_A y \Rightarrow p_x - p_y \leq \beta
\end{array} \right\}.
\]  \hspace{1cm} (1)

As a result, the ratio \( \ell(R)/\ell(P) \) is equal to the correspond-
ing ratio of measures \( \mu(\mathcal{A}_\beta)/\mu(\mathcal{A}_0) \). In Section 4 we present
an algorithm for uniform sampling from \([0, 1]^A \) in the pre-

cence of generalized order constraints, of which the sets \( \mathcal{A}_\beta \)
are a special case. For now, we assume that we can sample
from \( \mathcal{A}_\beta \) and thus the requirements of TPA introduced in
Section 2.2 are satisfied, yielding the following result:

**Theorem 1.** Given a relaxation \( R \) for \( P \) with \( \ell(R) \) known,
The Relaxation Tootsie Pop algorithm computes an \((\epsilon, \delta)\)-
approximation for \( \ell(P) \) in \( O(\log^2(\ell(R)/\ell(P)) / \epsilon^2 \log \delta^{-1}) \)
calls to a generalized order constraint sampling subroutine.
3.1 Relaxation Finding

Choosing the empty relaxation \( R = (A, \emptyset) \) with \( \ell(R) = |A|! \) is always valid, but finding a relaxation where \( \ell(R) \) is closer to \( \ell(P) \) will make the algorithm run faster. Even though Talvitie et al. [2018] investigated relaxation finding heuristics for the ARMC method, their heuristics rely heavily on exponential exact counting algorithms. Investing that much time to relaxation finding does not pay off in our case: the gains are much smaller due to the polylogarithmic dependence of the running time on the ratio \( \ell(R)/\ell(P) \). For that reason, we will use faster heuristics to find reasonable relaxations.

Our heuristics stem from the observation that for many special classes of posets \( P = (A, \prec) \), 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, i.e., posets \( P' = (A', \prec) \) where \( A' \subseteq A \). First we check if the poset is already a member of such a special class:

1. If \( A \) can be partitioned into sets \( A_1 \) and \( A_2 \) that are unrelated in the sense that for all \( a_1 \in A_1 \) and \( a_2 \in A_2 \), \( a_1 \not\prec a_2 \) and \( a_2 \not\prec a_1 \), then we recursively find relaxations \( R_1 = (A_1, \prec_1) \) and \( R_2 = (A_2, \prec_2) \) for the respective subposets, and combine them into relaxation \( R = (A, \prec_1 \cup \prec_2) \) where \( \ell(R) = \ell(R_1) \ell(R_2) \).

2. If \( A \) can be partitioned into sets \( A_1 \) and \( A_2 \) that are ordered in the sense that for all \( a_1 \in A_1 \) and \( a_2 \in A_2 \), \( a_1 \prec a_2 \), then similarly to the previous part we combine the relaxations for the subposets into relaxation \( R = (A, \prec_1 \cup \prec_2) \) where \( \ell(R) = \ell(R_1) \ell(R_2) \).

3. If the cover graph of the poset \( P \) is a tree, then we use \( P \) as the relaxation and count its linear extensions using the \( O(n^2) \) algorithm due to Atkinson [1990].

4. If any exact linear extension counter finishes computing \( \ell(P) \) within a short time limit, we can use \( P \) as the relaxation. We run 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. If \( A \) can be partitioned into three sets \( A_1, A_2, \) and \( A_3 \) such that \( A_1 \) and \( A_2 \) are ordered \( (a_1 \prec a_2) \) for all \( a_1 \in A_1 \) and \( a_2 \in A_2 \), then we recursively find relaxations \( R_k = (A_k, \prec_k) \) for \( k \in \{1, 2, 3\} \) and combine them into relaxation \( R = (A, \prec') \) where we keep the constraints between \( A_1 \) and \( A_2 \) but remove all constraints between \( A_3 \) and \( A_1 \cup A_2 \), i.e., \( \prec' = (\bigcup_{k=1}^3 \prec_k) \cup (A_1 \times A_2) \).

Now \( \ell(R) = \left( |A_1| \ell(R_1) \right) \ell(R_2) \ell(R_3) \). Our partition finding attempts to maximize the heuristic \( \left( |A_1| \ell(R_1) \right) \ell(R_2) \ell(R_3) \), which is inversely correlated with the number of linear extensions if we assume that the relaxation for all the parts \( A_k \) is the empty partial order. For every \( x \in A_i \), we use greedy hill climbing, starting from \( A_1 = \{ x \} \) and adding more vertices to \( A_1 \) while maintaining \( A_2 \) as the maximal set such that \( A_1 \) and \( A_2 \) are ordered.

6. We relax \( P \) by taking a spanning tree of its cover graph and then apply the \( O(n^2) \) time algorithm. We sample 6 minimum-weight spanning trees, with the arcs of the cover graph sorted randomly by weight, and choose the one with the smallest number of linear extensions.

4 The Sampler

As the last remaining piece of relaxation Tootsie Pop algorithm, we consider the problem of sampling from sets of type \( A_3 \) defined in equation (1). More exactly, we consider the more general problem of sampling uniformly from the set \( \Omega = \{ x \in [0, 1]^n \mid x_i - x_j \leq s_{ij} \text{ for all } 1 \leq i < j \leq n \} \), where \( s_{ij} \geq 0 \) for all \( 1 \leq i < j \leq n \).

4.1 Continuous Relocation Chain

To sample from \( \Omega \), we define the continuous relocation Markov chain \( (X_k)_{k=0}^{\infty} \) in \( \Omega \) in which each state transition is given by the rule \( \tilde{X}_{k+1} = \phi(X_k, i_k, t_k) \), where dimension \( i_k \in [n] \) and position \( t_k \in [0, 1] \) are sampled uniformly at random, and the transition function \( \phi : \Omega \times [n] \times [0, 1] \to \Omega \) maps \((x, i, t)\) to \((y, t')\) as follows:

- Construct the proposition state \( y' \) by setting \( y'_i = t \) and \( y'_j = x_j \) for all \( j \in [n] \setminus \{ i \} \).
- Move to state \( y = y' \) if \( y' \in \Omega \); otherwise stay in \( y = x \).

See Figure 3 for an example. From the symmetry of the transition distributions we get the following result:

Theorem 2. The uniform distribution on \( \Omega \) is a stationary distribution for the continuous relocation chain.

4.2 Perfect Sampling

Although by Theorem 2 the stationary distribution of the chain is uniform, we do not have a bound for its mixing time, and so we do not know how many iterations are required for the desired precision. Nevertheless, we next see that one can sample exactly from the uniform distribution using monotonic coupling from the past [Propp and Wilson, 1996]. For this we need the chain to be monotonic in relation to the update function \( \phi \) and some partial order \( \subseteq \) on \( \Omega \):
Lemma 1. Let $x \sqsubseteq y$ if $x_i \leq y_i$ for all $i \in [n]$ and $x \neq y$. 

(a) The relation $\sqsubseteq$ is a partial order on $\Omega$;
(b) there exist $0, 1 \in \Omega$ such that $0 \sqsubseteq x \sqsubseteq 1$ for all $x \in \Omega$;
(c) if $x, y \in \Omega$ such that $x \sqsubseteq y$, then $\phi(x, i, t) \sqsubseteq \phi(y, i, t)$ for all $i \in [n]$ and $t \in [0, 1]$.

Proof. It is easy to verify that $\sqsubseteq$ is a partial order and that $0 = (0, 0, \ldots, 0)$ and $1 = (1, 1, \ldots, 1)$ satisfy the conditions in (b). Assume that $x, y \in \Omega$ and $x \sqsubseteq y$. To prove that $\phi(x, i, t) \sqsubseteq \phi(y, i, t)$, it suffices to prove that $\phi(x, i, t_i) \leq \phi(y, i, t_i)$, because for all $j \in [n] \setminus \{i\}$ we have that $\phi(x, i, t) \leq \phi(y, i, t)$. For each $z \in \{x, y\}$ let

$$L_z = \max \left(\{0\} \cup \{z_j - s_{ji} \mid 1 \leq j < i\}\right),$$
$$R_z = \min \left(\{1\} \cup \{z_j + s_{ij} \mid i < j \leq n\}\right).$$

By the definition of $\Omega$ we have that $z_i \in [L_z, R_z]$ and if $t \in [L_z, R_z]$, then $\phi(z, i, t) = t$ and otherwise $\phi(z, i, t) = z_i$. Because $x \sqsubseteq y$ we also have that $L_x \leq L_y$ and $R_x \leq R_y$.

We prove that $\phi(x, i, t_i) \leq \phi(y, i, t_i)$ in cases: First, if $t \in [L_x, R_x]$, then $t \in [L_y, R_y] \cap [L_x, R_x]$ and thus $\phi(x, i, t_i) = t = \phi(y, i, t_i)$. Second, if $t < L_y$ (case $t > R_x$ is symmetric), then $\phi(x, i, t_i)$ is either $x_i$ or $t$. It holds that $x_i \leq y_i$ because $x \sqsubseteq y$ and $t < L_y \leq y_i$ because $y_i \in [L_y, R_y]$. Combined, these yield that $\phi(x, i, t_i) \leq \phi(y, i, t_i)$.

The method of Propp and Wilson [1996] now yields the following algorithm: We run two instances $(X_k^0)_{k=0}^{\infty}$ and $(Z_k^0)_{k=0}^{\infty}$ of the continuous relocation chain for $T > 0$ iterations starting from time $-T$ with initial states $X_0 = 0$ and $Z_0 = 1$. We use the same sequences of parameters $i_k$ and $t_k$ for both chains. If $X_0 = Z_0$, then $X_0$ is the output sample, and otherwise we rerun the whole algorithm with the value of $T$ doubled (i.e., starting from further in the past) still using the same parameters $i_k$ and $t_k$. After the algorithm has terminated, we know that if we started the chain $(Y_k^0)_{k=0}^{\infty}$ from any initial state and used the same parameters $i_k$ and $t_k$, the output $Y_0$ would match the output $X_0$ of the algorithm because by Lemma 1 we have that $X_{-T} \subseteq Y_{-T} \subseteq Z_{-T}$ and inductively $X_k \subseteq Y_k \subseteq Z_k$ for all $-T + 1 \leq k \leq 0$, yielding $X_0 \subseteq Y_0 \subseteq X_0$. By considering a chain $(Y_k^0)_{k=0}^{\infty}$ where the initial state $Y_{-T}$ is sampled uniformly at random from $\Omega$, we see that the distribution of the output $X_0 = Y_0$ is also uniform on $\Omega$ because it is a stationary distribution of $\Omega$.

While we do not know a good upper bound for the expected running time of the algorithm, we can show that it is finite.

Theorem 3. The monotonically coupled-from-the-pass algorithm for the continuous relocation chain outputs a sample from the uniform distribution on $\Omega$ in finite expected time.

Proof. Let $c = 2n - 1$ and define the event $A_l$ by $(l-c, l-c+1, \ldots, l-c+1) = (1, 2, \ldots, n, n-1, \ldots, 1)$ and $t_k \in [(i_k - 1)/n, i_k/n]$ for all $-lc \leq k < -(l-1)c$. We see that if on some iteration $T \geq lc$ an event $A_l$ occurs, then $X_{-(l-1)c} = Z_{-(l-1)c}$, which means that the algorithm terminates after that iteration. All the events $A_1, A_2, \ldots$ are mutually exclusive and have the same probability $p > 0$.

Let $L$ be the smallest $l$ such that the event $A_l$ occurs. Because $T$ is doubled on every iteration, at least half of the time is spent in the last iteration, and thus the running time is $O(L)$ for constant $n$. Because $L$ is geometrically distributed with parameter $p$, its expected value is $1/p < \infty$.

Our algorithm is similar to Huber’s [2014] sampler. The main difference is that whereas our method resamples $x_i$ along a randomly chosen dimension from the whole set $[0, 1]$, rejecting the moves that fall outside, that method uses Gibbs sampling, meaning that it samples uniformly at random from only the set of allowed values. Even though this is beneficial in the sense that the chain stalls less in the same state, the downside is that the sampling never terminates (i.e., reaches $X_0 = Z_0$), and therefore additional tricks, which do not work with generalized order constraints, are required.

5 Experiments

We empirically evaluated the following schemes:

**Telescopic Product:** The scheme of Brightwell and Winkler [1991], with optimizations [Talvitie et al., 2018].

**Extension Tootsie Pop (ETP):** The scheme based on TPA due to Banks et al. [2017].

**Adaptive Relaxation Monte Carlo (ARMC):** The Monte Carlo scheme based on exponential-time counting by Talvitie et al. [2018].

**SAT #1 and #2:** Reducing the problem into SAT model counting using encodings given in Section 2.4 and solving it using sharpSAT [Thurley, 2006] or D4 [Lagniez and Marquis, 2017] exact model counter or ApproxMC2 approximate model counter [Chakraborty et al., 2016].

**(Trivial) Relaxation Tootsie Pop (RTP):** The TPA based scheme given in Section 3. For comparison, we also consider a version with the trivial relaxation (empty poset).

We adopted the experimental setup and benchmark instances of Talvitie et al. [2018]: The instances are randomly generated and include posets of average indegree $k \in \{3, 5\}$, bipartite posets of density $p \in \{0.2, 0.5\}$, and posets extracted from subgraphs of the networks Andes, Diabetes, Link, Munin, and Pigs from the Bayesian Network Repository. We ran each algorithm for each instance for at most 24 hours of CPU time and memory limited to 8 gigabytes. All the algorithms were instantiated to produce a (1, 1/4)-approximation.

We use the implementations due to Talvitie et al. [2018] for the telescopic product, extension Tootsie Pop and adaptive relaxation Monte Carlo methods. We will make our implementation for relaxation Tootsie Pop and the SAT encoding generators publicly available.

Figure 4 shows the results of the experiments. The growth of the running time of RTP clearly resembles polynomial growth, as opposed to the exponential growth of the running time of ARMC. While ARMC is faster in small cases, when the number of elements exceeds roughly 200 elements, RTP becomes the fastest scheme in most instance classes. The version of RTP that uses the trivial relaxation is already faster.
Figure 4: The running times of the algorithms as functions of the number of elements. Each marker shows the median running time and the surrounding shaded area is the range of variation among the five runs. The SAT-based results are for encoding #1, as it was found to always perform better than encoding #2. Both axes are logarithmic.

than the telescopic product scheme and ETP, and using proper relaxation instead of the trivial relaxation improves the running time roughly by an order of magnitude. The performance of the SAT based schemes is poor: they take hours to run already in the smallest cases.

6 Discussion

The relaxation Tootsie Pop scheme is a novel application of TPA [Huber and Schott, 2010] for approximate counting of linear extensions. Our empirical results suggest that the scheme is the first that scales to large instances in practice. In contrast to the exponential-time ARMC scheme, relaxation Tootsie Pop requires only little memory and enables efficient large-scale parallelization (not demonstrated here). In applications like Bayesian network structure learning [Niinimäki et al., 2016], the new scheme should enable the handling of partial orders (i.e., networks) with several hundreds of elements, which are beyond the reach of ARMC; for smaller partial orders ARMC is superior, though.

Our key technical contribution was the perfect sampler in the constrained continuous space. We observed that, even if the sampler does not have—or we do not know how to prove—polynomial worst-case time complexity bounds, the sampler runs fast on all tested benchmark instances. This success should encourage working on heuristic designs of perfect samplers also for other counting and sampling problems. Compared to theoretical works that are concerned with provable upper bounds, the heuristic approach gives access also to samplers whose complexity is difficult to analyze.

For the poor performance of the SAT solver based schemes we offer three possible explanations. First, our encodings of the problem were straightforward and could potentially be beaten by more sophisticated encodings. Second, while the tested algorithms represent the state of the art, other related schemes [Gomes et al., 2006; Zhao et al., 2016] could be more efficient in the present application. Third, the idea of
reducing a counting problem to the NP-hard CNF-SAT problem may be ill-suited in general—even if powerful solvers exist—when the counting problem is “easy” in the sense that it admits a fully polynomial-time approximation scheme.

Acknowledgments

This work was supported in part by the Academy of Finland, under Grants 276864, 303816 and 284642.

References


